MASTER EN SCIENCES INFORMATIQUES

Étude, conception et réalisation d'un ordinateur pédagogique

Vendrix, Johnny

Award date:
1981

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

ETUDE, CONCEPTION ET REALISATION

D'UN ORDINATEUR PEDAGOGIQUE.

Mémoire présenté par

Johnny Vendrix

en vue de l'obtention
du titre de
Licencié et Maître en Informatique.

Année académique 1980-1981

"Avant d'introduire ce travail, je tiens à remercier Monsieur le professeur J.-L. HAINAUT, promoteur de ce mémoire, pour les conseils et les encouragements qu'il m'a prodigués.

Je remercie R. VERHAEGHE et J.P. ADANS pour leur aide.

Je remercie vivement tout mon entourage qui m'a permis de mener à bien ce travail."

TABLE DES MATIERES.

INTRODUCTION.

1. Préambule.

Malgré sa présence dans tous les domaines, l'informatique est encore chargée de mystère. Pourtant, un ordinateur n'est qu'un outil comme un autre.

C'est pourquoi, comprendre le fonctionnement d'un ordinateur, les langages qu'il utilise, les relations entre son architecture interne et un programme, permet la démystification de cet outil informatique.

Une introduction simple à l'informatique doit permettre à chacun de mieux comprendre cette nouvelle technologie.

2. Définition d'un ordinateur pédagogique.

Un ordinateur pédagogique est un outil pédagogique utilisé pour comprendre l'informatique.

Il illustre l'architecture et l'organisation de base de tous les ordinateurs.

Il aide à comprendre l'organisation interne et le fonctionnement de l'ordinateur.

L'ordinateur pédagogique est un simulateur d'un ordinateur.

Il doit pouvoir permettre :

- l'initiation à la programmation
  1. assemblages et corrections conversationnels de programmes dans un langage simple,
  2. simulation des programmes;
- l'utilisation de programmes conversationnels divers tel le programme d'édition.

3. Objectif du mémoire.

L'objectif de ce mémoire consiste en la réalisation de l'étude, de la conception et de l'implémentation d'un ordinateur pédagogique disposant d'un simulateur graphique.

Le chapitre 1 présente deux ordinateurs "pedagogiques" qui ont servi comme ingrédients de base à l'ordinateur pédagogique proposé.

Le chapitre 2 décrit l'ordinateur pédagogique proposé.

Les chapitres 3 et 4 définissent la réalisation et l'implémentation de l'ordinateur pédagogique.

Le chapitre 5 propose des solutions pour la maintenance de la réalisation actuelle.

Le chapitre 6 présente brièvement les problèmes rencontrés durant l'élaboration de ce mémoire.

1. PRESENTATION D'ORDINATEURS PEDAGOGIQUES EXISTANTS.

1.1. MIX.

MIX est une machine imaginaire qui a été conçue pour expliquer le fonctionnement d'un ordinateur sans tenir compte d'un ordinateur particulier ni d'un langage spécifique.

Cette machine possède une structure proche des ordinateurs existants. Le langage MIX est son langage machine.

Les caractéristiques principales sont :

- une mémoire de 4000 mots;
- un mot possède 6 bytes, le premier représentant le signe et les autres contiennent une valeur décimale comprise entre 0 et 64;
- 9 registres (accumulateur, registre extension, registres d'index, registre de saut);
- un indicateur d'overflow;
- un indicateur de comparaison;
- une unité d'entrée;
- une unité de sortie.

Cette machine est intéressante mais pas très appropriée pour une simulation graphique. En effet chaque mot occupe 12 caractères sur un graphique. De plus, il est aussi impossible de garder un nombre important de registres.

1.2. Machine implémentée sur TRS80.

Cette machine, qui répond le mieux à un objectif pédagogique, a été développé par Mr. J.-L. HAINAUT (F.N.D.P à NAMUR).

L'architecture et la description du langage de commandes sont presques semblables à celles qui seront developpées plus loin.

Malheureusement, cet ordinateur pédagogique ne procure pas aux futurs utilisateurs des facilités pour la création et la modification de programmes. Ce qui pouvait nuire à l'application de l'ordinateur pédagogique.

2. PRESENTATION DE L'ORDINATEUR PEDAGOGIQUE.

2.1. Introduction.

L'objectif pédagogique poursuivit par ce système est de communiquer des connaissances d'informatiques à l'utilisateur afin qu'il les assimile, les retienne et qu'il puisse les appliquer.

1. La communication est assurée grâce à l'utilisation :

   - d'un schéma graphique représentant un ordinateur fictif;
   - d'un langage de communication simple.

2. L' assimilation est facilitée grâce à l'existence d'une simulation graphique montrant le fonctionnement de l'ordinateur.

3. L' application est simple car l'utilisateur dispose :

   - d'un ensemble de commandes lui permettant la manipulation de l'ordinateur;
   - d'un ensemble d'instructions grâce auxquelles il pourra réaliser des programmes exécutables directement ou indirectement par l'ordinateur;
   - d'un éditeur de texte qui permettera la création ou modification des textes (programmes,donnes,...).

Nous allons décrire par la suite les moyens de communication et d'application offerts à l'utilisateur.

2.2. La communication.

2.2.1. Introduction.

Le rôle de la communication est de transmettre des informations d'un émetteur vers un récepteur.

Cette communication subit souvent des perturbations, des déformations et des distorsions, c'est pourquoi il faut essayer d'avoir

- un langage clair et simple;
- une forme de message simple.

L'utilisation d'un schéma graphique est intéressante car le graphisme est une langue universelle.

De plus, l'utilisation d'un langage de communication simple permet une assimilation plus rapide.

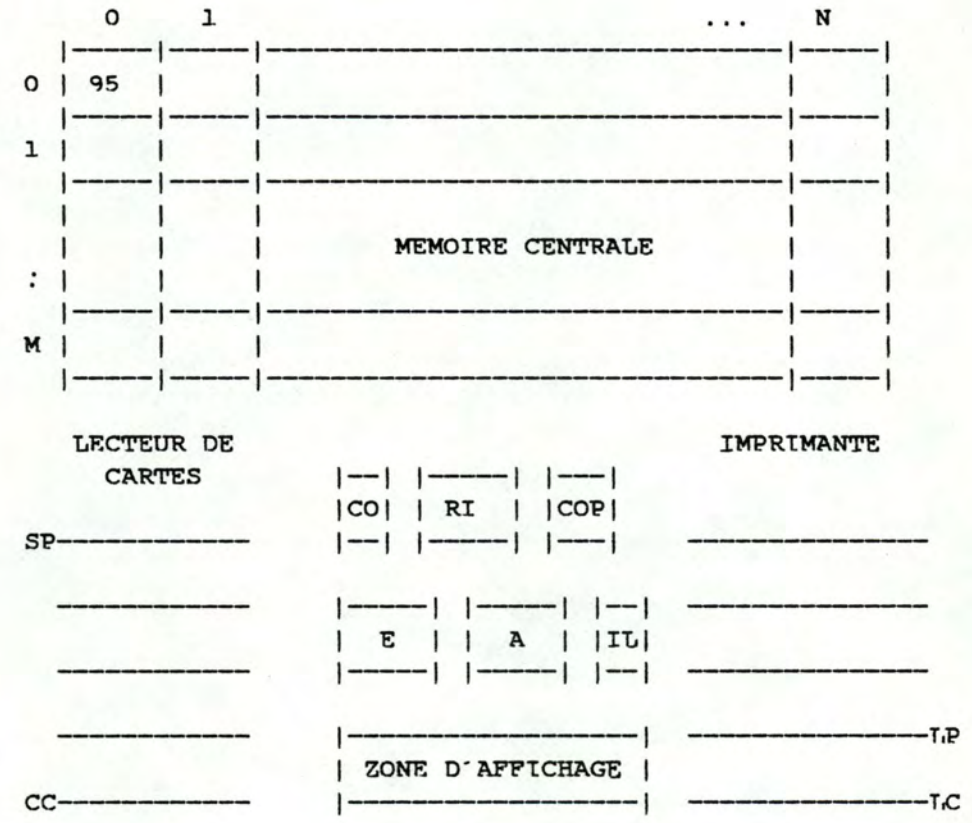
2.2.2. Présentation de la machine.

2.2.2.1. Introduction.

La machine est représentée par un graphique. Ce dernier doit être didactique c'est-à-dire simple et clair.

"UNE IMAGE VAUT MILLE MOTS" (Proverbe Chinois)

2.2.2.2. Description du shéma graphique.



légende :

CC = Carte courante	RI = Registre Instruction
SP = Sommet du Paquet	COP = Code opératoire
LC = Ligne courante	E = registre Extension
LP = Ligne précédente	A = Accumulateur
CO = Compteur ordinal	IL = Indicateur Logique

L'architecture est simple mais suffisamment complète . Elle comprend :

- L'unité centrale et la mémoire centrale;
- les périphériques (unité d'entrée et de sortie).

On remarque aussi la présence

- d'une zone d'affichage qui est une zone de communication entre l'utilisateur et le système;
- d'une zone (COP) contenant le code mnémonique de l'opérateur pour l'instruction en cours d'exécution.

Ces 2 zones ne sont pas accessibles par l'utilisateur.

2.2.2.3. Description de la mémoire centrale et de l'unité centrale.

2.2.2.3.1. La mémoire centrale.

La mémoire centrale est composée d'un ensemble de cellules qui peuvent contenir chacune une unité d'information (ou unité de donnée). La quantité d'unité de donnée stockable dans la mémoire est déterminée par la capacité mémoire qui s'exprime en mots. Chaque cellule est identifiée par une adresse.

2.2.2.3.2. L'unité Centrale.

Elle comprend :

- une unité de commande;
- une unité arithmétique et logique.

1. L'unité de commande.

C'est l'unité qui commande l'ordinateur; on y trouve :

- un compteur ordinal (compteur d'adresses ou compteur d'instructions);
- un registre d'instructions.

Le compteur d'instructions reçoit une valeur représentant l'adresse d'une cellule en mémoire centrale.

Exemple :

(voir figure page avant)
la valeur 95 occupe
la cellule d'adresse 0

Le registre d'instructions contient une information représentant une instruction ou commande.

2. L'unité arithmétique et logique.

Cette unité se charge des opérations arithmétiques et des comparaisons,elle se compose :

- d'un registre extension;
- d'un accumulateur;
- d'un indicateur logique.

Le registre extension possède 2 fonctions :

- il recoit les débordements provenant de l'exécution d'une opération arithmétique;
- il sert de registre auxiliaire.

Son contenu peut être protegé par inhibition des débordements.

L'accumulateur contient soit l'opérande principal d'une instruction, soit le résulat d'une opération arithmétique ou autre.

L'indicateur logique décrit le résultat de certaines opérations telles que la comparaison, les opérations arithmétiques, la lecture de cartes.

Elle peut prendre les valeurs suivantes

"=" signifiant égalité ou = 0
"<" signifiant plus petit ou < 0
">" signifiant plus grand ou > 0
"OV" signifiant débordement ou fin de carte
"ER" signifiant erreur de lecture

2.2.2.3.3. Description de l'information contenue dans l'ordinateur.

Une information est représentée par un nombre entier digital précédé éventuellement d'un signe.

Si la longueur d'un mot est de 6 caractères alors tout information contenue dans une unité de donnée sera comprise dans l'intervalle |-|9|9|9|9|9| et | |9|9|9|9|9| le 1er caractère du mot représentant le signe ("-" pour négatif et " " pour positif).

Dorénavant on considèrera que la longueur du mot est de 6 caractères.

Selon l'utilisation qu'on en fait, ce mot sera interprété comme :

- une valeur numérique;
- une suite de caractères;
- une instruction.

1. Une valeur numérique : la valeur contenue dans le mot = valeur numérique.

Exemple

| |0|0|0|5|9| = 59

2. Une suite de caractères : chaque caractère possède une valeur codée (voir tableau de codification des caractères).Chaque caractère y est représenté par une suite de 2 chiffres décimaux ainsi "A" est représenté par la valeur 65 et "B" par la valeur 66.

Un mot comprenant 5 chiffres numériques ne peut contenir que 2 caractères au maximun. Ceux-ci sont alignés à droite dans le mot.

Exemple :
| |0|6|6|6|5| représente les 2 caractères "BA"

Le tableau de codification qui suit reprend des

- caractères normaux,usuels;
- caractères spéciaux (+,-,.,...);
- caractères fonctions.

Tableau de codification des caractères.

Code	caractère	Code	caractère	Code	caractère
00	sans effet	51	3	75	K
01	tabulation	52	4	76	L
..24	de 1 à 54	53	2	77	M
25	écrire de 2	54	6	78	N
..31	à 8 espaces	55	7	79	O
32	espace	56	8	80	P
33	!	57	9	81	Q
34	"	58	:	82	R
35	#	59	;	83	S
36	$	60	<	84	T
37	%	61	=	85	U
38	&	62	>	86	V
39	'	63	?	87	W
40	(	64	@	88	X
41	)	65	A	89	Y
42	*	66	B	90	Z
43	+	67	C	91	curseur
44	,	68	D	92	sans
45	-	69	E	..97	effet
46	.	70	F	98	ligne suivante
47	/	71	G	99	page suivante
48	0	72	H		
49	1	73	I		
50	2	74	J		

3. Une instruction : nous examinerons ce point par la suite.

2.2.2.4. Description des périphériques.

Les périphériques sont au nombre de 2 :
- un lecteur de cartes;
- une imprimante.

2.2.2.4.1. Le lecteur de cartes.

Il se compose :
- d'un compartiment de cartes;
- d'une tête de lecture.

1. Le compartiment de cartes contient les prochaines cartes qui seront lues.
   Une carte peut contenir 20 caractères et peut être lue suivant 2 modes :
   - numérique : la carte ne peut contenir que des chiffres, des espaces et des signes "-". Elle est interprétée comme contenant une suite de valeurs numériques;
   - alphabétique : la carte peut contenir n'importe quel caractère existant dans le tableau de codification des caractères.

2. La tête de lecture est dirigée sur la carte courante qui sera lue la première.

   Il est à remarquer que 5 cartes seulement sont visibles.

2.2.2.4.2. L'imprimante.

L'impression se fait sur un listing. Une page (5 lignes) est affichée à l'écran.
Chaque ligne peut contenir 24 caractères.
L'imprimante possède une tête d'écriture qui, en période d'inactivité, reste à la suite du dernier caractère ou à la position de tabulation indiquée.

L'écriture peut se faire suivant 2 modes :

- numérique : l'imprimante écrit les nombres sur 6 positions (avec cadrage à droite) comptées à partir de la position courante de

la tête. S'il ne reste pas assez de d'espace pour écrire un nombre, celui-ci est écrit au début de la ligne suivante;

- alphanumérique : l'imprimante reçoit des codes numériques qu'elle interprète. En effet ces caractères correspondent à des caractères imprimables (qui sont imprimés) ou à des commandes (qui sont exécutées) ou peuvent aussi ne pas être reconnus.
  Ces commandes concernent :

  - le saut de page;
  - le passage à la ligne suivante;
  - l'espacement;
  - la tabulation (positionnement de la tête d'écriture)

2.2.2.5. Le langage machine.

2.2.2.5.1. Conventions.

- Les registres sont désignés par un nom (voir description graphique);

- Une cellule de la mémoire centrale est désignée par son adresse : 0,2,10,...;

- Le contenu d'un registre ou d'une cellule sera désigné par la mise entre parenthèses de ce registre ou de cette cellule : (0),(2),(A),...;

- Si le contenu d'un registre ou d'une cellule est une adresse valide, le contenu de la cellule portant cette adresse sera désigné par la mise entre parenthèses de la désignation du contenu de ce registre ou de cette cellule : ((0), ((2)), ((A)),...;

- L'expression : d <--- exp
  où "d" est la désignation du contenu d'un registre ou d'une cellule;
  où "exp" est la représentation d'une expression numérique;
  s'interprète comme suit :
  évaluer l'expression "exp" et ranger la valeur obtenue dans le registre ou dans la cellule "d".

Exemples :

A <--- 12 : ranger 12 dans l'accumulateur;
20 <--- (A) : ranger le contenu de l'accumulateur dans la cellule 20;
A <--- ((0)) : ranger le contenu de la cellule, dont l'adresse se trouve dans la cellule 0, dans l'accumulateur.

2.2.2.5.2. Introduction.

Le langage machine est le seul langage compris directement par l'unité centrale de la machine.

Son alphabet, qui se compose des symboles "-", " ", "1", "2", "3", "4", "5", "6", "7", "8", "9", "0", permet la représentation :

- des caractères (voir tableau de codification des caractères p. 13);
- des nombres;
- des instructions.

2.2.2.5.3. Structure d'une instruction machine.

Une instruction machine, représentant une opération, se subdivise en 2 parties : l'opérande et l'opérateur.

1. L'opérateur est constitué de 2 champs :

   - champ 1 , qui spécifie le groupe auquel appartient l'instruction. Dans notre cas, ce champ se compose d'un chiffre qui prend la valeur

     1 pour transfert de valeurs;
     2 pour opération arithmétique;
     3 pour opération logique;
     4 pour branchement;
     7 pour lecture sur périphérique;
     8 pour écriture sur périphérique;
     9 pour groupe divers.

   - champ 2 , qui localise l'instruction dans le groupe.

2. L'opérande est aussi constitué de 2 champs :

- champ 3 ,qui spécifie le mode d'adressage. Dans notre cas, il se compose d'un chiffre qui prend la valeur

  0 pour adressage immédiat;
  1 pour adressage direct;
  2 pour adressage indirect;
  3 pour registre E;
  4 pour adressage indirect avec pré-décrémentation;
  5 pour adressage indirect avec post-incrémentation.

- champ 4, qui spécifie l'argument éventuel de l'instruction.

2.2.2.5.4. Les modes d'adressage.

La plupart des instructions travaillent sur 1 ou 2 arguments dont l'un des deux est souvent implicite (registre A, registre CO); l'autre est désigné explicitement par les champs 3 et 4.

Appelons nn le champ 4.

1. Adressage immédiat (champ 3 = 0).

nn est l'argument lui même. Sa valeur est donc limitée à l'intervalle des entiers [0,9..9]. La valeur maximun dépend de la taille du champ 4.
Supposons-la égale à 2, dans ce cas l'intervalle sera [0,99].

Exemple :
20034
2 = champ 1 (opération arithmétique)
0 = champ 2 (addition)
0 = champ 3
34= champ 4
qui équivaut à ajouter 34 au contenu de l'accumulateur
soit : A <--- (A) + 34.

2. Adressage direct (champ 3 = 1)

nn est l'adresse d'une cellule de la mémoire centrale. L'argument est le contenu de cette cellule, soit (nn).

Exemple :

20134

équivaut à ajouter le contenu de la cellule 34 au contenu de l'accumulateur
soit : A <--- (A) + (34).

3. Adressage indirect (champ 3 = 2)

nn est l'adresse d'une cellule dont le contenu est l'adresse de la cellule qui contient l'argument. L'argument est donc ((nn)).

Exemple :

20234

(en supposant que la cellule 34 contienne 47)
équivaut à ajouter le contenu de la cellule 47 au contenu de l'accumulateur
soit : A <--- (A) + ((34)) .
(34) = 47

4. Registre E (champ 3 = 3)

nn est indifférent. L'argument est le contenu de E.

Exemple :

20300

équivaut à ajouter le contenu de E à celui de A
soit : A <--- (A) + (E).

5. Adressage indirect avec pré-décrémentation automatique (champ 3 = 4)

Il s'agit d'un adressage indirect réalisé après décrémentation du contenu de la cellule nn.

Exemple :

20434

avec (34) = 47

équivaut à décrémenter le contenu de la cellule 34 (qui contient donc 46) puis ajouter le contenu de la cellule 46 au contenu de l'accumulateur
soit : 34 <--- (34) - 1
A <--- (A) + ((34)).

6. Adressage indirect avec post_incrementation automatique (champ 3 = 5)

Il s'agit d'un adressage indirect suivi de l'incrémentation du contenu de la cellule nn.

Exemple :

20534

avec (34) = 47

équivaut à ajouter le contenu de la cellule 47 au contenu de l'accumulateur puis incrémenter le contenu de la cellule 34 (qui contient alors 48)

soit : A <--- (A) + ((34))
34 <--- (34) + 1.

Remarque : dans ces 2 derniers modes, l'incrément est le nombre de cellules traitées de la mémoire centrale qui ont été traitées par l'instruction. Dans certains cas, cet incrément est différent de 1.

2.2.2.5.5. Description du langage machine.

La description détaillée de ce langage machine n'est pas réalisée ici car on montrera par après un langage plus "evolué", qui sera employé par les utilisateurs, le langage Assembleur. Une table de correspondance entre les instructions machine et les instructions Assembleur suivra cette description.

2.2.3. Le langage de communication.

2.2.3.1. Introduction.

L'utilisateur communique avec le système grâce à un langage de commandes. Ces commandes lui permetteront de faire exécuter des travaux disponibles dans le système.

Le terminal est l'outil de dialogue entre le système et l'utilisateur. Il se compose d'un clavier et d'un écran.

- le clavier permet à l'utilisateur d'introduire des informations (données, commandes, instructions,...); certaines touches peuvent avoir des fonctions spéciales :

  <DEL> représente la touche du clavier qui efface le caractère précédent la position du curseur (ce dernier représente la tête d'écriture et de lecture sur un terminal);

  <RETOUR> représente la touche du clavier qui signale la fin du message introduit par l'utilisateur.

- L'écran contient la zone d'affichage. Celle-ci remplit l'écran totalement ou partiellement, selon le type de travail que l'utilisateur effectue.

  La zone d'affichage visualise les informations introduites par l'utilisateur et celles que le système envoie vers l'utilisateur (messages d'erreurs, message d'attente,..).

2.2.3.2. Description du langage de commandes.

2.2.3.2.1. Introduction.

Les messages de communication de ce langage sont simples mais suffisamment explicites pour l'utilisateur.

Chaque message utilisateur possède la structure suivante

| opérateur | opérande(s) |

mais dans certains cas l'une ou l'autre des parties ou parfois même les deux ne sont pas nécessaires.

L' opérateur est représenté par un nom mnémonique qui rappelle la fonction qui est demandée. Ce nom est court (1 ou 2 caractères), réduisant les erreurs de frappe.

Les opérandes sont souvent réduites et parfois même inexistantes.

Le langage de commandes peut être décomposé en 3 parties:

- langage de commandes du Superviseur;
- langage de commandes du Moniteur;
- langage de commandes de l'Editeur-Assembleur.

Le superviseur a pour fonction de contrôler le système; le moniteur a pour rôle la gestion de l'ordinateur; l'Editeur-assembleur a pour rôle de faciliter la tâche de l'utilisateur dans la création ou la modification du texte et dans l'écriture des programmes.

Chaque commande sera décrite de la manière suivante :

- le nom de la commande avec son code mnémonique;
- le format de la commande c'est-à-dire la forme correcte de la commande que l'utilisateur devra introduire;
- la définition c'est-à-dire ce que la commande signifie et ce qu'elle entraîne;
- l'erreur éventuelle causée par l'introduction d'un opérande incorrect.

2.2.3.2.2. Définitions des concepts utilisés dans les parties qui vont suivre.

- texte : un texte représente une suite de lignes;
- ligne : une ligne représente une suite de caractères;
- fichier : un fichier est une collection d'informations identifiables par un nom;
- espace de travail : un espace de travail représente une zone de la mémoire du système;
- numéro de ligne : un numéro de ligne représente la position d'une ligne par rapport au autre lignes du texte;
- texte courant : le texte courant représente le dernier texte traité entre l'entrée de l'utilisateur dans le système et l'introduction de la dernière commande;
- ligne courante : la ligne courante représente la dernière ligne traitée du texte courant;
- curseur : le curseur représente la position où le prochain caractère va être affiché à l'écran"; il sera représenté par le symbole "[];
- attente du système : l'attente du système détermine le moment où le système attend que l'utilisateur introduise une information; elle sera représenté par le symbole "?".

2.2.3.2.3. Langage de commandes du superviseur.

Ce langage est constitué de 3 commandes, celles-ci peuvent être employées n'importe quand par l'utilisateur.

commande 1:

nom : E (Editeur, assembleur)
format : E
définition : La commande E provoque l'appel du module Editeur_assembleur;

commande 2:

nom : M (Moniteur)
format : M
définition : La commande M provoque l'appel du module Moniteur;

commande 3:

nom : Q (Quitter)
format : Q
définition : La commande Q provoque la clôture des opérations et la sortie du système pour l'utilisateur.

2.2.3.2.4. Langage de commandes de l'éditeur assembleur.

1. Introduction.

L'éditeur de texte est un programme qui donne à l'utilisateur les moyens de créer ou de modifier un texte;

L'assembleur est un programme qui analyse et traduit en langage machine un programme écrit dans un langage "évolué" c'est-à-dire un langage facilement assimilable par l'homme.

2. L'éditeur.

a) Introduction.

L'utilisateur qui travaille avec l'éditeur doit avant tout préciser s'il désire créer un texte ou modifier un texte existant.

Cette précision est nécessaire car les commandes de l'éditeur n'ont d'effet que sur un espace de travail; ce dernier devra contenir :
- en cas de modification, le texte à modifier;
- en cas de création, un espace vierge.

b) Description des commandes de l'éditeur.

Convention : Les commandes sont accompagnées parfois d'indications optionnelles; ces dernières seront placées entre "{" et "}".

Chargement de l'espace de travail.

nom : L (Load)
format : L <nom du fichier>
définition :
cette commande provoque la copie du contenu du fichier, identifié par <nom du fichier> , dans l'espace de travail de l'éditeur
erreur :
un message d'erreur sera affiché si aucun fichier n'est défini par <nom du fichier>.

Effacement du contenu de l'espace de travail.

nom : R (Rub out)
format : R
définition
Cette ommande provoque l'effacement du contenu de l'espa e de travail de l'éditeur.

Insertion de lignes.

nom : I (Insert)
format : I {<a>} <a>= nombre de lignes
définition :
Cette commande demande l'insertion du nombre de lignes spécifiées par <a> après la ligne courante. Si <a> est absent, cette commande provoque l'insertion d'une seule ligne.
L'insertion se termine lorsque l'utilisateur a introduit le nombre de lignes demandé.
erreur :
un message d'erreur sera affiché si <a> n'est pas un entier strictement positif.
exemple :
On suppose que la ligne courante est la ligne "1".

```
|1        |I 2     |I 2              |2 exemple
|?        |2 []    |2 exemple        |3 d'insertion
|I 2      |        |3 []             |?
```

Destruction de lignes.

nom : D (Delete)
format : D {<a{,b}>}
<a> = numéro de la première ligne à détruire;
<b> = numéro de la dernière ligne à détruire;

définition :
Cette commande provoque la destruction des lignes spécifiées par l'intervalle [a,minimun(b,nombre de lignes existant dans le texte après la ligne)]
Si <b> est absent, seule la ligne <a> est détruite;
si <a> est absent, c'est la ligne courante qui sera prise en considération.

erreur :
un message d'erreur sera affiché si :
- <a>,<b> ne sont pas des entiers strictement positifs;
- <a> est strictement supérieur à <b>;
- <a> n'existe pas dans le texte.

exemple :

```
|3              |?
|4 exemple de   |D 4,5  les lignes
|5 destruction  |[]      4 et 5 sont
|?              |        détruites
|[]             |
```

Affichage de lignes.

nom : P (Print)
format : P {<a{,b}>}
<a> = numéro de la première ligne à afficher;
<b> = numéro de la dernière ligne à afficher.

définition :
Cette commande provoque l'affichage des lignes appartenant à l'intervalle [a,minimun(b,nombre de lignes du texte existant après <a>)].
Si <b> est absent, seule la ligne <a> est affichée;
si <a> est absent, la ligne considérée sera la ligne qui suit la ligne courante.

erreur :
(voir commande précédente)

exmple :

```
|?     |P 3            |?              |P 4,3
|P 3   |3 exemple      |P 4            |ERREUR
|      |?              |4 d'affichage  |?
```

Affichage de la ligne suivante.

nom :
format :
définition :
L'introduction d'une commande vide provoque l'affichage de la ligne qui suit la ligne courante. Un message vide est introduit lorsque l'utilisateur frappe uniquement sur <RETOUR>

erreur :
un message d'erreur sera affiché s'il n'existe pas de nouvelle ligne.

exemple :

```
|3 exemple      |3 exmple
|?              |?
|[]             |
|               |4 d'affichage
|               |?
```

Affichage de la ligne précédente.

nom : - (Minus)
format : -
définition :
Cette commande provoque l'affichage de la ligne qui précède la ligne courante.
erreur :
Une erreur sera affichée si aucune ligne ne précède la ligne courante.
exemple :

```
|2      |?
|?      |-
|-      |1 exemple
```

Sauvetage de l'espace de travail de l'éditeur.

nom : S (Save)
format : S {<nom du fichier>}
définition :
Cette commande provoque la copie du contenu de l'espace de travail dans un fichier sous le nom <nom du fichier>. Si ce dernier est absent, le sauvetage se fait dans le fichier dont le nom a été défini lors du chargenent de l'espace de travail.
erreur :
un message d'erreur sera affiché si <nom du fichier> n'existe pas et que l'espace de travail n'avait pas été chargé auparavant.
remarque :
il faut toujours faire attention dans la spécification de <nom du fichier> car après la copie, seule la nouvelle version sera accessible à l'utilisateur.
exemple :

```
|1 exemple
|2 de
|3 sauvetage
|?
|S AB espace de travail est
       sauvé dans un fichier
       de nom "AB"
```

3. L'assembleur.

a) Introduction.

L'assembleur dispose aussi d'un espace de travail où il va ranger le résultat de la traductiond'un texte se trouvant dans l'espace de travail de l'éditeur.

b) Description des commandes de l'assembleur.

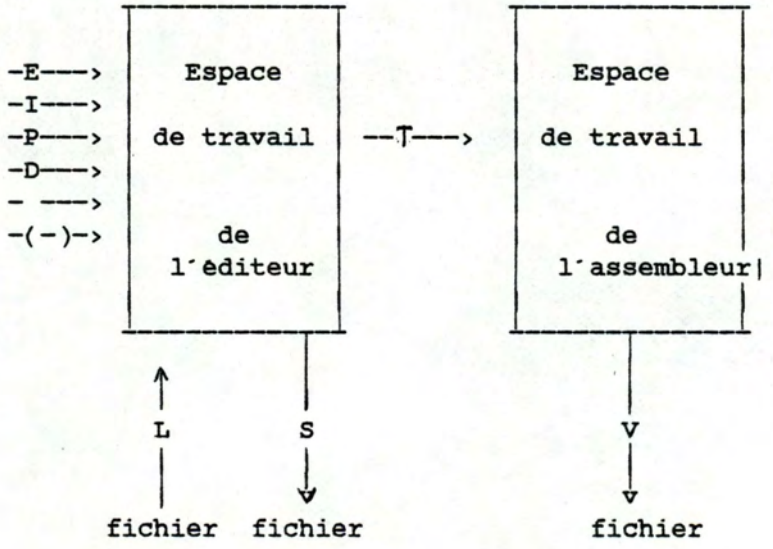
Traduction d'un programme assembleur.

nom : T (Traduction)
format : T
définition :
Le texte contenu dans l'espace de travail de l'éditeur est analysé.
Si le texte représente un programme assembleur correct, la traduction est réalisée et rangée dans l'espace de travail de l'assembleur.
Si le texte n'est pas un programme assembleur correct, chaque ligne incorrecte contenue dans l'espace de travail de l'éditeur est suivie de(s) message(s) d'erreurs associés à la ligne analysée. L'espace de travail de l'assembleur n'est alors pas rempli.
erreur :
un message d'erreur sera affiché si l'espace de travail de l'éditeur est vierge.

Sauvetage de l'espace de travail de l'assembleur.

nom : V (saVe)
format : V ‹nom du fichier›
définition :
Cette commande provoque la copie de l'espace de travail dans un fichier spécifié par ‹nom du fichier›.
erreur :
Un message d'erreur sera affiché si l'espace de travail est vierge ou si aucun ‹nom du fichier› n'a été défini.

4. Représentation des opérations de l'éditeur_assembleur.



2.2.3.2.5. Langage de commandes du moniteur.

2.2.3.2.5.1. Introduction.

Le moniteur est un programme du système qui permet :

- la simulation du fonctionnement d'un ordinateur;
- l'exécution de tâches annexes, comme
  1. l'initialisation, le contrôle et l'arrêt de l'ordinateur pédagogique;
  2. le lancement de l'exécution d'un programme;
  3. le chargement de programmes, de cartes,etc...

L'utilisateur qui se trouve dans ce mode, voit apparaître sur l'écran de son terminal la simulation graphique de l'ordinateur pédagogique.

La zone de communication entre l'utilisateur et l'ordinateur est représentée par une zone d'affichage. (voir schéma page 9).

2.2.3.2.5.2. Description des commandes du moniteur.

1. Initialisation de l'unité centrale.

a) Garnissage de la mémoire centrale.

nom : FM (Fill Memory)
format :
FM ‹adresse dans la mémoire centrale›
‹valeur d'un mot mémoire›
‹RETOUR›
‹X›
définition :
Cette commande demande le remplissage de la mémoire centrale à partir de la cellule mémoire définie par ‹adresse dans la mémoire centrale.

- La valeur ‹adresse dans la mémoire centrale› est rangée dans le compteur d'adresses qui identifie la cellule courante dans la mémoire centrale;
- l'introduction de ‹valeur d'un mot mémoire› entraîne le rangement de cette valeur dans la cellule courante et l'incrémentation de la valeur du compteur d'adresses;
- ‹RETOUR› introduit seul, provoque la modification du contenu du compteur d'adresses qui identifie la cellule suivante;
- le caractère "X" introduit seul, signale la fin de l'opération de remplissage de la mémoire centrale.

erreur :
un message d'erreur d'erreur sera affiché si :

- ‹adresse dans la mémoire centrale› n'est pas l'adresse d'une cellule de la mémoire centrale;
- ‹valeur d'un mot mémoire› n'est pas une valeur stockable dans une cellule de la mémoire centrale.

exemple :

FM 0	adresse cellule courante = 0;
12345	rangement de la valeur "12345" dans la cellule 0;
	adresse cellule courante = 1;
‹RETOUR›	adresse cellule courante = 2;
67890	rangement de la valeur "67890" dans la cellule 2;
	adresse cellule courante = 3;
X	fin remplissage de la zone de la mémoire centrale.

b) Garnissage des registres.

garnissage du regsitre A (Accumulateur).

nom : FA (Fill Accumulator)
format : FA ‹valeur d'un mot mémoire›
définition :
la valeur ‹valeur d'un mot mémoire› est rangée dans l'accumulateur;
erreur :
un message d'erreur sera affiché si ‹valeur d'un mot mémoire› n'est pas stockable dans un registre.

garnissage du registre I (Instruction).

nom : FI (Fill Instruction)
format : FI ‹valeur d'un mot mémoire›
définition :
la valeur ‹valeur d'un mot mémoire› est rangée dans le registre d'instruction.
errreur : voir FA

Garnissage du compteur ordinal (compteur d'adresses).

nom : FC (Fill adresses Counter)
format : FC ‹adresse de la mémoire centrale›
définition :
La valeur ‹adresse de la mémoire centrale› est rangée dans le compteur d'adresse.
erreur :
un message d'erreur sera affiché si ‹adresse de la mémoire centrale› n'est pas l'adresse d'une cellule de la mémoire centrale.

Garnissage du registre d'instruction.

nom : FE (Fill Extension regsiter)
format : FE ‹valeur d'un mot mémoire›
défintion :
la valeur ‹valeur d'un mot mémoire› est rangée dans le registre extension.
erreur : voir FA.

Garnissage de l'indicateur logique.

nom : FL (Fill Logical indicator)
format : FL ‹symbole›
définition :
le symbole ‹symbole› est rangé dans l'indicateur logique.
erreur :
un message d'erreur sera affiché si le symbole n'appartient pas à {<,>,=,OV,ER}.

c) Initialisation de l'unité centrale.

nom : ZM (Zero machine)
format : ZM
définition :
Cette commande provoque :
- la remise à zéro du contenu des registres et de la mémoire centrale;
- l'initialisation de l'indicateur logique, du code opératoire, de l'imprimante et du lecteur de cartes.

d) Chargement de la mémoire centrale.

nom : LM (Load Memory)
format : LM {‹nom du fichier›}
définition :
Cette commande provoque le chargement du contenu du fichier ‹nom de fichier› dans la mémoire centrale de l'ordinateur à l'adresse spécifiée par la valeur contenue dans le compteur d'adresses.
Si ‹nom du fichier› est absent alors c'est le texte existant dans l'espace de travail de l'assembleur qui sera considéré pour le chargement.
erreur :
Un message d'erreur sera affiché si :
- la taille du texte à charger est supérieure à l'espace mémoire existant entre l'adresse de la cellule courante et l'adresse de la dernière cellule de la mémoire centrale;
- le texte contient des caractères non stockables dans la mémoire centrale;
- ‹nom du fichier› n'existe pas et que l'espace de travail de l'assembleur est vierge.

e) Règlage de la vitesse entre chaque instruction exécutée.

nom : SI (Speed Instruction)
format : SI ‹nombre de secondes›
‹nombre de secondes› compris dans [0..99]
définition :
Un délai défini par ‹nombre de secondes› est observé après l'exécution de chaque instruction.
La valeur initiale de ce délai = 0 seconde;
si ‹nombre de secondes› = 99 alors la machine s'arrête après chaque instruction et le passage à l'exécution de l'instruction suivante se fait en frappant sur la touche ‹RETOUR›. Si ‹nombre de secondes› est ‹ 0 alors la valeur considérée sera égale à 0, si ‹nombre de secondes› est › 99 alors la valeur considérée sera égale à 99.

2. Exécution d'une instruction ou d'un programme.

a) Introduction.

Un programme est constitué d'un ensemble d'instructions. Chacunes d'elles définit une opération particulière.

b) Exécution d'une instruction.

nom : EI (Execute Instruction)
format : EI
définition :
Le contenu de la cellule courante est rangé dans le registre instruction, celui-ci est alors décodé puis exécuté.
erreur :
Un message d'erreur sera affiché si :
- l'instruction est invalide;
- l'adresse du compteur d'adresses n'identifie pas une cellule de la mémoire centrale.

c) Exécution d'un programme.

nom : EP (Execute Program)
format : EP {‹c›}
où ‹c› = nombre d'instructions à exécuter;
cette valeur est comprise dans [2..99]
définition :
Cette commande provoque :
- le lancement de l'exécution du programme (ou partie de programme) dont la cellule courante identifie la première instruction à exécuter;
- l'arrêt de l'exécution du programme est realisé lorsque la fin programme a été rencontrée ou que ‹a› instructions ont été exécutées, ce dernier vaut 99 si ‹a› est absent.

3. Sauvetage du contenu de la mémoire centrale.

nom : SM (Save Memory)
format : SM <nom du fichier>,<a>,<b>
<a> = adresse de la première cellule a sauvé,
<b> = adresse de la dernière cellule a sauv´.
définition :
Cette commande provoque la copie du contenu de la mémoire centrale à partir de la cellule <a> jusqu´à la cellule <b> dans un fichier identifié par <nom du fichier>.
erreur :
Un message d´erreur sera affiché si ;
- <a> est supérieur à <b>;
- <a> ou <b> n´est pas une adresse valide.

4. Gestion de l´imprimante.

a) Avance du papier.

nom : NL (Next Line)
format : NL
définition :
Par cette commande, la feuille de l´imprimante est avancé d´une ligne.

b) Sauvetage du contenu de l´imprimante.

nom : SL (Save Line)
format : SL <nom du fichier>
définition :
Cette commande provoque le sauvetage du contenu de l´imprimante dans un fichier défini par <nom du fichier>.

5. Gestion des cartes.

a) Initialisation du lecteur de cartes.

nom : IC (Initialisation Card reader)
format : IC
définition :
Les cartes contenues dans le compartiment des cartes sont enlevées.

b) Chargement du lecteur de cartes.

nom : LC (Load Card)
format : LC <nom du fichier>

définition ;
Le contenu du fichier défini par ‹nom du fichier› est chargé dans le compartiment "cartes" et la première carte du paquet sera la carte courante.
erreur :
Un message d'erreur sera affiché si aucun fichier n'est défini par ‹nom du fichier›.

c) Avance d'une carte.

nom : NC (Next Card)
format : NC
définition ;
La carte qui suit la carte courante devient courante.
erreur :
Un message d'erreur sera affiché s'il n'y a plus de carte qui suit la carte courante.

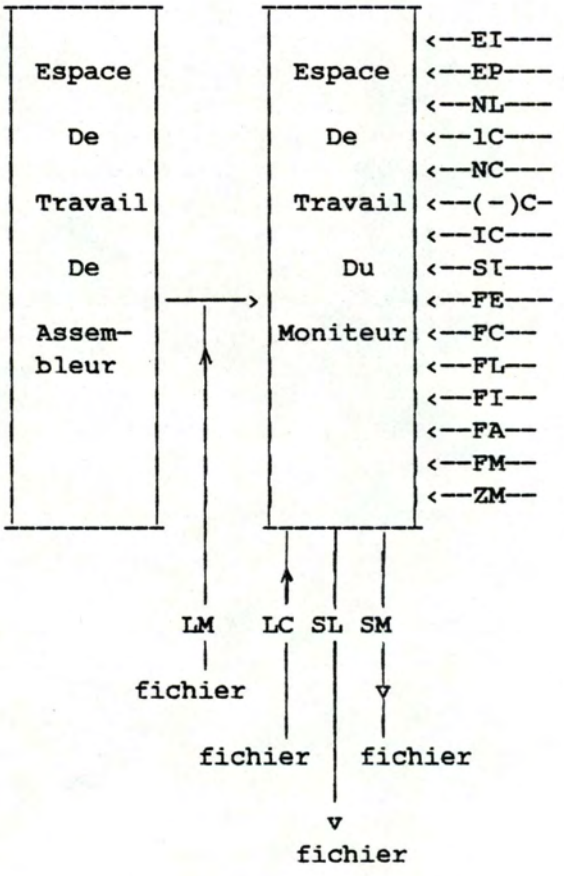
d) Recul d'une carte.

nom : -C
format : -C
définition :
La carte qui précède la carte courante devien courante.
erreur :
Un message d'erreur sera affiché s'il n'y a pas de carte qui précède la carte courante.

e)Positionnement de la tête de lecture sur la première carte.

nom : 1C
format : 1C
définition :
La première carte du paquet de carte devient la carte courante.

6. Représentation des opérations du moniteur.



2.2.3.3. Présentation des messages envoyés par le système.

1. message attente du système : ?

2. messages d'erreur :

   - ERR-MES-ENT : le message introduit par l'utilisateur est invalide;
   - ERR-OPER : l'opérateur de l'instruction est invalide;
   - ERR-ADR : l'adresse de l'opérande est invalide;
   - ERR-ARG : l'opérande de l'instruction est invalide;
   - ERR-CO : le contenu du compteur d'adresses est invalide;
   - ERR-ASS : la traduction d'un texte en langage machine n'a pu être réaliser.

2.2.3.4. Exemple d'utilisation du système :

1. Entrée dans l'éditeur (E);
2. Ecriture d'un programme (I,D,P,..);
3. Traduction du programme (T);
4. Si message reçu est ERR-ASS alors :
   - modification du programme (I,D,P,-, ,);
   - et aller en 3;
5. Appel du moniteur (M);
6. Initialisation de l'unité centrale (ZM,LM,FA,FE,FL,FM,..);
7. Exécution (EP);
8. Sauvetage du programme machine (SM);
9. Retour a l'éditeur (E);
10. sauvetage du programme écrit par l'utilisateur (S)
11. Sortie du système (Q).

2.3. L'application.

2.3.1. Introduction.

L'application est facilitée pour l'utilisateur grâce :
- à l'existence du langage de communication vue précedèment;
- à l'existence d'un langage assembleur.

C'est ce dernier qui va être developpé dans cette partie.

2.3.2. Le langage assembleur.

2.3.2.1. Introduction.

Le langage assembleur est un langage symbolique qui permet une grande facilité dans la rédaction des programmes pour l'utilisateur. Ce langage n'est pas directement accessible par notre ordinateur puisqu'il ne connait que le langage machine. C'est pourquoi il est nécessaire d'utiliser un programme d'assemblage qui réalise la transformation d'un programme en langage assembleur en un programme en langage machine.

Au niveau de la programmation, le langage symbolique présente de nombreux avantages sur le langage machine :
- l'utilisation d'un code opération mnémonique;
- les adresses des données et des instructions peuvent se présenter sous forme symbolique;
- l'introduction de constante et la réservation de zones pour les résultats est réalisée par des instructions spéciales;

2.3.2.2. Structure d'un programme assembleur.

Un programme assembleur se compose :
- d'instructions assembleur (obligatoire);
- de commentaires (facultatif).

La première instruction assembleur du programme détermine le nom du programme et l'adresse de chargement du programme lors de l'exécution.

La dernière instruction assembleur du programme détermine la fin du programme. Elle représente un délimiteur du texte assembleur.

2.3.2.2.1. Structure des instructions assembleurs.

Une instruction assembleur se subdivise en 3 parties :

- l'étiquette (facultative);
- le code opération;
- l'opérande (souvent obligatoire).

1. L'étiquette ou nom symbolique.

L'étiquette permet de donner un nom symbolique à une adresse auquel le programme pourra se référé ou à une donnée que le programme pourra utiliser. L'étiquette se compose d'une chaîne de caractères alphanumériques avec au moins un caractère alphabétique mais sans caractères spéciaux et sans espaces. Sa longueur sera limitée à 10 caractères. Le choix du nom se fera par le programmeur.

2. Le code opération.

Signalons d'abord qu'une instruction peut-être :

- une information pour le programme d'assemblage;
- une définition de constante ou de réservation de zone;
- une instruction à exécuter.

Le code opération indique au programme d'assemblage de quelle instruction il s'agit.

3. L'opérande.

Cette zone sert à décrire les données ou l'adresse de celles-ci à manipuler pour exécuter l'instruction. On peut utiliser indifféremment des adresses réelles ou des adresses symboliques.

2.3.2.2.2. Structure d'un commentaires.

Un commentaire est une suite de caractères alphanumériques précédé du signe ";". L'utilisation de commentaires permet d'expliciter ce que fait le programme.

exemple :

```
      |
      |      ;exemple de
      |      ; commentaires
      |
```

2.3.2.2.3. Règles d'écriture d'une ligne de programme en langage assembleur.

Une ligne peut se subdiviser en 4 zones :

1. l'étiquette : cette zone doit commencer en colonne 1 du programme;
2. le code opération : cette zone doit commencer après la colonne 1;
3. l'opérande : cette zone doit commencer après la zone précédente;
4. le commentaire : cette zone doit commencer par un ";".

Ces 4 zones doivent se suivre dans l'ordre et au moins un caractère blanc doit exister entre chacunes d'elles.

La forme d'une ligne sera donc

```
<zone 1><espaces>      (facultative)   |
      +                                |
<zone 2><espaces>                      |
      +                                |
                                       | facultatives
<zone 3><espaces>      (souvent        |
      +                  obligatoire)  |
  <zone 4><espaces>
```

2.3.2.3. Définition sémantique des instructions.

2.3.2.3.1. Instruction pour le programme d'assemblage.

1. Instructions relatives au compteur d'adresse.

(compteur d'adresse = compteur d'instruction = compteur ordinal)

a)code opération : PRG

fonction :

- l'étiquette de cette instruction détermine le nom du programme;

- l'opérande spécifie la valeur que doit avoir le compteur d'adresse devant cette instruction.

remarque :
cette instruction doit être la première du programme.

exemple :

```
FACT PRG 30 ; le programme porte le nom
            ; FACT et 30
            ;  détermine l'adresse de
            ; chargement du programme en
            ; mémoire centrale.
```

b)code opération : ORG

fonction :
l'opérande de cette instruction détermine la valeur du compteur d'adresse.

exemple :

CO	Instruction	Commentaire
30	FACT PRG 30	
31	ORG 35;	35---> CO
35		

2) Instruction de contrôle de programme.

code opération : END

fonction :
cette instruction indique au programme d'assemblage que c'est la fin du programme.

3) Instruction de définition de symbole.

code opération : EQU

fonction :
l'étiquette de l'instruction reçoit comme valeur le contenu de l'opérande.

exemple :

```
CO      Instructions     Commentaire
-----------------------------------
 30     UN       EQU 1   ; le symbole UN reçoit la
                         ;valeur 1
 31      ONE      EQU UN  ; le symbole  ONE  reçoit
                         ; la valeur 1
```

2.3.2.3.2. Instruction de définition de constantes et de réservation de zones.

1. Instruction de définition de zone.

code opératoire : DC

fonction :

l'opérande de cette instruction définit une constante ou valeur fixe qui est introduite en mémoire comme partie du programme.
L'étiquette (éventuelle) de cette instruction définit le nom de la zone contenant la constante. L'adresse de l'étiquette détermine l'adresse du premier mot de la zone réservée.
La valeur de l'opérande sera :

- une valeur numérique signée ou non. Cette valeur sera rangée dans le mot reservé.
- une valeur alphanumérique, c'est-à-dire une chaîne de caractères placée entre quote. Cette valeur sera rangée dans une zone de longueur définie à partir de la longueur de la chaîne de caractères. Dans notre cas, 2 caractères sont stockables dans un mot de la mémoire de l'ordinateur pédagogique.

exemple :

```
CO      Instructions     Commentaire
-----------------------------------
 30     THE      DC 'TEA' ; la zone THE réserve 2
                         ; mots contenant |TE|A |
 32              DC 'FOR'
 34     DEU      DC 2     ; la zone DEU réserve 1
                         ; mot contenant la valeur
                         ; 2
 35     ADEU     DC DEU   ; la zone ADEU contient
                         ; l'adresse de DEU c'est-
                         ; à-dire la valeur 34
```

2. Instruction de réservation de zone.

code opératoire : DS

fonction :

Cette instruction réserve une zone de longueur (exprimé en mots) déterminé par l'opérande et lui assigne le nom spécifié par l'étiquette de l'instruction.

remarque :

L'adresse de l'étiquette correspond à l'adresse du premier mot de la zone réservée.

exemple :

```
CO      Instruction     Commentaire
-------------------------------------------------
 30     TRAV    DS 10   ; réservation d'une zone
                        ; de 10 mots pour une zone
                        ; appelee TRAV
 31     DEU     EQU 2
 32     TRAV2   DS DEU  ; réservation d'une zone
                        ; de 2 mots pour une zone
                        ; appelee TRAV2
```

2.3.2.3.3. Instruction à exécuter.

1. Convention.

Voir convention du langage machine (page 15).

2. Modes d'adressages.

Les adresses disponibles sont ceux qui existent pour le langage machine (page 17).

3. Description des instructions.

a) Groupe des transferts.

Chargement de l'accumulateur.

code opératoire : LDA

fonction :
Cette instruction provoque le stockage dans l'accumulateur de l'argument spécifié dans l'opérande.

mode adressage	effet
nn	A <--- nn
(nn)	A <--- (nn)
((nn))	A <--- ((nn))
(E)	A <--- (E)
D((nn))	nn <--- (nn) - 1
	A <--- ((nn))
((nn))I	A <--- ((nn))
	nn <--- (nn) + 1

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
30      LDA 3           ;charger la valeur 3
                        ;dans l'accumulateur
31      LDA ((ADTAB))   ;charger le 1er élément
                        ;du tableau TAB
                        ;dans A
:
40 ADTAB DC 41          ;ADTAB contient la
                        ;valeur 41
41 TAB DS 10            ;TAB est un tableau
                        ;de 10 mots
```

Stockage du contenu de l'accumulateur.

code opératoire : STA

fonction :
Cette instruction provoque le stockage du contenu de l'accumulateur dans l'adresse spécifiée par l'opérande.

mode adressage	effet
(nn)	nn <--- (A)
((nn))	(nn) <--- (A)
(E)	E <--- (A)
D((nn))	nn <--- (nn) - 1
	(nn) <--- (A)
((nn))I	(nn) <--- (A)
	nn <--- (nn) + 1

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
29      LDA ADTAB       ;chargement de l'adresse
                        ;ADTAB dans l'accumulateur
30      STA (ADTAB)     ;transfert du contenu de A
                        ;dans la celule ADTAB
31      LDA (ELEM1)     ;chargement du contenu de
                        ;ELEM1 dans A
32      STA D((ADTAB))  ;décrémentation du contenu
                        ;de ADTAB et chargement du
                        ;contenu de A à la fin du
                        ;tableau TAB
:
40 TAB DS 10
50 ADTAB DS 1
51 ELEM1 DS 1
```

Echange du contenu de l'accumulateur.

code opératoire : EXC

fonction :
Cette instruction provoque l'échange du contenu de A avec l'argument.

mode adressage	effet
(nn)	échange de (A) avec (nn)
((nn))	échange de (A) avec ((nn))
(E)	échange de (A) avec (E)
D((nn))	nn <--- (nn) - 1
	échange de (A) avec ((nn))
((nn))I	échange de (A) avec ((nn))
	nn <--- (nn) + 1

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
 30     LDA (OPER1)     ;chargement dans A du
                        ;contenu de OPER1
 31     EXC (OPER2)     ;échange du contenu de
                        ;OPER2 avec A
 32     STA (OPER1)     ;chargement du contenu de
                        ;A dans OPER1
                        ;a ce stade
                        ;(OPER1) = 5
                        ;(OPER2) = 2
 40 OPER1 DC 2
 41 OPER2 DC 5
```

b) Groupe arithmétique.

Addition.

code opératoire : ADD

fonction :
Cette instruction provoque l'addition de l'argument au contenu de l'accumulateur.

Soustraction.

code opératoire : SUB

fonction :
Cette instruction provoque la soustraction de l'accumulateur du contenu de l'argument.

Multiplication.

code opératoire : MUL

fonction :
Cette instruction provoque la multiplication de l'argument au contenu de l'accumulateur.

Division.

code opératoire : DIV

fonction :
Cette instruction provoque la division du contenu de l'accumulateur par l'argument.

si on représente par OP un des symboles suivant {+,-,*,/}
on aura :

mode adressage	effet
nn	R <--- (A) OP nn
(nn)	R <--- (A) OP (nn)
((nn))	R <--- (A) OP ((nn))
D((nn))	nn <--- (nn) - 1
	R <--- (A) OP ((nn))
((nn))I	R <--- (A) OP ((nn))
	nn <--- (nn) + 1

si |R| < 100000 : A <--- R
si |R| > ou = 100000
et débordement permis alors :
R = (100000 * (E)) + (A)
et débordement non permis alors :
A <--- reste de (R/100000)

valeur de l'indicateur logique :

```
        si (A) = 0 alors IL <--- "="
        si (A) < 0 alors IL <--- "<"
        si (A) > 0 alors IL <--- ">"
        si dèbordement alors IL <--- "OV"
```

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
 30     LDA (OPER1)     ;(A) = 2
 31     MUL (OPER2)     ; (A) = -14
                        ; (IL) = "-"
 32     DIV (OPER3)     ; (A) = 3
                        ; (IL) = "+"
 33     ADD (MAXNB)     ; (A) = 2
                        ; (IL) = "OV"
                        ;si on considère que
                        ;les dèbordements ne
                        ;sont pas permis
 :
 40 OPER1 DC 2
 41 OPER2 DC -7
 42 OPER3 DC -5
 43 MAXNB DC 99999
```

Nègation du contenu de l'accumulateur.

code opèratoire : NEG

fonction :

Cette instruction provoque la multiplication du contenu de l'accumulateur par -1.

mode adressage	effet
indifférent	A <--- (A) * -1

valeur de l'indicateur logique : (voir b.1)

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
 30                     ;si on suppose l'état
                        ;suivant :
                        ;(A) = 3
                        ;(IL) = ">"
 31     NEG             ;(A) = -3
                        ;(IL) = "<"
```

Rendre absolu le contenu de l'accumulateur.

code opératoire : ABS

fonction :
Cette instruction rend positif le contenu de l'accumulateur.

valeur de l'indicateur logique : (voir b.1)

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
 32                     ;si on suppose que
                        ;(A) = -3
                        ;(IL) = "<"
 33     ABS             ;(A) = 3
                        ;(IL) = ">"
```

Décrémentation de (E) et branchement si (E) = 0.

code opératoire : DBZ

fonction :
Cette instruction provoque la décrémentation du contenu de E et le branchement à l'adresse spécifiée par l'opérande si (E) = 0.

```
mode adressage          effet
--------------          -----------------
 nn             E <--- (E) - 1
                       si (E) = 0 alors
                       CO <--- nn
 (nn)           E <--- (E) - 1
                       si (E) = 0 alors
                       CO <--- (nn)
 ((nn))         E <--- (E) - 1
                       si (E) = 0 alors
                       CO <--- ((nn))
 D((nn))        nn <--- (nn) - 1
                E <--- (E) - 1
                       si (E) = 0 alors
                       CO <--- ((nn))
 ((nn))I        E <--- (E) - 1
                       si (E) = 0 alors
                       CO <--- ((nn))
                       nn <--- (nn) + 1
```

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
 35     DBZ FIN         ;branchement à FIN
                        ;si (E) = 0
 :
 40 FIN HLT
```

Décrémentation de (E) et branchement si (E) > 0.

code opératoire : DBP

fonction :
Cette instruction provoque la décrémentation du contenu de E et le branchement à l'adresse spécifiée par l'opérande si (E) > 0.

mode adressage	effet
nn	E <--- (E) - 1 si (E) > 0 alors CO <--- nn
(nn)	E <--- (E) - 1 si (E) > 0 alors CO <--- (nn)
((nn))	E <--- (E) - 1 si (E) > 0 alors CO <--- ((nn))
D((nn))	nn <--- (nn) - 1 E <--- (E) - 1 si (E) > 0 alors CO <--- ((nn))
((nn))I	E <--- (E) - 1 si (E) > 0 alors CO <--- ((nn)) nn <--- (nn) + 1

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
 35     DBP FIN         ;branchement à FIN
                        ;si (E) > 0
 :
 40 FIN HLT
```

Décrémentation de (E) et branchement si (E) < 0.

code opératoire : DBN

fonction :
Cette instruction provoque la décrémentation du contenu de E et le branchement à l'adresse spécifiée par l'opérande si (E) < 0.

mode adressage		effet
nn	E <---	(E) - 1
		si (E) < 0 alors
		CO <--- nn
(nn)	E <---	(E) - 1
		si (E) < 0 alors
		CO <--- (nn)
((nn))	E <---	(E) - 1
		si (E) < 0 alors
		CO <--- ((nn))
D((nn))	nn <---	(nn) - 1
	E <---	(E) - 1
		si (E) < 0 alors
		CO <--- ((nn))
((nn))I	E <---	(E) - 1
		si (E) < 0 alors
		CO <--- ((nn))
		nn <--- (nn) + 1

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
35      DBN FIN         ;branchement à FIN
                        ;si (E) < 0
:
40 FIN HLT
```

c) Groupe logique

Comparaison.

code opératoire : CPA

fonction :
Cette instruction provoque la comparaison du contenu de l'accumulateur avec l'argument.

mode adressage	effet
nn	arg <--- nn
(nn)	arg <--- (nn)
((nn))	arg <--- ((nn))
D((nn))	nn <--- (nn) - 1
	arg <--- ((nn))
((nn))I	arg <--- ((nn))
	nn <--- (nn) + 1

	{=}		{+}
si (A)	{<} arg	alors	{<} ---> IL
	{>}		{>}

exemple :

CO	Instructions	Commentaires
30	LDA 3	;(A) = 3
31	CPA 5	;(IL) = "<"

d) Groupe des branchements

Branchement inconditionnel.

code opératoire : B

fonction :
Cette instruction provoque le branchement inconditionnel à l'adresse spécifiée par l'argument.

mode adressage	effet
nn	CO <--- nn
(nn)	CO <--- (nn)
((nn))	CO <--- ((nn))
D((nn))	nn <--- (nn) - 1
	CO <--- ((nn))
((nn))I	CO <--- ((nn))
	nn <--- (nn) + 1

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
 30     B FIN           ;(CO) <--- 40
 :
 40 FIN HLT
```

Branchement conditionnel simple.

code opératoire : B<, B>, B=, BOV, BER

fonction :
Ces instructions provoque le branchement conditionnel suivant la valeur contenue dans l'indicateur logique.

Soit Z qui représente une valeur de {<,>,=,OV,ER}
On aura BZ

mode adressage	effet
nn	si (IL) = Z alors
	CO <--- nn
(nn)	si (IL) = Z alors
	CO <--- (nn)
((nn))	si (IL) = Z alors
	CO <--- ((nn))
D((nn))	nn <--- (nn) - 1
	si (IL) = Z alors
	CO <--- ((nn))
((nn))I	si (IL) = Z alors
	CO <--- ((nn))
	nn <--- (nn) + 1

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
 30     LDA 5
 31     CPA 0           ;(IL) = ">"
 32     B< MIN          ;pas de branchement
 33     B> MAX          ;branchement à MAX
 :
 40 MIN
 :
 50 MAX
```

Branchement conditionnel composé.

code opération : B<>, B<=, B>=

Soit le code opération BZ où Z appartient à {<>,<=,>=} avec Z = XY, X appartenant à {<,>} et Y appartenant à {>,=}.

```
mode adressage     effet
--------------     ----------------
 nn                si (IL) = X ou Y
                           alors CO <--- nn
 (nn)              si (IL) = X ou Y
                           alors CO <--- (nn)
 ((nn))            si (IL) = X ou Y
                           alors CO <--- ((nn))
 D((nn))           nn <--- (nn) - 1
                   si (IL) = X ou Y
                           alors CO <--- ((nn))
 ((nn))I           si (IL) = X ou Y
                           alors CO <--- ((nn))
                   nn <--- (nn) + 1
```

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
 30     LDA (VAL1)
 31     CPA 0           ;(IL) = ">"
 32     B<> NONO        ;CO <--- 40
 :
 40 NONO
 :
 50 VAL1 DC 5
```

Appel de programme.

code opération : CAL

fonction :

Cette instruction provoque le sauvetage du contenu du compteur d'adresse dans l'accumulateur et le chargement dans le compteur d'adresse de la valeur spécifiée par l'opérande.

mode adressage	effet
nn	A <--- (CO) CO <--- nn
(nn)	A <--- (CO) CO <--- (nn)
((nn))	A <--- (CO) CO <--- ((nn))
(E)	A <--- (CO) A <--- (E)
D((nn))	nn <--- (nn) - 1 A <--- (CO) CO <--- ((nn))
((nn))I	A <--- (CO) CO <--- ((nn)) nn <--- (nn) + 1

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
30      CAL  SP1        ;A <--- 31
                        ;CO <--- 40
 :
40 SP1
```

e) Groupe des entrées (à partir du lecteur de cartes).

Lecture numérique.

code opératoire : RDN

fonction :

Cette instruction provoque :

- la lecture (|A|) nombres à partir de la tête de lecture, puis le rangement successif à partir de l'endroit désigné par l'opérande;
- après rangement, le chargement dans A du nombre exact de nombres lus.

remarque :

- s'il n'y a pas de carte suivante alors la valeur "OV" est rangée dans l'indicateur logique ("OV" ---> IL);
- si une zone n'est pas numérique, c'est-à-dire contient des caractères, alors la valeur "ER" est rangée dans l'indicateur logique ("ER" ---> IL);
- si le nombre de nombres restant à lire est inférieure aux nombres pas encore lus sur la carte, alors on lira les autres nombres sur la carte suivante.

Soit TMEM = la plus haute adresse disponible en mémoire centrale.

mode adressage	effet
(nn)	soit N = minimun(|(A)|,nombre de valeurs lus,TMEM - nn + 1) lecture de N nombres et rangement dans les cellules nn,nn + 1,..,nn + N - 1
((nn))	soit N = minimun(|(A)|,nombre de valeurs lus,TMEM - (nn) + 1) lecture de N nombres et rangement dans les cellules (nn),(nn) + 1,..,(nn) + N - 1
(E)	lecture du premier nombre et rangement dans E si |(A)| > 0
((nn))I	soit N = minimun(|(A)|,nombre de valeurs lus,TMEM - (nn) + 1) lecture de N nombres et rangement dans les cellules (nn),(nn) + 1,..,(nn) + N - 1; nn <--- (nn) + N;

format d'une carte :

Une carte ne contient que des chiffres, des signes "-" et des espaces. Tout autre caractère provoque la fin de la lecture (comme la fin de la carte) ainsi que la mise-à-jour de l'indicateur logique (IL <--- "ER") et A reçoit le nombre de nombres lus avant l'erreur.

Un nombre est représenté par une suite de chiffres consécutifs précédé éventuellement d'une suite d'espaces et/ou de signes "-". (le nombre de chiffres considérés correspond aux nombres de caractères stockables dans un mot de la mémoire centrale.

La fin de la représentation d'un nombre est detectée à la lecture :

- du nombre maximun de chiffres lus;
- d'un espace;
- d'un signe "-";
- de la fin de la carte.

Le début de la représentation d'un nombre est soit le premier caractère de la carte, soit le premier caractère qui suit la représentation d'un nombre.

Exemple :

Soit 5 le nombre de chiffres stockables dans un mot.
la carte :

```
|123456 -3-    14       5|
```

représente les nombres :

```
12345
6
-3
-14
5
```

Exemple de lecture de carte :

Soit la carte çi-dessus représentant la carte courante.

exemple :

```
CO      Instructions    Commentaires
-------------------------------------------
 30     LDA 5
 31     RDN (OPER1)     ;chargement des valeurs
                        ;lues sur la carte
                        ; à partir
                        ;de l'adresse OPER1
 :
 40 OPER1 DS 1          ;contient 12345 après RDN
 41 OPER2 DS 1          ;contient 6
 42 OPER3 DS 1          ;contient -3
 43 OPER4 DS 1          ;contient -14
 44 OPER5 DS 1          ;contient 5
```

Lecture alphanumérique.

code opératoire : RDC

fonctions :

Cette instruction provoque :

- la lecture de |(A)| caractères, puis le rangement à partir de l'endroit désigné par l'opérande;
- le rangement par groupe de 2 caractères dans chaque cellule ou registre. Les 2 caractères sont cadrés à droite dans la cellule ou registre.

remarque :

- la partie d'une cellule qui n'est pas garnie sera remplie de 0;
- s'il n'y a plus assez de caractères sur une carte, la lecture continuera sur la carte suivante (si elle existe);
- s'il n'y a plus de carte alors IL reçoit la valeur "OV".

mode adressage	effet
(nn)	soit N = min (|(A)|,nombre de caractères lus,TMEM -nn + 1) lecture des N caractères et rangement dans les cellules nn,nn + 1, .., nn + M - 1; $M = \text{entier}\left(\frac{(N+1)}{2}\right)$
((nn))	soit N = min (|(A)|,nombre de caractères lus,TMEM - (nn) + 1) lecture des N caractères et rangement dans les cellules (nn),(nn) + 1,..,(nn) M - 1;

$$M = \text{entier}\left(\frac{(N+1)}{2}\right)$$

(E) lecture de 2 carctères et rangement dans E si |(A)| > 0

((nn))I idem que ((nn))
ensuite faire nn <--- (nn) + N

exemple :

Soit la carte courante suivante :

```
| ABCDEF          |
```

La tête de lecture est positionnée devant "A".

```
CO      Instructions    Commentaires
-------------------------------------------
 30     LDA 5
 31     RDC   (ZONLEC)  ;la   zone   ZONLEC
                        ;reçoit
                        ;5 caractères
                        ;"AB"--> 1er mot de
                        ; ZONLEC
                        ;"CD"--> 2eme mot de
                        ; ZONLEC
                        ;"E"---> 3eme mot de
                        ; ZONLEC
 :
 40 ZONLEC DS 5
```

f) Groupe des sorties.

Ecriture numérique.

code opératoire : PRN

fonction :
Cette instruction provoque l'impression de |(A)| nombres définit par l'argument de l'opérande, chacun dans une zone définie sur une longueur de 6 chiffres, cadrée à droite, à partir de la position courante de la tête d'écriture. Le passage à la ligne suivante est réalisé lorsqu'il reste moins que 6 positions à droite , à partir de la position courante de la tête d'écriture.

remarque :
Après l'impression, le nombre exact de nombres imprimés est rangé dans A.

mode adressage	effet
nn	imprime nn si |(A)| > 0
(nn)	soit N = min (|(A)|,TMEM - nn + 1) imprime (nn),(nn + 1),.. (nn + N - 1)
(E)	imprime (E) si |(A)| > 0
((nn))	soit N = min (|(A)|,TMEM - (nn) + 1) imprime ((nn)),((nn) + 1),.. ((nn) + N - 1)
((nn))I	idem que ((nn)) ensuite nn <--- (nn) + 1

Ecriture alphanumérique.

code opératoire : PRC

fonction :
Cette instruction provoque l'envoit à l'imprimante d'une chaîne de caractères de longueur |(A)|;
- s'il s'agit d'un caractère de commande, celle-ci est exécutée;
- s'il s'agit d'un caractère imprimable, celle-ci est imprimée;
- tout autre caractère est sans effet.

remarque :
- L'impression se fait à la position courante de la tête d'écriture; celle-ci est ensuite deplacée d'une position vers la droite ou au début de la ligne suivante si la tête vient d'imprimer à la dernière position de la

ligne;

- Le contenu d'un registre ou d'une cellule est interprètè comme ètant constituè de 2 caractères qui occupent les positions de droite de la cellule ou du registre;
- Après impression, le nombre exact de caractères envoyès est rangè dans A.

mode adressage	effet
nn	imprimer nn si |(A)| > 0
(nn)	soit N = minimun(|(A)|,(TMEM - (nn) + 1)*2) représentant la longueur de la chaîne à écrire exprimée en caractères; soit M = (N + 1)/2 représentant le nombre de cellules concernées; imprimer les N caractères suivant contenus dans nn,(nn + 1),.(nn + N - 1)
(E)	si |(A)| = 1 alors envoit du 1er caractère de (E) si |(A)| > 1 alors envoit du 1er et 2eme caractère de (E)
((nn))	soit N = minimun(|(A)|,(TMEM - (nn)+ 1)*2) représentant la longueur de la chaîne à écrire exprimée en caractères; soit M = (N + 1)/2 représentant le nombre de celules concernées; imprimer les N caractères suivant contenus dans ((nn)),((nn) + 1),..((nn) + N - 1).
((nn))I	idem que ((nn)) ensuite nn <--- (nn) + 1

Il est à remarquè que ces deux dernières instructions sont sans effet si |(A)| = 0.

exemple :

Soit la tête d'écriture représentée par []. Soit une ligne de 20 caractères

```
CO      Instructions    Commentaires
-------------------------------------------
 30     PRC 6           ;positionnement de la
                        ;tête d'écriture à la
                        ;position 6
 31     LDA 12          ;
 32     PRC (NOM)
 33     PRC 27          ;écriture de 2 espaces
 34     PRN 2           ;écriture du chiffre 2
 :
 40 NOM DC "NOM : "
 41     DC "DUPONT"
```

Etat de l'imprimante en fonction des instructions ci-dessus.

```
 29 []____________________
 30 _____[]_______________
 31 _____[]_______________
 32 _____NOM : DUPONT[]__
 33 _____NOM : DUPONT  []
 34 ____NOM : DUPONT    (saut à la ligne
                        suivante )
        2[]______________
```

g) Groupe divers

Sans effet.

code opération : NOP

fonction : sans effet

Arrêt de la machine.

code opération : HLT

fonction : arrêt de la machine

Overflow du registre E non permis.

code opération : DOV

fonction :
Cette instruction empêche l'overflow du registre E.
effet :
Inhibe la mise-à-jour du registre E lors des débordements survenant lors d'une opération arithmétique. La mise-à-jour de l'indicateur logique n'est pas inhibé.

Overflow de registre E permis.

code opération : EOV

fonction :
Cette instruction permet l'overflow du registre E.
effet :
Autorise la mise-à-jour du registre E lors des débordements survenant lors d'une opération arithmétique. Ce mode est celui de la machine lors de son allumage.

2.3.3. Table de correspondance entre les instructions machines et les instructions assembleurs.

nn = opérande de l'instruction;
.. = opérande sans signification;
zone blanche = cas impossible.

Mode D'adressage	nn	(nn)	((nn))	(E)	D((nn))	((nn))I
Nom du code opératoire assembleur						
LDA	100nn	101nn	102nn	103..	104nn	105nn
STA		111nn	112nn	113..	114nn	115nn
EXC		121nn	122nn	123..	124nn	125nn
ADD	200nn	201nn	202nn	203..	204nn	205nn
SU.	210nn	211nn	212nn	213..	214nn	215nn
MU'	220nn	221nn	222nn	223..	224nn	225nn
DI'	230nn	231nn	232nn	233..	234nn	235nn
NEG	240..					
ABS	250..					
DBZ	260nn	261nn	262nn	263..	264nn	265nn
DBP	270nn	271nn	272nn	273..	274nn	275nn
DBN	280nn	281nn	282nn	283..	284nn	285nn
CPA	300nn	301nn	302nn	303..	304nn	305nn
B	400nn	401nn	402nn	403..	404nn	405nn
B=	410nn	411nn	412nn	413..	414nn	415nn
B<	420nn	421nn	422nn	423..	424nn	425nn
B>	430nn	431nn	432nn	433..	434nn	435nn
B<>	440nn	441nn	442nn	443..	444nn	445nn
B<=	450nn	451nn	452nn	453..	454nn	455nn
B>=	460nn	461nn	462nn	463..	464nn	465nn
BOV	470nn	471nn	472nn	473..	474nn	475nn
BER	480nn	481nn	482nn	483..	484nn	485nn
CAL	490nn	491nn	492nn	493..	494nn	495nn
RDN		701nn	702nn	703..		705nn
RDC		711nn	712nn	713..		715nn
PRN	800nn	801nn	802nn	803..		805nn
PRC	810nn	811nn	812nn	813..		815nn
NOP	900..					
HLT	910..					
DOV	920..					
EOV	930..					

3. DESCRIPTION DE LA REALISATION.

3.1. Introduction.

Après avoir vu les différentes fonctions de l'ordinateur pédagogique proposé, nous allons voir comment est réalisé le système qui gère tout ceci.

Avant de décrire l'architecture du système, nous présenterons les critères retenus pour la décomposition de l'architecture.

3.2. Critères de décomposition.

Les critères de structuration sont nombreux, ceux qui ont été retenus sont les suivants :

- fiabilité : le produit final doit être cohérent avec les spécifications de l'ordinateur pédagogique proposé;
- portabilité : la structure générale doit être indépendante d'une configuration "hardsoft";
- performance/efficacité : le temps de réponse doit être bref et l'espace mémoire ne doit pas être gaspillé;
- indépendance entre décision du "design" : les traitements doivent être indépendants de ma structure.

3.3. Description de l'architecture.

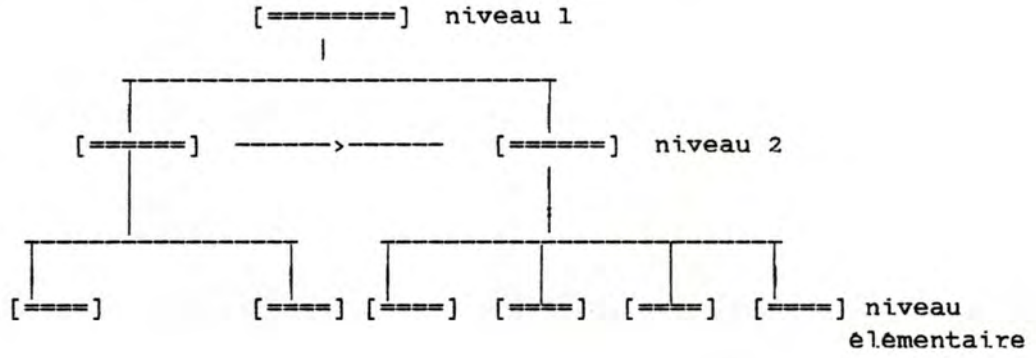
3.3.1. Introduction.

L'architecture a été élaborée suivant une démarche "TOP DOWN", elle permet de ramener un problème complexe en un ensemble de sous-problèmes facile à résoudre. Etablir l'architecture d'un système revient donc à définir un ensemble structuré de composantes.

3.3.2. Description des concepts utilisés.

modularisation : la structure d'un système est modulaire si elle est developpée suivant une série de niveaux distincts et ordonnés, chaque composante d'un niveau peut être en relation avec des composantes du même niveau ou du noveau inférieur.

le shèma suivant montre un exemple de découpe modulaire :



Une composante "UTILISE" une autre composante : une composante A UTILISE une composante B si le fonctionnement correct de A dépend de la disponibilité d'une version correcte de B. La composante A qui utilise B peut ignorer totalement le fonctionnement interne de B.

Découpe fonctionnelle : une structure qui possède une découpe fonctionnelle si chacune de ces composantes représente une fonction de l'analyse fonctionnelle. Une telle découpe permet de définir les fonctions qui sont indépendantes et ainsi on peut créer des composantes indépendantes.

Découpe par niveau d'abstraction : une structure qui possède une découpe par niveau d'abstraction traite uniquement les aspects essentiels à un niveau et laisse le détail de l'implémentation à des niveaux ultérieurs.

3.3.3. Présentation de l'architecture.

3.3.3.1. Introduction.

L'architecture a une structure modulaire. La découpe a été d'abord fonctionelle ensuite chaque module fonctionnelle a été divisé suivant une découpe par niveau d'abstraction.

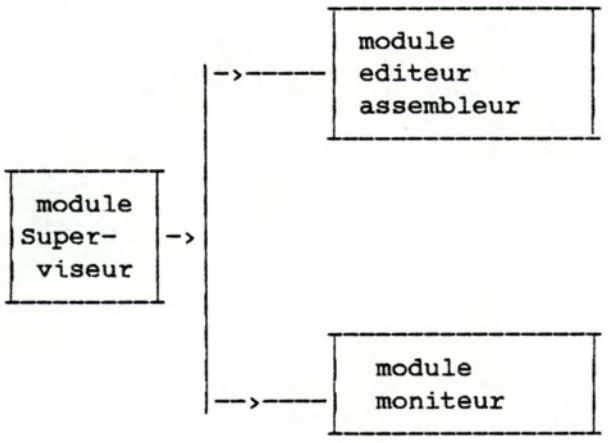
3.3.3.2. L'architecture.

Cette description comprend d'abord une représentation graphique de l'architecture , vient ensuite la description des modules.

A.1) Découpe fonctionnelle

Une découpe fonctionnelle reflète parfaitement les fonctions de l'ordinateur pédagogique. En effet, chacune d'elle pourra être considerée indépendament entraînant ainsi une augmentation de la fiabilité.

niveau 1



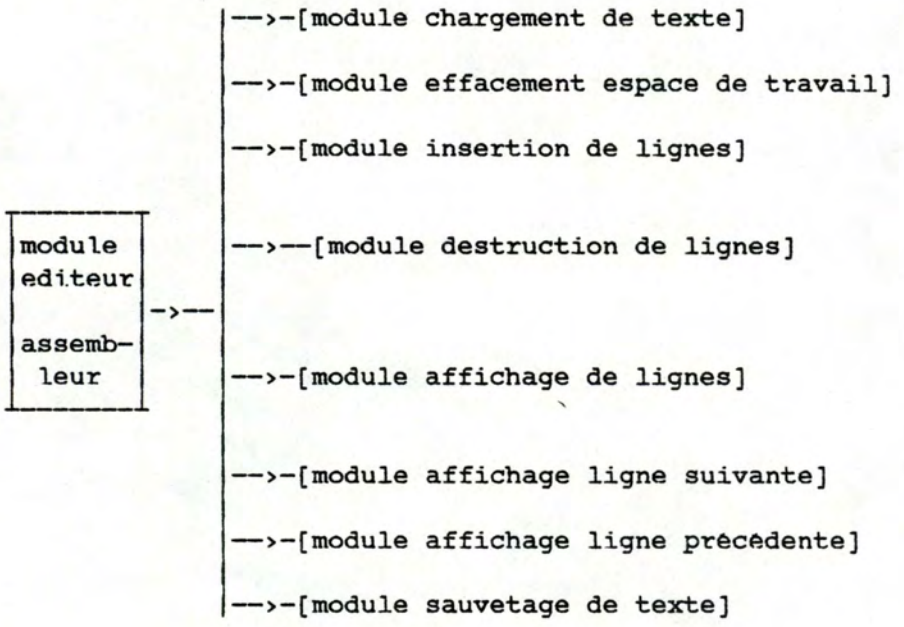
Module superviseur :

ce module lit une commande superviseur et ensuite appelle :

- le module éditeur-assembleur;
- le module moniteur;

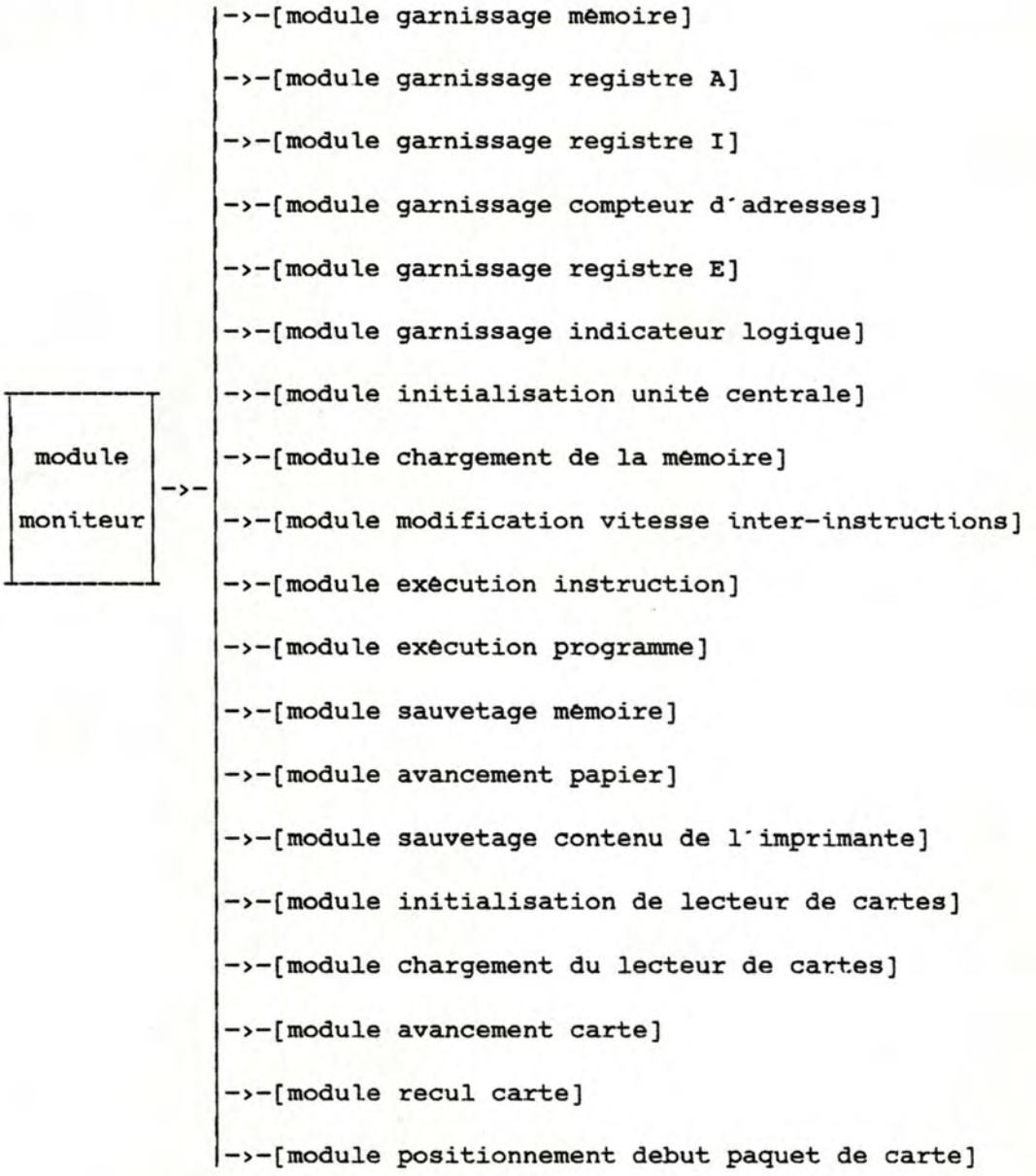
ou provoque la sortie du système.

niveau 2.1



Module éditeur/assembleur :
ce module lit une commande
- de type "superviseur" entraînant un retour au module superviseur;
- de type "éditeur/assembleur" entraînant l'appel au module qui lui est associée.

niveau 2.2



Module moniteur:

Ce module lit une commande :

- de type "superviseur" auquel cas il rend la main au superviseur;
- de type "moniteur" auquel cas il appelle le module qui lui est associé.

A.2) Découpe par niveau d'abstraction.

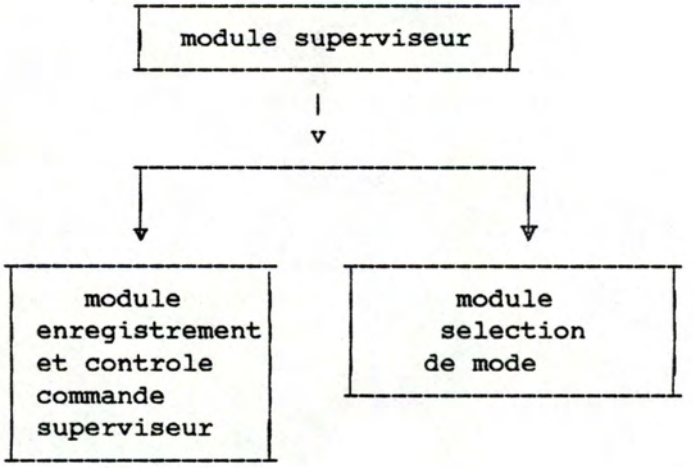
Chaque module "fonctionnel" est à nouveau décomposé en plusieurs niveaux. Cette seconde découpe permet :

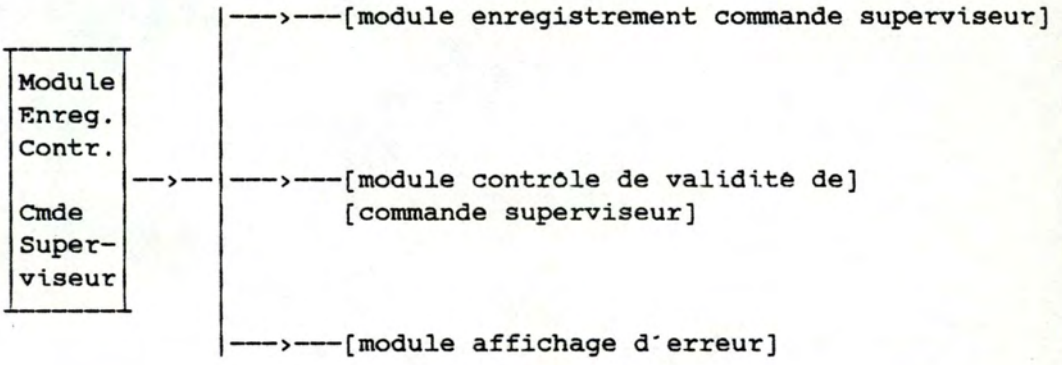
- de diminuer la dimension du module. En effet si la dimension est trop grande alors son but échappe. Plus un module est petit et plus il est facile à le concevoir et à le valider. Mais une découpe exessive a pour conséquence d'augmenter le nombre d'interfaces. Il faut donc faire un compromis entre la taille et le nombre d'interfaces d'un module;
- d'obtenir des modules utilisables par d'autres modules fonctionnels. Ces modules doivent donc être développé de telle manière que leur fonctionnement correct est indépendant du module qui l'appelle;
- d'isoler à un bas niveaux, les modules dépendant de la configuration du matétiel utilisé. Chacun de ces modules est donc localisable et facilement modifiable;
- d'introduire facilement de nouveaux modules, l'architecture dispose de plusieurs modules "d'aiguillages" permettant l'intégration aisée de nouvelles procédures.

Etant donné le nombre parfois important de modules utilisés par un module fonctionnel nous n'envisagerons ici que la découpe partielle du module superviseur. Il est a remarqué que la méthode de découpe est semblable pour les autres modules fonctionnels.

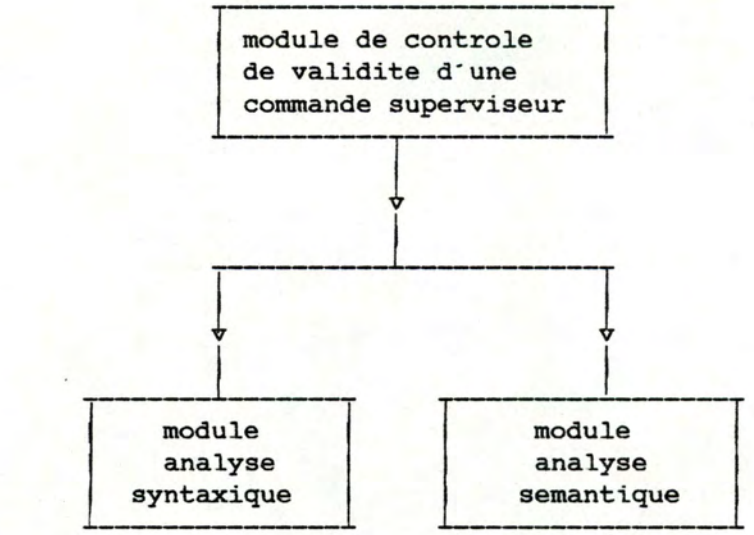
Actuellement, 24 modules sont issus de la découpe des modules fonctionnels, chacun d'eux utilise de 1 à 9 modules du niveau "d'abstraction".

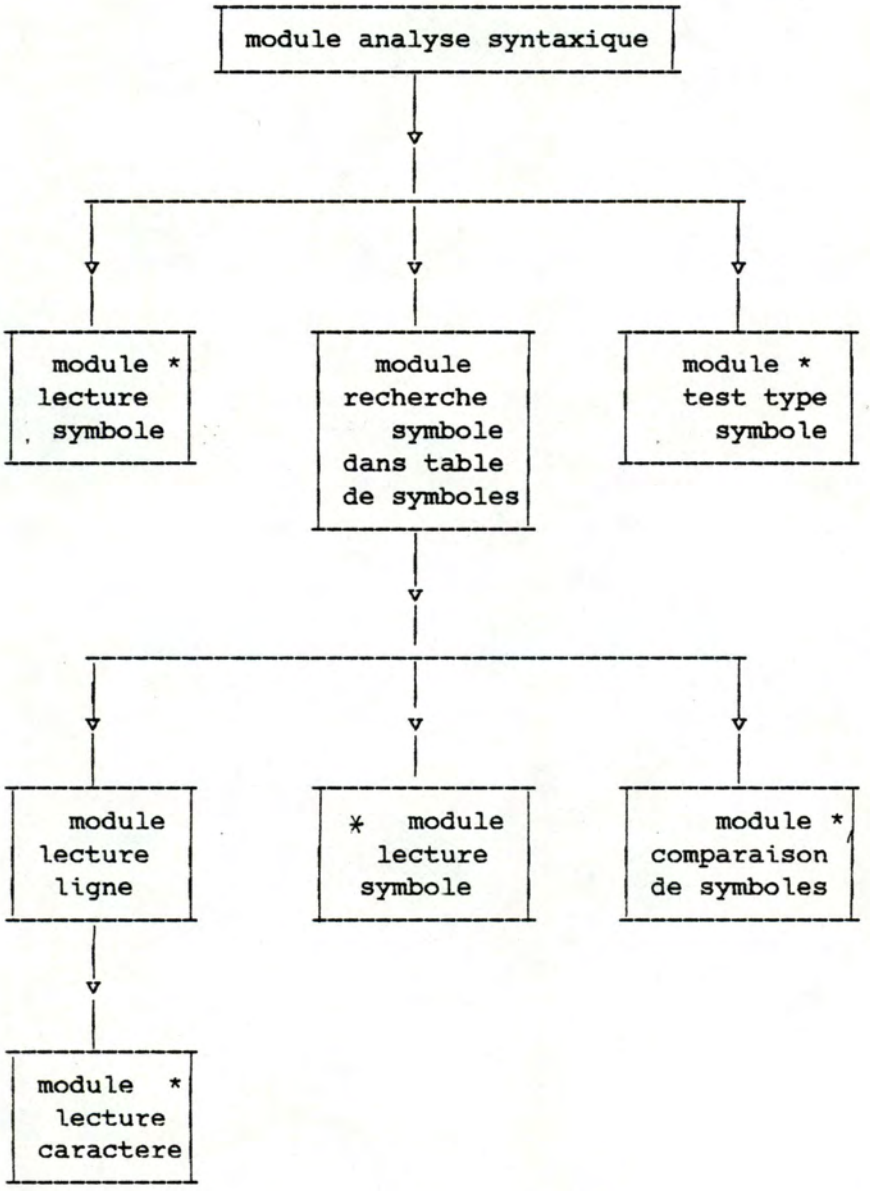
Le symbole * signifie que le module appartient au niveau élémentaire.


module superviseur
module enregistrement et controle commande superviseur
module selection de mode


Module Enreg. Contr. Cmde Super- viseur
[module enregistrement commande superviseur]
[module contrôle de validité de] [commande superviseur]
[module affichage d'erreur]


module enregistrem. commande superviseur
module lecture ligne
module * lecture caractère


module de controle
de validite d'une
commande superviseur
module
analyse
syntaxique
module
analyse
semantique


module analyse syntaxique
module *
lecture
symbole
module
recherche
symbole
dans table
de symboles
module *
test type
symbole
module
lecture
ligne
* module
lecture
symbole
module *
comparaison
de symboles
module *
lecture
caractere

B) Description des modules.

Chaque module sera décrit de la manière suivante :

- une définition, qui présente brièvement la fonction du module;
- un format, qui définit la syntaxe de l'appel de la procédure;
- les arguments, qui présente les types d'éléments utilisés par le module;
- le résultat, qui définit les informations de sortie du module;
- la pré-condition, qui explicite la condition d'entrée du module;
- la fonction, qui décrit ce que réalise le module;
- la post-condition, qui explicite la condition de sortie du module.

Conventions :

- un module appellé est représentépar la mise entre parenthèses du nom du module, exemple (LECTSYMB);
- une variable qui est utilisée co. ne élément d'entrée et de sortie sera représenté par le nom de la variable suivi du symbole " pour la valeur à l'entr'e du module et le nom de la variable seul pour la valeur de sortie du module.

Exemple de description :

Ce module a été choisi car c'est l'un des plus utilisés.

LECTSYMB

définition : lecture d'un symbole;

format :

```
LECTSYMB( zone_de_lecture,
          adr_debut_lecture,
          symbole_lu, long_symb_lu );
```

argument :

- zone_de_lecture : chaîne de caractères;
- adr_debut_lecture : nombre entier représentant une position dans zone_de_lecture;
- une sentinelle située dans la chaîne précisant la fin de la chaîne;
- symbole_lu : chaîne de caractères;
- long_symb_lu : nombre entier représentant la longueur du symbole lu;

résultat :

(symbole lu = 1er symbole qui suit la position de adr_debut_lecture)
ET
(long_symb_lu = longueur du symbole lu)
ET
(adr_debut_lecture = position qui suit le symbole lu dans
la zone zone_de_lecture);

prê_lectsymb :

(1 <= adr_debut_lecture <= position de la sentinelle <=
longueur maximun de la chaîne de caractères);

fonction :

- rechercher à partir de adr_debut_lecture un caractère significatif (differrent du blanc);
- extraire et compter chaque caractère significatif jusqu'à la rencontre d'un caractère blanc ou de la sentinelle;

post_lectsymb :

((long_symb_lu > 0)
ET (symbole lu appartient à zone_de_lecture dans
l'intervalle [adr_debut_lecture"..adr_deb_lecture])
ET (adr_debut_lecture appartient à l'intervalle
[adr_debut_lecture"..longueur maximun de
zone_de_lecture])
OU
((long_symb_lu = 0)
ET zone_de_lecture[adr_debut_lecture] = sentinelle)

Description des autres modules.

Ceux-ci seront décrites en annexes.

4. IMPLEMENTATION DE L'ORDINATEUR PEDAGOGIQUE.

4.1. Introduction :

Cette partie comprend les outils utilisés pour réaliser l'ordinateur pédagogique. Elle comprend :

- le choix du langage de programmation;
- le matériel utilisé pour l'implémentation actuel.

4.2. Choix du langage de programmation.

4.2.1. Introduction.

C'est le langage PASCAL qui a été choisi pour implémenter l'ordinateur pédagogique. Ce langage répond aux critères suivants :

- facilité d'apprentissage du langage;
- portabilité : la pluspart des ordinateurs (micro, mini et gros ordinateurs) disposent de ce langage;
- facilité de structuration de programme .

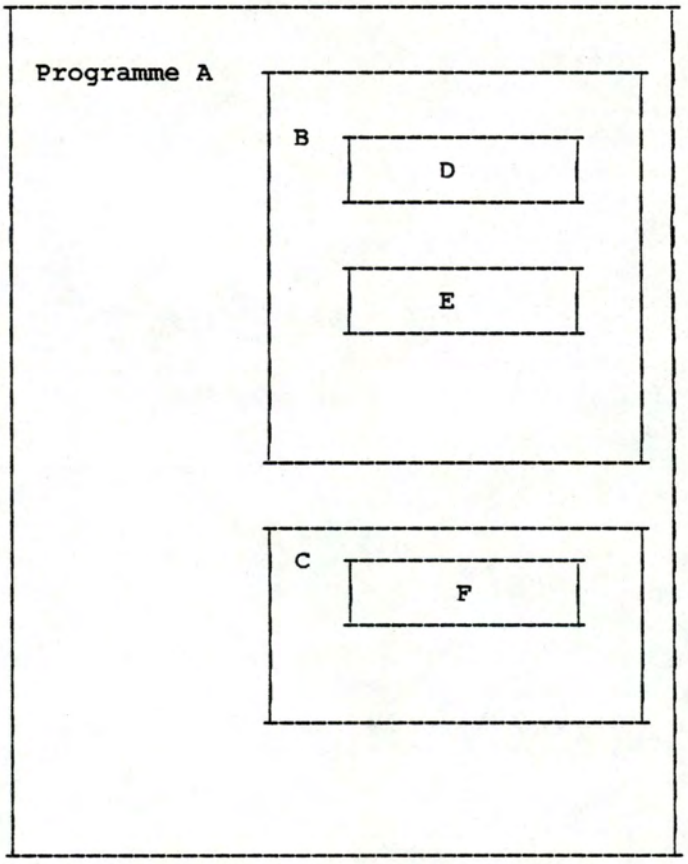
4.2.2. Caractéristiques du langage.

Le corps d'un programme PASCAL est constitué de sections qui peuvent être groupées en 2 :

- les sections de déclaration (étiquettes, constantes, types, variables, procedures ou fonctions);
- la section d'instruction exécutables.

Chaque procédures ou fonctions déclarées peut aussi contenir des blocs possédant toute les sections définies ci-dessus.

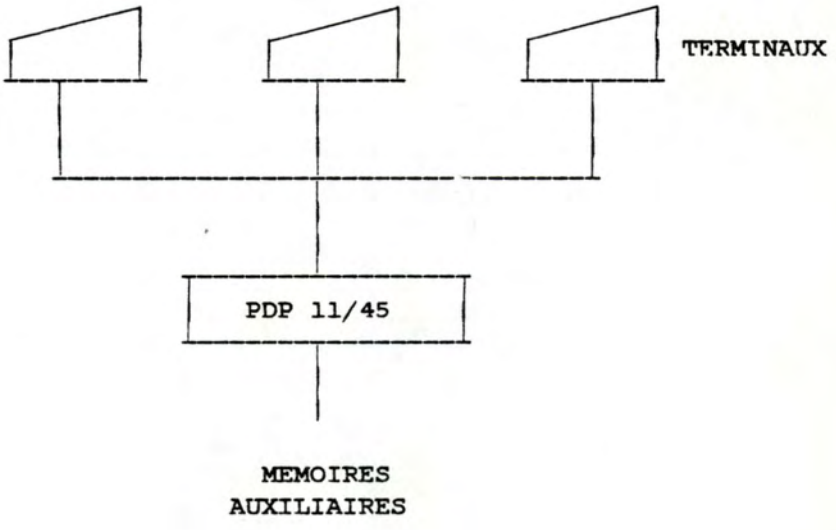
Cette structure peut se représenter par un schéma du type :



On peut facilement voir que PASCAL est un langage structuré. Dans ce schéma, le programme A utilise B et C tandis que B utilise D et E, C utilise F.

3.3. Présentation du matériel utilisé pour l'implémentation actuelle.

Actuellement, l'ordinateur pédagogique est partiellement implémenté sur un mini-ordinateur de Digital Equipement Corporation (PDP 11/45). Il possède une mémoire centrale de 256 Kbytes, un mot mémoire occupe 16 bits. Cet ordinateur dispose d'un système d'exploitation UNIX. Ce dernier propose un ensemble important de commandes, souvent inexistantes sur d'autres systèmes d'exploitation, qui permettent de faciliter grandement le travail de l'utilisateur.



5. MAINTENANCE DE L'ORDINATEUR PEDAGOGIQUE.

5.1. Introduction.

L'insatisfaction des utilisateurs, la découverte d'erreurs non détectées et le changement de matériel entraîne la maintenance de l'ordinateur pédagogique.

Cette partie décrira les efforts fournis pour faciliter la maintenance. Il est à remarquer que le problème de maintenance est très important en informatique.

5.2. Insatisfaction des utilisateurs.

5.2.1. Introduction.

Un utilisateur peut être insatisfait à cause de :

- l'absence d'une commande utilisateur;
- la dénomination d'une commande utilisateur;
- l'imprécision d'un message du système;
- l'absence d'une instruction machine;
- la configuration de l'ordinateur pédagogique proposé.

5.2.2. Absence d'une commande utilisateur.

L'ajoût d'une nouvelle commande se fait suivant les étapes suivantes :

- insertion du nom de la commande dans le fichier des commandes utilisateurs;
- modification des valeurs des délimiteurs des commandes dans le fichier des délimiteurs;
- insertion d'un nouveau type de commande dans le fichier des types de commandes;
- création du module fonctionnel associé à la commande;
- déclaration de ce module dans le module d'aiguillage concerné;
- compilation des procédures de niveau 1 à 3 et des procédures d'analyse syntaxique et sémantique des commandes.

Remarque :

Actuellement, 10 commandes supplémentaires ont été introduites (5 pour le moniteur et 5 pour l'éditeur), ces commandes permettent l'ajoût facile de nouvelle commande. En effet, une nouvelle commande est introduite par :

- la modification du nom d'une de cette commande dans le fichier des commandes;
- le developpement d'un nouveau module fonctionnel associé à la commande;

- la déclaration de ce module dans le module d'aiguillage;
- la compilation unique du module appelant et du nouveau module (cela permet une grande économie du nombre de compilations).

5.2.3. Modification de la dénomination d'une commande.

Ceci est réalisée simplement par modification du nom de la commande dans le fichier des commandes.

5.2.4. Modification d'un message du système.

Tout les messages du système sont stockè dans un fichier de messages. La modification d'un nom revient donc à modifier ce nom dans ce fichier de messages.

5.2.5. Absence d'une instruction machine.

Les instructions du langage machine sont stockées dans un fichier décrit de la manière suivante :
- le code opératoire de l'instruction machine;
- les adressages possibles pour cette instruction machine;
- le code opératoire mnémonique.

L'ajoût d'une nouvelle instruction se réalise en réalisant les étapes suivantes :
- mise-à-jour du fichier des instructions machines;
- mise-à-jour du module d'exécution d'instruction machine c'est-à-dire :
  1. si la nouvelle instruction appartient à un nouveau groupe d'instructions alors
     - mise-à-jour du module d'aiguillage d'exécution d'instruction machine;
     - développement du module d'exécution de la nouvelle instruction.
  2. si la nouvelle instruction n'appartient pas à un nouveau groupe alors insertion dans le groupe concerné du module d'exécution de la nouvelle instruction.

5.2.6. Modification de la configuration de l'ordinateur proposé.

Cette modification peut se situer à plusieurs niveaux :
- changement dans le format ou la localisation d'une composante de l'ordinateur;

- ajoût d'une nouvelle composante.

La configuration de l'ordinateur pédagogique est stockée dans un fichier. Le contenu de ce fichier provient de l'exécution d'un programme spécial, celui-ci calcule et contrôle , à patir de données associées à chaque composante , la position et le format de la composante dans la représentation de l'ordinateur. Ce programme génère deux éléments :

- le graphique de l'ordinateur pédagogique, qui est calculé à partir de données de chaque composante, c'est-à-dire une position relative de la composante vis-à-vis de la composante de niveau supérieur et vis-à-vis de la composante immédiatement à gauche;
- les nouvelles données de chaque composante en valeurs absolues.

Un changement dans le format ou la localisation est réalisé :

- en modifiant le fichier contenant les données initiales;
- en lançant l'exécution du programme spécial;
- en compilant tout les autres modules.

Un ajoût d'une composante se réalise :

- en modifiant le fichier contenant les données initiales;
- en modifiant le programme spécial qui doit calculer et contrôler les valeurs pour la nouvelle composante;
- en lançant l'exécution du programme spécial (après compilation de celui-ci bien sûr);
- en compilant tout les autres modules.

5.3. Découverte d'erreurs non détectées.

Il faut alors localiser l'erreur et la corriger. Chaque module contient une description de ce qu'il reçoit en entrée et en sortie , de ce que réalise le module. Cela permet de comprendre plus rapidement la fonction du module et donc la correction est alors plus rapide.

5.4. Changement de matériel.

Tout les modules qui sont dépendant du software ou du hardware sur laquelle l'ordinateur a été implémenté sont facilement localisable. Le nombre de ces modules est relativement réduit étant donné le critère de portabilité qui a été retenu pour la réalisation.

On peut citer les modules suivants :

1. dépendance avec la manipulation de fichier :
   - lecture d'un fichier externe;
   - création et écriture d'un fichier externe.
2. dépendance avec le type de terminal utilisé :
   - positionnement du curseur;
   - modification du mode du terminal (graphique ou caractère);
   - effacement du contenu de l'écran.

6. PROBLEMES RENCONTRES.

Des problèmes sont apparus surtout lors de l'implémentation. Les raisons sont les suivantes :

- l'impossibilité au moment de l'élaboration de l'architecture d'établir une architecture définitive;
- certaines faiblesses du langage PASCAL ainsi :

  1. les fichiers utilisés par ce langage sont de type séquentiel;
  2. l'utilisation par un sous-programme d'un fichier ne peut se faire que si le fichier lui est envoyé comme paramètre;
  3. l'ordre des passages de paramètres suivant leur type a une importance capitale. (un mois de recherche sur la cause d'une erreur m'a permis de vérifier la vérité de cette ordre de passage.

CONCLUSION.

Le but de ce mémoire consistait à réaliser l'étude, la conception et la réalisation d'un ordinateur pédagogique.

Malheureusement, le peu de temps alloué pour accomplir ce travail a empêché son élaboration complète. Pour cette raison, seul le noyau de cet ordinateur pédagogique (simulateur graphique, langage machine et langage de manipulation de l'ordinateur) a pu être complètement réalisé.

Néanmoins, d'autres composantes, tel l'éditeur et le programme d'assemblage, peuvent s'intégrer dans l'architecture, ce dernier dispose d'une découpe modulaire permettant l'insertion aisée de nouveaux modules.

Malgré cela, ce mémoire m'a donné l'occasion de parcourir les différentes étapes d'un projet informatique. De plus, cela m'a aussi permis d'appliquer les principes méthodoloques qui m'avaient été enseignés. Ces principes m'ont servi de guide tout au long de ce mémoire.

BIBLIOGRAPHIE.

- M. Campbell- Kelly,
  An introduction to macros (1973)
  (Collection U);

- John A.N.Lee,
  The anatomy of a compiler (1967)
  (Reinhold Book Corporation);

- Robert J. Wimmert,
  Computer Programming Techniques
  (Volume 1 Machine/Assembly languages)

- AFCET,
  Système C.I.C (compilateur,interpréteur universel);

- AFCET,
  Langages et communication graphique;

- AFCET, Colloque Université de Grenoble (1968)
  Les systèmes conversationnels (système SPARTACUS)
  (DUNOD);

- Kernighan Plauger,
  Software Tools (1976);

- Knuth,
  The art of computer programming
  Fundamental Algorithms (MIX chapitre 1);

- M.A. Jackson,
  Principles of program design (1975);

- Computer education : time for change,
  Input two-nine (1977);

- Walker,Gurd,Drawneek,
  Interactive Computer Graphics,
  Computer systems engineering series (1975);

- C.T. Meadow,
  Man-machine communication,
  Information sciences series (1970);

- Conversational computers,
  John Wiley & sons, Inc. (1968)

- J.-L Hainaut,
  Manuel utilisateur de l'ordinateur pédagogique
  implémenté sur le TRS 80;

- P. Lignelet,
  Pascal (tome 1 & 2),
  Masson (1980);

- K. Jensen N. Wirth,
  Pascal, manuel de l'utilisateur,
  Eyrolles (1980);

- J. Welsh & J. Elder,
  Introduction to Pascal,
  P.H.I. (1979).

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

ETUDE, CONCEPTION ET REALISATION

D'UN ORDINATEUR PEDAGOGIQUE.

(ANNEXE 1)

Mémoire présenté par

Johnny Vendrix

en vue de l'obtention
du titre de
Licencié et Maître en Informatique.

Année académique 1980-1981

DESCRIPTION DES MODULES

TABLE DES MATIERES

NIVEAU 0

TITRE : Superviseur

DEFINITION : aiguillage vers le module moniteur ou éditeur assembleur

FORMAT : AP_SUP(fichiers du système)

ARGUMENT : - 4 fichiers de types séquentiels

RESULTAT : appel du module moniteur ou de l'éditeur assembleur suivant le type de commande "superviseur" lu ou reçu

PRE_AP_SUP : _

FONCTION :

a) si l'utilisateur vient d'entrer dans le système : enregistrement et contrôle d'une commande superviseur (PH_ECCSUP)

b) sélection d'un module suivant le type de la commande reçu (PH_SEL_APP)

POST_AP_SUP : dernière commande = commande de sortie du système.

NIVEAU 1

TITRE : enregistrement et contrôle commande superviseur

DEFINITION : lecture d'une commande et vérification de la
validité de celle-ci

FORMAT : PH_ECCSUP(message,fichier,fichier)

ARGUMENT :
- un message contenant le résultat de la lecture
- le fichier des commandes utilisateurs
- le fichier des messages d'erreurs

RESULTAT : enregistrement d'une commande superviseur

PRE_PH_ECCSUP : _

FONCTION : a) lecture d'une commande (FC_LECTLIGNE)
si commande vide alors retour à a)
b) contrôle de la commande lue
au niveau syntaxe (FC_MESSYNTAX) et
au niveau sémantique (FC_MESSEMANT)
c) si erreur envoit d'un message d'erreur
et retour à a)

POST_PH_ECCSUP : contenu du message = commande superviseur.

TITRE : sélection de modules du niveau 2

DEFINITION : module d'aiguillage vers des modules du
niveau 2

FORMAT : PH_SEL_APP(message,fichiers)

ARGUMENT :
- le message contenant le type de la commande
pour l'aiguillage
- des fichiers séquentiels nécessaires aux modules
du niveau 2

RESULTAT : appel du module du niveau 2 associé au type
de commande contenu dans le message reçu

PRE_PH_SEL_APP : le type de la commande dans le message est
de type "superviseur"

FONCTION : sélection du module du niveau 2
- éditeur assembleur (AP_EC)
- moniteur (AP_IP)
suivant le type
de la commande existant dans le message reçu

POST_PH_SEL_APP : module appellé = module sélectionné

NIVEAU 2

TITRE : éditeur assembleur

DEFINITION : lecture et/ou exécution d'une commande éditeur assembleur ou superviseur

FORMAT : AP_EC(message,fichiers)

ARGUMENT :
- message contenant la commande à réaliser
- fichiers séquentiels utlisés par les modules éditeur et assembleur

RESULTAT : appel d'un module éditeur assembleur de niveau 3 ou retour au superviseur

PRE_AP_EC : la commande contenue dans le message est du type superviseur associé à l'éditeur assembleur

FONCTION :
a) enregistrement et contrôle d'une commande éditeur assembleur ou superviseur (PH_ECCEC)
b) si commande du type éditeur assembleur alors exécution de la commande (PH_EXCEC) et retour à a)
c) retour au superviseur

POST_AP_EC :
dernière commande lue du type superviseur

TITRE : moniteur

DEFINITION : lecture et/ou exécution d'une commande moniteur
ou superviseur

FORMAT : AP_IP(message,fichiers)

ARGUMENT :
- message contenant la commande à réaliser
- fichiers séquentiels utlisés par le module moniteur

RESULTAT : appel d'un module moniteur de niveau 3
ou retour au superviseur

PRE_AP_EC : commande contenue dans le message du type superviseur associé au moniteur

FONCTION :
a) enregistrement et contrôle d'une commande moniteur ou superviseur (PH_ECCIP)
b) si commande moniteur alors :
1) affichage de l'ordinateur pedagogique (PH_AFFGRAPH)
2) initialisation de l'ordinateur pédagogique
3) exécution de la commande (PH_EXCIP)
4) retour à a)
c) retour au superviseur

POST_AP_EC :
dernière commande lue du type superviseur

NIVEAU 3

TITRE : enregistrement et contrôle commande éditeur assembleur

DEFINITION : lecture d'une commande et verification de la
validité de la commande lue

FORMAT : PH_ECCEC(message,fichier,fichier)

ARGUMENT :
- un message contenant le resultat de la lecture
- le fichier des commandes utilisateurs
- le fichier des messages d'erreurs

RESULTAT : enregistrement d'une commande éditeur assembleur

PRE_PH_ECCEC : _

FONCTION : a) lecture d'une commande (FC_LECTLIGNE)
si commande vide alors retour à a)
b) contrôle de la commande lue
au niveau syntaxe (FC_MESSYNTAX) et
au niveau sémantique (FC_SEMANTMES)
c) si erreur envoit d'un message d'erreur
et retour à a)

POST_PH_ECCEC : contenu du message = commande éditeur assembleur.
ou superviseur

TITRE : sélection de modules éditeur assembleur du niveau 4

DEFINITION : module d'aiguillage vers des modules de l'éditeur
assembleur niveau 4

FORMAT : PH_EXCEC(message,fichiers)

ARGUMENT :
- le message contenant le type de la commande pour l'aiguillage
- des fichiers séquentiels nécessaires aux modules du niveau 4

RESULTAT : appel du module du niveau 4 associé au type de commande contenu dans le message reçu

PRE_PH_EXCEC : le type de la commande dans le message est de type "éditeur assembleur"

FONCTION : sélection du module du niveau 4 suivant le type de la commande existant dans le message reçu
(ces modules n'ont pas encore été développés)

POST_PH_EXCEC : module appellé = module sélectionné

TITRE : enregistrement et contrôle commande moniteur

DEFINITION : lecture d'une commande et vérification de la
validité de la commande lue

FORMAT : PH_ECCIP(message,fichier,fichier)

ARGUMENT :
- un message contenant le résultat de la lecture
- le fichier des commandes utilisateurs
- le fichier des messages d'erreurs

RESULTAT : enregistrement d'une commande moniteur

PRE_PH_ECCSUP : _

FONCTION : a) lecture d'une commande (FC_LECTLIGNE)
si commande vide alors retour à a)
b) contrôle de la commande lue
au niveau syntaxe (FC_MESSYNTAX) et
au niveau sémantique (FC_MESSEMANT)
c) si erreur envoit d'un message d'erreur
et retour à a)

POST_PH_ECCIP : contenu du message = commande moniteur
ou superviseur.

TITRE : sélection de modules "moniteur" du niveau 4

DEFINITION : module d'aiguillage vers des modules du
"moniteur" du niveau 4

FORMAT : PH_EXCIP(message,fichiers)

ARGUMENT :
- le message contenant le type de la commande pour l'aiguillage
- des fichiers séquentiels nécessaires au module du niveau 4

RESULTAT : appel du module du niveau 4 associé au type de commande contenu dans le message reçu

PRE_PH_SEL_APP : le type de la commande dans le message est de type "moniteur"

FONCTION : sélection du module du niveau 4
- sauvetage de la mémoire (FC_SAUVMEM)
- garnissage de la mémoire centrale (FC_GMEMCENT)
- garnissage de l'accumulateur (FC_GREGA)
- garnissage du registre instruction (FC_GREGI)
- garnissage du compteur d'instructions (FC_GREGC)
- garnissage du registre extension (FC_REGE)
- garnissage de l'indicateur logique (FC_GREGL)
- initialisation de la machine (FC_INITMACH)
- réglage du temps d'attente entre chaque instruction exécutée (FC_RARINST)
- chargement de la mémoire centrale (FC_CHMEMCENT)
- exécution d'une instruction machine (FC_EXINST)
- exécution d'un programme machine (FC_EXPRG)
- chargement du lecteur de cartes (FC_CHLECT)
- initialisation du lecteur de cartes (FC_CARTINIT)
- rendre courante la première carte (FC_CARTUN)
- passer à la carte suivante (FC_CARTSUIV)
- passer à la carte précédente (FC_CARTPREC)
- avancer le papier de l'imprimante (FC_AVANCPAP)
- sauver le contenu de l'imprimante (FC_SAUVLIST)

suivant le type de la commande existant dans le message reçu

POST_PH_SEL_APP : module appellé = module sélectionné

TITRE : affichage graphique de l'ordinateur pédagogique

DEFINITION : affichage du contenu d'un fichier à l'écran
du terminal

FORMAT : PH_AFFGRAPH(fichier)

ARGUMENT : fichier séquentiel contenant le schéma graphique

RESULTAT : le contenu du fichier est affiché à l'écran

PRE_AFFGRAPH : (nombre de lignes dans fichier <
largeur de l'écran) ET
(nombre de caractères dans chaque ligne
<= longueur de l'écran)

FONCTION :
- effacement des caractères affichés à l'écran
(FC_NETECRAN)
- positionnement du curseur (FC_POSITCURS)
- placement du terminal en mode graphique
(MODE_GRAPHE)
- lecture (FC_LECTLIGNE) et affichage (FC_ECRLIGNE)
de chaque ligne du fichier contenant le graphique
- placement du terminal en mode texte
(MODE_ASCII)

POST_PH_AFFGRAPH : les lignes affichées à l'écran =
contenu du fichier reçu.

NIVEAU 4

TITRE : garnissage de la mémoire centrale

DEFINITION : remplissage des cellules de la mémoire centrale
de l'ordinateur pédagogique

FORMAT : FC_GMEMCENT(ordinateur,fichier)

ARGUMENT :
- la description interne de l'ordinateur pédagogique
- le fichier contenant les messages d'erreurs

RESULTAT : garnissage des données introduites par l'utilisateur
dans la mémoire de l'ordinateur pédagogique

PRE_FC_GMEMCENT : 0 <= adresse début de remplissage <=
nombre de cellules de la mémoire centrale

FONCTION :
- lecture de la donnée introduite par l'utilisateur
(FC_LECTLIGNE)
- si donnée = caractère de fin de remplissage alors
retour au module moniteur
- contrôle syntaxique de la donnée (FC_TESTNUM)
- si erreur envoit d'un message d'erreur et retour
au début de la fonction (FC_POSITCURS + FC_LECTLIGNE)
- garnissage de la cellule concernée (FC_GMOT)
- retour au début de la fonction

POST_FC_GMEMCENT : les mots mémoires valides introduitent par
l'utilisateur sont contenus dans la mémoire
centrale de l'ordinateur pédagogique

TITRE : garnissage de l'accumulateur

DEFINITION : remplissage de la donnée introduite par
l'utilisateur dans la composante
"accumulateur"

FORMAT : FC_GREGA(message,ordinateur)

ARGUMENT :
- un message contenant la donnée de l'utilisateur
- la représentation de l'ordinateur

RESULTAT : garnissage et affichage de la donnée dans
l'accumulateur de l'ordinateur pédagogique

PRE_FC_GREGA : la donnée doit être valide

FONCTION :
- formattage de la donnée pour obtenir la
forme de stockage voulue de l'information
(FC_TRANSFC)
- garnissage de la composante dans la
représentation interne de l'ordinateur
pédagogique
- affichage à l'écran, de la donnée introduite
par l'utilisateur dans l'accumulateur
(FC_POSITCURS + FC_ECRLIGNE)

POST_FC_GREGA : le contenu de l'accumulateur contient
la donnée introduite par l'utilisateur.

TITRE : garnissage du compteur d'adresses

DEFINITION : remplissage de la donnée introduite par
l'utilisateur dans la composante
"compteur d'adresses"

FORMAT : FC_GREGC(message,ordinateur)

ARGUMENT :
- un message contenant la donnée de l'utilisateur
- la représentation de l'ordinateur

RESULTAT : garnissage et affichage de la donnée dans le
compteur d'adresses de l'ordinateur pédagogique

PRE_FC_GREGC : la donnée doit être valide

FONCTION :
- formattage de la donnée pour obtenir la
forme de stockage voulue de l'information
(FC_TRANSFC)
- garnissage de la composante dans la
représentation interne de l'ordinateur
pédagogique
- affichage à l'écran, de la donnée introduite
par l'utilisateur dans le compteur d'instructions
(FC_POSITCURS + FC_ECRLIGNE)

POST_FC_GREGC : le contenu du compteur d'adresses contient
la donnée introduite par l'utilisateur.

TITRE : garnissage du registre instruction

DEFINITION : remplissage de la donnée introduite par
l'utilisateur dans la composante
"registre instruction"

FORMAT : FC_GREGI(message,ordinateur)

ARGUMENT :
- un message contenant la donnée de l'utilisateur
- la représentation de l'ordinateur

RESULTAT : garnissage et affichage de la donnée dans le
registre instruction de l'ordinateur pédagogique

PRE_FC_GREGI : donnée doit être valide

FONCTION :
- formattage de la donnée pour obtenir la
forme de stockage voulue de l'information
(FC_TRANSFC)
- garnissage de la composante dans la
représentation interne de l'ordinateur
pédagogique
- affichage à l'écran, de la donnée introduite
par l'utilisateur dans le registre instruction
(FC_POSITCURS + FC_ECRLIGNE)

POST_FC_GREGI : contenu de registre instruction contient
la donnée introduite par l'utilisateur.

TITRE : garnissage du registre extension

DEFINITION : remplissage de la donnée introduite par
l'utilisateur dans la composante
"registre extension"

FORMAT : FC_GREGE(message,ordinateur)

ARGUMENT :
- un message contenant la donnée de l'utilisateur
- la représentation de l'ordinateur

RESULTAT : garnissage et affichage de la donnée dans le
registre extension de l'ordinateur pédagogique

PRE_FC_GREGE : la donnée doit être valide

FONCTION :
- formattage de la donnée pour obtenir la
forme de stockage voulue de l'information
(FC_TRANSFC)
- garnissage de la composante dans la
représentation interne de l'ordinateur
pédagogique
- affichage à l'écran, de la donnée introduite
par l'utilisateur dans le registre extension
(FC_POSITCURS + FC_ECRLIGNE)

POST_FC_GREGE : le contenu de registre extension contient
la donnée introduite par l'utilisateur.

TITRE : garnissage de l'indicateur logique

DEFINITION : remplissage de la donnée introduite par
l'utilisateur dans la composante
"indicateur logique"

FORMAT : FC_GREGL(message,ordinateur)

ARGUMENT :
- un message contenant la donnée de l'utilisateur
- la représentation de l'ordinateur

RESULTAT : garnissage et affichage de la donnée dans
l'indicateur logique de l'ordinateur pédagogique

PRE_FC_GREGL : la donnée doit être valide

FONCTION :
- formattage de la donnée pour obtenir la
forme de stockage voulue de l'information
(FC_TRANSFC)
- garnissage de la composante dans la
représentation interne de l'ordinateur
pédagogique
- affichage à l'écran, de la donnée introduite
par l'utilisateur dans l'indicateur logique
(FC_POSITCURS + FC_ECRLIGNE)

POST_FC_GREGL : le contenu de l'indicateur logique contient
la donnée introduite par l'utilisateur.

TITRE : initialisation de l'ordinateur

DEFINITION : initialisation de l'ordinateur pédagogique dans la représentation interne et dans le schéma graphique

FORMAT : FC_INITMACH(ordinateur)

ARGUMENT : représentation interne de l'ordinateur pédagogique

RESULTAT : vidage du lecteur de cartes et de l'imprimante ; remise à blanc de l'indicateur logique et du code opératoire; remise à zéro de la mémoire centrale et des registres

PRE_FC_INITMACH : _

FONCTION :

- remplissage de zéros dans la mémoire centrale , les registres et le compte r d'instructions
- remplissage de caractères blanc dans l'indicateur logique, le code opératoire, l'imprimante et le lecteur de cartes
- modification de l'état de l'imprimante et du lecteur de cartes
- affichage du contenu de ces composantes à l'écran (FC_POSITCURS + FC_ECRLIGNE)

POST_FC_INITMACH : contenu de la mémoire centrale et des registres remis à zéro ET contenu du lecteur de cartes, de l'imprimante, de l'indicateur logique et du code opératoire remis à blanc.

TITRE : chargement de la mémoire centrale

DEFINITION : chargement du contenu d'un fichier dans la mémoire centrale de l'ordinateur pédagogique

FORMAT : FC_CHMEMCENT(message,ordinateur,fichier)

ARGUMENT : message contenant :
- le nom du fichier à charger,
- l'adresse de chargement du fichier
un fichier contenant les messages d'erreurs

PRE_FC_CHMEMCENT : _

FONCTION :

a) lecture du fichier externe identifié par le nom de fichier existant dans le message et écriture de celui-ci dans un fichier interne (FLECTURE)
b) si lecture réussi alors
   1) détection si le texte à charger contient une adresse de chargement dans la mémoire centrale
      si oui alors modification de la valeur du compteur d'adresses (FC_GREGC)
   2) transfert du fichier dans la mémoire centrale à partir de l'adresse spécifié par le compteur d'adresses (FC_GMOT)
   sinon envoit d'un message d'erreur (FC_ERRMON)

POST_FC_CHMEMCENT :
(chargement complet du fichier dans la mémoire centrale de l'ordinateur pédagogique)
OU
((chargement partiel du fichier dans la mémoire centrale de l'ordinateur pédagogique) ET (envoi d'un message d'erreur)
OU
((envoi d'un message d'erreur) ET (aucun chargement))

TITRE : règlage du temps d'attente entre chaque instruction exécutée

DEFINITION : modification du temps d'arrêt entre chaque instruction exécutée

FORMAT : FC_RARINST(message,ordinateur)

ARGUMENT :

- un message contenant la nouvelle valeur (sous forme caractères) du temps d'arrêt
- la représentation de l'ordinateur pédagogique contenant la zone de réception de cette valeur

RESULTAT :

le nombre de secondes séparant chaque instructions exécutées équivaut à la valeur de la donnée reçue

PRE_FC_RARINST : donnée numérique

FONCTION :

- transformation de la valeur reçue en un nombre (FC_CARNOMB)
- si le nombre est > 99 alors la nouvelle valeur = 99
  sinon la nouvelle valeur = la valeur reue

POST_FC_RARINST : valeur du temps d'arrêt = 99 si valeur reçue est > 99 sinon valeur du temps d'arrêt = valeur reçue

TITRE : exécution d'une instruction machine

DEFINITION : garnissage du registre instruction et exécution du contenu de celle-ci

FORMAT : FC_EXINST(ordinateur,reussi,fichiers)

ARGUMENT :

- représentation interne de l'ordinateur pédagogique
- une valeur booléenne signalant le succès de l'exécution
- un fichier contenant les messages d'erreurs
- un fichier contenant les codes opérations des instructions du langage ainsi que les types d'adressages qui leurs sont associés

RESULTAT : garnissage du registre instruction, décodage de celle-ci et exécution de l'instruction qui y est contenue

PRE_FC_EXINST : _

FONCTION :

- recherche de l'instruction identifiée par la valeur contenue dans le compteur d'adresses (FC_CONTORD)
- extraction de la zone opération du registre instruction et localisation de celle-ci dans le fichier des codes opératoires (FC_LOCALSYMB)
  si pas localisé alors réussi = FAUX
- si réussi = VRAI
  alors extraction du type d'adressage dans le registre instruction et recherche de celle-ci suivant le code opératoire reçu (FC_VALASS)
  si adressage invalide alors réussi = FAUX
- si réussi = VRAI
  recherche de l'adresse de l'opérande
  si adresse invalide alors réussi = FAUX
  (FC_CARNOMB)
- si réussi = VRAI
  1) branchement vers le groupe concerné
  2) exécution de l'instruction
  3) affichage et garnissage des composantes concernées
     (FC_TRANSFC)
     (FC_POSITCURS)
     (FC_NOMBCAR)
     (FC_ECRLIGNE)
     (FC_INCRMCO)
     (FC_GREGI)
     (FC_GREGC)

(FC_GREGA)
(FC_GREGE)
(FC_GMOT)
(FC_GCOP)

- si réussi = FAUX alors affichage d'un message d'erreur (FC_ERRMON)

POST_FC_EXINST :
((réussi = FAUX) ET (affichage d'un message d'erreur)
OU
(réussi = VRAI) ET (instruction contenue dans registre instruction est exécutée)

TITRE : exécution d'un programme machine

DEFINITION : exécution de la suite d'instructions commençant à l'adresse spécifiée par la valeur contenue dans le compteur d'instructions

FORMAT : FC_EXPRG(message,ordinateur)

ARGUMENT :
- un message contenant éventuellement le nombre d'instructions à exécuter
- la représentation interne de l'ordinateur pédagogique

RESULTAT : affichage des résultats d'exécution de chaque instruction machine rencontrée jusqu'à la rencontre de l'instruction de fin de programme ou après le nombre d'instructions demandé par l'utilisateur

PRE_FC_EXPRG : _

FONCTION :

a) exécution de l'instruction identifiée par le compteur d'instructions (FC_EXINST)
b) si temps d'attente entre chaque instruction est > 0 alors attente d'un laps de temps (FC_ATTENTE)
c) si (instruction de fin de programme rencontrée) OU (nombre d'instructions demandé par l'utilisateur exécutées) OU (exécution pas réussie) alors fin exécution du programme sinon recommencer em a)

POST_FC_EXPRG :

(instruction contenu dans le registre instruction représente la fin d'un programme machine)
OU
(nombre d'instructions exécutées demandé atteint)
OU
(exécution de l'instruction contenue dans le registre instruction non réussie)

TITRE : sauvetage de la mémoire centrale

DEFINITION : les cellules de la mémoire centrale qui sont désignées sont sauvées dans un fichier

FORMAT : FC_SAUVMEM(message,ordinateur,fichier)

ARGUMENT :
- un message contenant
  - le nom du fichier de réception
  - les adresses début et fin des cellules concernées par le sauvetage
- la représentation interne de l'ordinateur pédagogique
- un fichier contenant les messages d'erreurs

RESULTAT : transfert du contenu des cellules concernées dans le fichier de réception

PRE_FC_SAUVMEM : _

FONCTION :
a) extraction et écriture dans un fichier interne des cellules concernées
   (FC_CONTORD)
   (FC_ECRLIGNE)
b) écriture du fichier interne dans un fichier externe (FECRITURE)
c) si écriture dans fichier externe pas réussi alors écriture d'un message d'erreur (FC_ERRMON)

POST_FC_SAUVMEM :
(sauvetage réalisé)
OU
((affichage message d'erreur) ET
 (sauvetage pas réalisé))

TITRE : avancement du papier de l'imprimante

DEFINITION : avancement de l'image du papier de l'imprimante d'une ligne

FORMAT : FC_AVANCPAP(ordinateur)

ARGUMENT : la représentation interne de l'ordinateur pédagogique contenant la description de l'imprimante

RESULTAT :
avancement de chaque caractère de l'imprimante d'une ligne et écriture d'une ligne blanche

PRE_FC_AVANCPAP : _

FONCTION :
a) décalage d'une ligne de chaque caractère existant sur le papier de l'imprimante
b) écriture d'une ligne blanche au bas de la page
c) affichage du résultat
(FC_POSITCURS)
(FC_ECRLIGNE)

POST_FC_AVANCPAP :
(image de la page sur l'imprimante =
(image de la page sur l'imprimante)" -
ligne du sommet de la page +
ligne blanche au bas de la page)

TITRE : sauvetage du listing

DEFINITION : copie de la page de l'imprimante dans un fichier

FORMAT : FC_SAUVLIST(message,ordinateur,fichier)

ARGUMENT :
- un message contenant le nom du fichier de réception
- la représentation interne de l'ordinateur pédagogique
- un fichier contenant les messages d'erreurs

RESULTAT : transfert du contenu de la page concernée dans le fichier de réception

PRE_FC_SAUVMEM : _

FONCTION :
a) écriture dans un fichier interne de la page concernée (FC_ECRLIGNE)
b) écriture du fichier interne dans un fichier externe (FECRITURE)
c) si l'écriture dans le fichier externe n'est pas réussie alors écriture d'un message d'erreur (FC_ERRMON)

POST_FC_SAUVMEM :
(sauvetage réalisé)
OU
((affichage message d'erreur) ET
(sauvetage pas réalisé))

TITRE : chargement du lecteur de cartes

DEFINITION : chargement du contenu d'un fichier dans le lecteur de cartes de l'ordinateur pédagogique

FORMAT : FC_CHLECT(message,ordinateur,fichier)

ARGUMENT : message contenant :
- le nom du fichier à charger,
- l'adresse de chargement du fichier
un fichier contenant les messages d'erreurs

RESULTAT :
un nombre entier de cartes sont extraites du fichier désigné et chargées dans le lecteur de cartes suivant la capacité de ce dernier

PRE_FC_CHMEMCENT : _

FONCTION :
a) lecture du fichier externe identifié par le nom de fichier existant dans le message et écriture de celui-ci dans un fichier interne (FLECTURE)
b) si lecture réussi alors
alors extraction d'un nombre entier de cartes et transfert de ces cartes dans le lecteur de cartes
et affichage du chargement (FC_AFFCARTE)
sinon envoit d'un message d'erreur (FC_ERRMON)

POST_FC_CHMEMCENT :
(chargement complet du fichier dans la mémoire centrale de l'ordinateur pédagogique)
OU
((chargement partiel du fichier dans la mémoire centrale de l'ordinateur pédagogique)
OU
((envoi d'un message d'erreur) ET (aucun chargement))

TITRE : rendre courant la carte précédente

DEFINITION : la carte qui précède la carte courante
est placée devant la tête de lecture
du lecteur de cartes

FORMAT : FC_CARTPREC(ordinateur,fichier)

ARGUMENT :
- représentation interne de l'ordinateur pédagogique avec l'état du lecteur de cartes
- un fichier contenant les messages d'erreurs

RESULTAT : la carte qui précède la carte courante
devient courante, les autres cartes
reculent d'une position dans le lecteur
de cartes

PRE_FC_CARTPREC : _

FONCTION :
s'il existe encore une carte qui
précède la carte courante alors
rendre courant cette carte et
affichage du nouvel état du lecteur
cartes à l'écran (FC_AFFCARTE)
sinon affichage d'un message
d'erreur (FC_ERRMON)

POST_FC_CARTPREC:
(carte courante = (carte précédant la (carte
courante)")
OU
((carte courante) = (carte courante)"
ET affichage d'un message d'erreur)

TITRE : rendre courant la carte suivante

DEFINITION : la carte qui suit la carte courante
est placée devant la tête de lecture
du lecteur de cartes

FORMAT : FC_CARTSUIV(ordinateur,fichier)

ARGUMENT :
- représentation interne de l'ordinateur pédagogique avec l'état du lecteur de cartes
- un fichier contenant les messages d'erreurs

RESULTAT : la carte qui suit la carte courante
devient courante, les autres cartes
avancent d'une position dans le lecteur
de cartes

PRE_FC_CARTSUIV : _

FONCTION :
s'il existe encore une carte qui
suit la carte courante alors
rendre courant cette carte et
affichage du nouvel état du lecteur
de cartes à l'écran (FC_AFFCARTE)
sinon affichage d'un message
d'erreur (FC_ERRMON)

POST_FC_CARTSUIV:
(carte courante = (carte succédant la (carte courante)")
OU
((carte courante) = (carte courante)"
ET affichage d'un message d'erreur)

TITRE : rendre courant la première carte du paquet

DEFINITION : la première carte du paquet de cartes
est placée devant la tête de lecture
du lecteur de cartes

FORMAT : FC_CARTUN(ordinateur,fichier)

ARGUMENT :
- représentation interne de l'ordinateur pédagogique avec l'état du lecteur de cartes
- un fichier contenant les messages d'erreurs

RESULTAT : la première carte du paquet de cartes existant dans le lecteur de cartes devient courante

PRE_FC_CARTUN : _

FONCTION :
s'il existe un paquet de cartes
rendre courante la première carte
de ce paquet et
affichage du nouvel état du lecteur
cartes à l'écran (FC_AFFCARTE)
sinon affichage d'un message
d'erreur (FC_ERRMON)

POST_FC_CARTUN:
(carte courante = première carte du paquet de cartes)
OU
affichage d'un message d'erreur

TITRE : initialisation du lecteur de cartes

DEFINITION : vidage du contenu du lecteur de cartes

FORMAT : FC_CARTINIT(ordinateur)

ARGUMENT : représentation interne de l'ordinateur pédagogique contenant le lecteur de cartes

RESULTAT : initialisation de l'état du lecteur de cartes

PRE_FC_CARTINIT : _

FONCTION : modification de l'état du lecteur de cartes et affichage du nouvel état (FC_POSITCURS + FC_ECRLIGNE)

POST_FC_CARTINIT :
contenu du lecteur de cartes = cartes vides (blanches)

NIVEAU 5

TITRE : garnissage d'un mot de la mémoire centrale

DEFINITION : remplissage d'une cellule de la mémoire centrale de la valeur reçue

FORMAT : FC_GMOT(valeur_mot,no_cellule,réussi)

ARGUMENT : valeur_mot représentant la chaîne de caractères contenant la donnée pour le remplissage
no_cellule représentant le numéro de la cellule de réception de la donnée
réussi une valeur booléenne signalant la réussite de l'opération

RESULTAT : la donnée contenue dans valeur_mot est copiée dans la cellule mémoire définit par no_cellule

PRE_FC_GMOT : _

FONCTION :

a) transformation du numéro de la cellule désignée en coordonnées ligne et colonne pour le tableau "mémoire centrale"
b) si les coordonnées appartiennent aux bornes du tableaux alors extraction de la donnée (FC_LECTSYMB) et formattage de celle-ci (FC_TRANSFC) ensuite remplissage de la cellule concernée et affichage du résultat (FC_POSITCURS + FC_ECRLIGNE) et réussi = VRAI
sinon réussi = FAUX

POST_FC_GMOT :
((réussi = VRAI) ET (contenu de la cellule) appartient à valeur_mot
OU
(réussi = FAUX))

TITRE : incrémentation du compteur d'instructions

DEFINITION : incrémentation de 1 de la valeur contenue dans le compteur d'instructions

FORMAT : FC_INCREMCO(ordinateur,ancien_valeur,réussi)

ARGUMENT :
- représentation interne de l'ordinateur pédagogique
- un nombre représentant l'ancienne valeur du compteur d'instructions
- réussi une valeur booléenne signalant le succès de l'opération

RESULTAT : le compteur d'instructions est incrémenté de 1

PRE_FC_INCREMCO : la valeur contenue dans le compteur d'instructions doit être numérique

FONCTION :
a) transformation de la chaîne de caractère représentant le contenu du compteur d'instructions en un nombre
(FC_CARNOMB)
b) copie du nombre obtenue dans ancien_valeur
c) si le nombre est < adresse maximun de la mémoire alors :
- incrémentation de 1 de cette valeur
- transformation du nombre incrémenté en une chaîne de caractères
(FC_NOMBCAR)
- formattage de la chaîne de caractères provenant du nombre incrémenté
(FC_TRANSFC)
- garnissage et affichage du résultat
(FC_GREGC)
- réussi = VRAI
sinon réussi = FAUX

POST_FC_INCREMCO :
(ancien_valeur = (compteur d'instructions)")
ET
(((réussi = VRAI) ET (contenu du compteur d'instruction) = (compteur d'instructions)" + 1))
OU
(réussi = FALSE))

```
TITRE            : extraction d'une valeur contenue dans
                   l'ordinateur pédagogique

DEFINITION       : le contenu de la composante sélectionnée
                   est extraite

FORMAT           : FC_CONTORD(nom_composante,no_cellule,
                              zone_recept,ordinateur,réussi)

ARGUMENT         :
                 - le nom de la composante sélectionné
                 - le numéro de la cellule mémoire considérée
                   si la composante est la mémoire centrale
                 - une zone de réception de la donnée extraite
                 - la représentation interne de l'ordinateur
                   pédagogique
                 - une valeur booléenne (réussi) signalant la
                   le succès de l'opération

RESULTAT         : le contenu de la zone de réception =
                   contenu de la composante sélectionnée

PRE_FC_CONTORD   : _

FONCTION         :
                 a) si le nom de la composante n'existe pas
                     alors réussi = FAUX
                 b) si le nom de la composante existe
                    alors
                     si le nom de la composante = "mémoire
                         centrale" alors
                         transformation du numéro de la
                         cellule en coordonnée ligne et
                         colonne et si elle appartient
                         aux bornes de la mémoire centrale
                         alors remplissage de la zone de
                         réception sinon réussi = FAUX
                       si le nom de la composante est différent
                         de "mémoire centrale"
                         alors remplissage de la zone
                         de réception

POST_FC_CONTORD  :
                 ((réussi = VRAI)
                  ET
                  (contenu de la zone de réception) =
                  (contenu de la composante sélectionnée))
                 OU
                 (réussi = FAUX)
```

TITRE : garnissage du code opératoire

DEFINITION : remplissage d'une valeur dans le code opératoire

FORMAT : FC_GCOP(valeur,ordinateur)

ARGUMENT :
- la valeur à ranger dans le code opératoire
- la représentation interne de l'ordinateur pédagogique

RESULTAT : la valeur reçue est transferée dans la composante "code opératoire" de l'ordinateur pédagogique

PRE_FC_GCOP : _

FONCTION :
- extraction et formattage de la donnée reçue (FC_TRANSFC)
- garnissage de la composante
- affichage du résultat
  (FC_POSITCURS)
  (FC_ECRLIGNE)

POST_FC_GCOP : contenu du code opératoire = valeur reçue

TITRE : affichage de cartes

DEFINITION : affichage de la carte courante et de celles qui la suivent

FORMAT : FC_AFFCARTE(ordinateur)

ARGUMENT :
- la représentation de l'ordinateur pédagogique avec l'état du lecteur de cartes

RESULTAT :
affichage à l'écran de la carte courante et de celles qui la suivent

PRE_FC_AFFCARTE : le numéro de la carte courante doit être comprise entre [1,..,nombre maximun de cartes stockables]

FONCTION :
a) prendre la carte courante et les cartes qui la suivent
b) afficher le maximun possible de cartes à l'écran,
s'il n'y pas assez de cartes alors remplissage de cartes blanches
(FC_POSITCURS)
(FC_ECRLIGNE)

POST_FC_AFFCARTE:
affichage de la carte courante et de celles qui la suivent ou affichage unique de cartes blanches

TITRE : affichage d'un message d'erreur moniteur

DEFINITION : affichage dans la zone d'affichage du schéma graphique d'un message d'erreur

FORMAT : FC_ERRMON(code_erreur,fichier)

ARGUMENT :
- un code erreur représentant la position du message d'erreur dans le fichier des messages d'erreurs
- le fichier des messages d'erreurs

RESULTAT : affichage du message contenu dans l'enregistrement fichier(code erreur)

PRE_FC_ERRMON : le code erreur doit représenter une position existante dans le fichier des messages

FONCTION :
- lecture du message suivant sa position dans le fichier (FC_VALASS)
- extraction du message (FC_LECTSYMB)
- formattage du message (FC_TRANSFC)
- affichage du message
  (FC_POSITCURS)
  (FC_LECTLIGNE)

POST_FC_ERRMON : message affiché = contenu du fichier à la position "code erreur"

NIVEAU 6

TITRE : lecture d'une chaîne de caractères

DEFINITION : lecture d'une suite de caractères
se trouvant dans la ligne courante
d'un fichier

FORMAT : FC_LECTLIGNE(zone_recept,nb_caract,fichier)

ARGUMENT :
zone_recept = zone de réception des caractères lus
nb_caract = nombre maximun de caractère à lire
fichier = l'adresse de lecture dans le fichier concerné

RESULTAT :
garnissage dans la zone de réception
des caractères contenus dans la ligne
courante du fichier considéré

PRE_FC_LECTLIGNE : nb_caract < longueur maximun de la
zone de réception
ET
fin du fichier pas atteint

FONCTION : lecture d'une chaîne de caractères dans
la ligne courante du fichier jusqu'à
la fin de la ligne ou jusqu'à la
longueur désirée

POST_FC_LECTLIGNE : (caractères lus appartiennent à (ligne
courante)" du fichier)
ET
(positionnement à la ligne suivante
du fichier si elle existe)

TITRE : analyse syntaxique de messages

DEFINITION : analyse du contenu d'une chaîne de caractères

FORMAT : FC_MESSYNTAX(nom_expediteur,chaine_caract, message,fichier)

ARGUMENT :
- nom_expediteur identifie le nom de l'expéditeur
- chaine_caract contient les caractères à analyser
- message représente le résultat de l'analyse
- fichier des commandes "utilisateur"

RESULTAT :
le message contient le résultat de l'analyse

PRE_FC_MESSYNTAX : _

FONCTION :
a) extraction des différents symboles non vides contenus dans la chaîne de caractères reçus (FC_LECTSYMB)
b) vérification de la validité des symboles extraits
   - validité de la commande (FC_LOCALSYMB)
   - validité des paramètres (FC_TESTNUM)
c) si validité correcte alors état du message = bon sinon état du message = mauvais

POSY_FC_MESSYNTAX :
((état du message = bon)
ET
(type de la commande et paramètres contenus dans le message))
OU
(état du message = mauvais)

TITRE : analyse sémantique de messages

DEFINITION : analyse des paramètres d'un message suivant le type de la commande

FORMAT : FC_MESSEMANT(nom_expediteur,message)

ARGUMENT :
- nom de l'expéditeur identifiant les types de test à réaliser
- le contenu du message à analyser

RESULTAT : le message contient le résultat de l'analyse

PRE_FC_MESSEMANT : type de la commande contenu dans le message existe

FONCTION :
vérification de la validité des paramètres suivant le type de la commande reçu (FC_TESTNUM)

POST_FC_MESSEMANT :
(état du message = bon) ou
(état du message = mauvais)

TITRE : écriture d'une chaine de caractères

DEFINITION : écriture dans un fichier séquentiel
d'une chaine de caractères

FORMAT : FC_ECRLIGNE(ligne_a_ecrire,long_ecrire,fichier)

RESULTAT :
garnissage à la fin du fichier considéré
des caractères reçus

ARGUMENT :
- la chaine de caractères à écrire
- la longueur de l'écriture
- le fichier de réception de la ligne

PRE_FC_ECRLIGNE :
- état du fichier séquentiel = fin de fichier
- long_ecrire < longueur de la ligne à écrire

FONCTION :
- écriture de la chaine de caractères dans
le fichier concerné sur la longueur
= min[fin de la chaine de caractères,longueur
demandé pour l'écriture]

POST_FC_ECRLIGNE :
dernière ligne du fichier = contenue de
la chaine de caractères à écrire

TITRE : transformation d'une chaine de caractères
en un nombre entier positif

DEFINITION : chaque caractère numérique contenu dans
une chaine de caractères est transformé
en un chiffre digital et combiné
pour formé un nombre entier positif

FORMAT : FC_CARNOMB(chaine_de_caractères,nombre)

ARGUMENT :
- la chaine de caractère contenant une
suite de caractères numériques
- le nombre entier positif obtenu après
transformation

RESULTAT :
f(chaine_de_caractères) ---> nombre

PRE_FC_CARNOMB : la suite de caractères numériques doit
représenter un nombre entier positif
et inférieur à la représentation du
entier maximun

FONCTION :
extraction de chaque caractère numérique
de la chaine de caractères et transformation
en un chiffre décimal, combinaison des
chiffres obtenus pour former le nombre
recherché

POST_FC_CARNOMB :
(nombre = 0)
OU
(nombre = représentation de la suite
de caractères numériques)

TITRE : transformation d'un nombre entier en une chaîne de caractères

DEFINITION : extraction de chaque chiffre du nombre reçu et transformation en un caractère alphanumérique

FORMAT : FC_NOMBCAR(nombre,chaîne_de_caractères)

ARGUMENT :
- nombre représentant un nombre entier positif
- chaîne_de_caractères contient le résultat de la transformation

RESULTAT : f(nombre) --> chaîne_de_caractères

PRE_FC_NOMBCAR : nombre reçu <= plus grand nombre entier

FONCTION :
a) si signe du nombre < 0
   alors chaîne_de_caractères[1] = signe moins
   sinon chaîne_de_caractères[1] = signe plus
b) rendre positif le nombre reçu
c) extraction des chiffres du nombre et codage sous forme caractères

POST_FC_NOMBCAR : contenu de la chaîne_de_caractères = représentation du nombre

```
TITRE : localisation d'un symbole;

DEFINITION : localisation d'un symbole donnè dans un fichier
             sèquentiel;

FORMAT :
  FC_LOCALSYMB(fichier,chaine_caractere,pos_symbole,pos_fichier);

ARGUMENT :
        - un fichier sèquentiel contenant des chai^nes de
          caracte`res, le symbole est la premie`re suite de
          caracte`res significatives (diffèrents du caracte`re
          blanc),
        - une chai^ne de caracte`res contenant premie`res
          positions le symbole a` localiser dans  le fichier,
          ce symbole est suivi d'une sentinnelle,
        - premie`re position du symbole dans la chai^ne de
          caracte`res;
        - position du symbole dans le fichier;

RESULTAT :
        position du symbole > 0 si symbole est trouvè sinon
        elle vaut 0;

PRE_FC_LOCALSYMB :
        (1 <= pos_symbole <= position sentinelle <= longueur
        maximun dans la chai^ne de caracte`res) ET
        ((symbole appartient au fichier) OU (symbole n'appartient
         pas au fichier))

FONCTION :
        -  positionnement dèbut du fichier
       a) lecture enregistrement (FC_LECTLIGNE)
       b) lecture du premier symbole de l'enregistrement (FC_LECTSYMB)
       c) comparer le symbole lu à celui recherchè
       d) s'ils sont diffèrents alors aller en a) si
          tout les enregistrements n'ont pas encore ètè
          lus
       e) placer dans pos_fichier le numèro relatif de
          l'enregistrement lu;

POST_FC_LOCALSYMB :

        ((pos_fichier > 0) ET (symbole appartient à
         enregistrement[pos_fichier]) ET
         (pos_fichier appartient [1..nombre d'enregistrements
         du fichier]))
        OU
        ((pos_fichier = 0) ET ( pour tout i appartenant à
         [1..nombre d'enregistrements du fichier]; symbole
         n'appartient pas à enregistrement[i])).
```

TITRE : recherche valeur;

DEFINITION : accès et lecture d'un enregistrement dans un fichier
séquentiel;

FORMAT : FC_VALASS(fichier,cle_enregistrement,enregistrement);

ARGUMENT :
- un fichier séquentiel contenant des chaînes de caractères alphanumériques,
- une valeur de clé d'un enregistrement se trouvant dans le fichier, cette clé représente la position relative de l'enregistrement par rapport au début du fichier,
- une chaîne de caractères contenant le contenu de l'enregistrement lu;

RESUTLAT : enregistrement = fichier(clé);

PRE_FC_VALASS : (fichier existe) ET (clé appartient à [1.. nombre d'enregistrement du fichier])

FONCTION :
- po itionnement debut du fichier,
- saut de ((clé) - 1) chaînes de caractères
- lecture de la chaîne de caractères;

POST_FC_VALASS : fichier(clé) = enregistrement

```
TITRE : lecture d' un fichier externe;

DEFINITION : ce module va copier dans un fichier interne
             le contenu d'un fichier externe;

FORMAT : FLECTURE( nom_du_fichier_émetteur,
                   resultat_copie,
                   fichier_interne); ARGUMENT :
         - nom_du_fichier_émetteur : chaîne de caractères
           contenant le nom du fichier ainsi qu'une sentinelle
           de fin de nom;
         - resultat_copie : boolèen;
         - fichier_interne : fichier séquentiel vide;

RESULTAT         :
         le contenu du fichier identifié par nom_du_fichier_
         destination est rangé dans le fichier interne;

PRE_FLECTURE     :
            (sentinelle existe dans nom_du_fichier_destination)
                 ET (position sentinelle différente de 1)

FONCTION : a) appel du module système qui ouvrira le fichier
              externe suivant le nom de fichier contenue dans
              nom_du_ficher_destination,
           b) lecture du fichier externe et écriture dans fichier
              interne de chaque caractère jusqu'à la rencontre
              du fin de fichier dans le fichier externe;

POST_FLECTURE :
              (contenu du fichier interne = contenu du fichier
               externe et resulat_copie = VRAI ) ou resulat_
               copie = FAUX.
```

TITRE : écriture dans un fichier externe;

DEFINITION : ce module va copier dans un fichier externe
le contenu d'un fichier interne;

FORMAT : FECRITURE(nom_du_fichier_destination,
resultat_copie,
fichier_interne);

ARGUMENT :
- nom_du_fichier_destination : chaîne de caractères contenant le nom du fichier ainsi qu'une sentinelle de fin de nom
- resultat_copie : booléen;
- fichier_interne : fichier séquentiel contenant une suite de caractères alphanumériques;

RESULTAT : le contenu du fichier identifié par nom_du_fichier_destination représente le contenu du fichier interne;

PRE_FECRITURE :
(sentinelle existe dans nom_du_fichier_destination) ET
(position sentinelle différente de 1)

FONCTION : a) appel du module système qui crèera le fichier externe suivant le nom de fichier contenu dans nom_du_fichier_destination,
b) lecture du fichier interne et écriture dans fichier externe de chaque caractère jusqu'à la rencontre du fin de fichier dans le fichier interne;

POST_FECRITURE : (contenu du fichier externe = contenu du fichier interne ET resultat_copie = VRAI) OU resulat_copie = FAUX.

NIVEAU 7

TITRE : nettoyage de l'écran

DEFINITION : effacement des caractères affichés à l'écran du terminal

FORMAT : FC_NETECRAN

ARGUMENT : _

RESULTAT : effacement des caractères affichés à l'écran du terminal

PRE_FC_NETECRAN : _

FONCTION : appel à une procédure du système qui réalise l'opération

POST_FC_NETECRAN: aucun caractère affiché à l'écran

```
TITRE           : positionnement du curseur à l'écran

DEFINITION      : positionnement de la tête d'écriture
                  et de lecture sur l'écran du terminal

FORMAT          : FC_POSITCURS(ligne_curseur,
                               colonne_curseur)

ARGUMENT        :
                  le numéro de la ligne et le numéro
                  de la colonne du curseur sur l'écran

RESULTAT        : curseur placé à l'endroit définit
                  par les coordonnées reçues

PRE_FC_POSITCURS:
                  (1 <= ligne_curseur <= (largeur-ecran - 1)
                  ET
                  (1 <= colonne_curseur <= longueur-ecran))

FONCTION        :
                  appel à la procédure système qui réalise
                  cette opération

POST_FC_POSITCURS:
                  position curseur = écran[ligne_curseur,
                          colonne_curseur]
```

TITRE : arrêt d'un laps de temps

DEFINITION : arrêt de l'exécution du processus en cours suivant une certaine durée

FORMAT : FC_ATTENTE(secondes)

ARGUMENT : secondes = nombre entier positif précisant le nombre de secondes pour l'arrêt

RESULTAT : aucune exécution pendant le nombre de secondes demandées

PRE_FC_ATTENTE : 0 <= secondes <= nombre entier maximun

FONCTION :
appel à une procédure du système qui réalise cette opération

POST_FC_ATTENTE : temps d'inactivité = nombre de secondes d'arrêt

TITRE : lecture d'un caractère;

DEFINITION : ce module réalise la lecture d'un caractère
dans un fichier séquentiel;

FORMAT : FC_LECTC(caractère,fichier_interne);

ARGUMENT :
- un caractère alphanumérique;
- un fichier_interne qui est un fichier séquentiel contenant une suite de caractères alphanumériques, et dont la position courante dans le fichier est celui qui est implicitement considérée

RESULTAT :
la lecture du caractère se trouvant à la position courante du fichier reçu

PRE_FC_LECTC :
la position courante dans le fichier doit être différente de la fin du fichier

FONCTION :
- lecture du caractère se trouvant à la position courante du fichier
- incrémentation de la position courante du fichier interne

POST_FC_LECTC : fichier_interne(position courante - 1) =
caractère lu

TITRE : écriture d'un caractère;

DEFINITION : ce module réalise l'écriture d'un caractère
dans un fichier séquentiel;

FORMAT : FC_ECRITC(caractère,fichier_interne);

ARGUMENT :
- un caractère alphanumérique;
- un fichier_interne qui est un fichier séquentiel contenant une suite de caractères alphanumériques, et dont la position courante dans le fichier est celui qui suit le dernier caractère du fichier;

RESULTAT : écrture du caractère reçu à la fin du
fichier_interne reçu;

PRE_FC_ECRITC : fichier_interne(dernière position) =
fin_de_fichier

FONCTION :
- copie du caractère reçu a la dernière position du fichier
- incrémentation de la position courante du fichier interne;

POST_FC_ECRITC : fichier_interne(position courante - 1) =
caractère reçu

```
TITRE : test numérique;

DEFINITION : ce module vérifie si une chaîne de caractères
                est numérique;

FORMAT : TESTNUM(chaine_caractere,pos_debut_test);

ARGUMENT :
        - une chaîne de caractères contenant une suite de
          caractères à tester, cette dernière est suivie
          d'une sentinelle,
        - la position du premier caractère à tester;

RESULAT : TESTNUM = VRAI  si la chaîne considérée est
           numérique sinon = FAUX;

PRE_TESTNUM :
         (1 < pos_debut_test < longueur maximun de la chaîne)
         ET (pos_debut_test <= position de la sentinelle <=
          longueur maximun de la chaîne);

FONCTION :
        - vérifier si chaque caractère appartenant à
         chaine_caractere[pos_debut_test,..,position de la
         sentinelle] est une caractère numérique;


POST_TESTNUM :
        ((TESTNUM = TRUE) ET (pour tout i appartenant à
        l'intervalle [pos_debut_test,..,position sentinelle],
        chaîne_caractere[i] appartient ['0',..'9']))
        OU
        ((TESTNUM = FALSE) ET ( il existe un i appartenant à
        l'intervalle [pos_debut_test,..,position sentinelle],
        tel    que    chaîne_caractere[i]    n'appartient     pas
        ['0',..'9']))
```

```
TITRE : transfert d'une chaîne de caractères;

DEFINITION : transfert d'une chaîne de caractères d'une zone
             émettrice vers une zone réceptrice;

FORMAT : FC_TRANSFC(emetteur,recepteur,
                    long_emetteur,long_recepteur,
                    test_signe,caractere_de_remplissage);

ARGUMENT :
        - un émetteur représentant une chaîne de caractères
          alphanumériques et contenant dans les premières
          positions une chaîne de caractères significatives,
        - un récepteur représentant une chaîne de caractères,
        - la longueur de la chaîne de caractères significatives
          dans l'émetteur,
        - la longueur desirée de la chaîne dans la zone de
          réception,
        - une valeur booléenne signalant s'il faut considérer
          que le premier caractère dans l'émetteur est un signe,
        - un caractère de remplissage si la longueur de la chaîne
          significative est inférieur à la longueur de la chaîne
          désirée;

RESULTAT  :
     recepteur[1..long_recepteur] = emetteur[1..long_emeteur]
                          + caractère de remplissage;

PRE_FC_TRANSF : long_emetteur <= long_recepteur;

FONCTION :
        - transfert en cadrant à droite dans la zone de réception
          sur une longueur définit par long_recepteur,
        - si test_signe = TRUE  alors  placé le  signe dans la
            première position de la zone réceptrice
          (signe = "-" OU " ")
        - remplir les positions inoccupées par le caractère de
          remplissage;

POST_FC_TRANSF :
        ((test_signe = TRUE) ET (recepteur[1] = signe) ET
       (recepteur[1 + 1..(long_recepteur - (long_emetteur - 1))] =
                caractère de remplissage) ET
         (recepteur[(long_recepteur - long_emetteur- 1),
         ..long_recepteur] = emetteur[2..long_emetteur
        OU
        ((test_signe = FALSE) ET (recepteur[1..(long_recepteur -
         long_emetteur)] = caractère de remplissage) ET
         (recepteur[long_recepteur - long_emetteur,
         ..long_recepteur] = emetteur[1..long_emetteur]
```

TITRE : changer le mode du terminal en mode texte

DEFINITION : le terminal affiche des caractères ASCII

FORMAT : MODE_ASCII

ARGUMENT : _

RESULTAT : tout caractère envoyé vers l'écran du terminal sera interprété comme un caractère ASCII

PRE_MODE_ASCII : _

FONCTION : appel à un procédure du système qui réalise cette opération

POST_MODE_ASCII :
tout caractère qui est affiché appartient au vocabulaire ASCII

TITRE : changer le mode du terminal en mode graphique

DEFINITION : tout caractère envoyé à l'écran du terminal sera interprété comme étant un caractère graphique

FORMAT : MODE_GRAPHE

ARGUMENT : _

RESULTAT : tout caractère envoyé vers l'écran du terminal sera interprété comme un caractère graphique

PRE_MODE_GRAPHE : _

FONCTION : appel à une procédure du système qui réalise cette opération

POST_MODE_GRAPHE:
tout caractère qui est affiché appartient au vocabulaire du langage graphique

Modules de MAINTENANCE

TITRE : génération du schéma graphique

DEFINITION : élaboration du schéma graphique de l'ordinateur pédagogique

FORMAT : GRAPH(fichier,fichier)

ARGUMENT :
- fichier séquentiel contenant éventuellement l'image du schéma graphique
- fichier séquentiel contenant éventuellement les nouvelles données des composantes du schéma graphique

RESULTAT :
génération du schéma graphique et des données des composantes de ce schéma et stockage de celles-ci dans des fichiers

PRE_GRAPH : _

FONCTION :
a) calcul pour chaque composante de la dimension et de la localisation de celle-ci
b) vérification si la composante peut être contenue entièrement dans le schéma graphique qui devra être affiché à l'écran
c) si la vérification ne détecte pas d'anomalies alors
    - affichage du schéma graphique à l'écran
    - demande de confirmation à l'utilisateur si le graphique affiché correspond au schéma souhaité
    - si réponse affirmative alors
        - stockage du graphique
        - calcul des nouvelles données de chaque composante et sauvetage dans un fichier (ZONRECP)
      sinon il y a aucun sauvetage
  sinon affichage du numéro de l'anomalie rencontrée

POST_GRAPH :
(message d'erreur affiché)
OU
((graphique affiché)
    ET
    ((sauvetage du schéma et des nouvelles données)
     OU
    (pas de sauvetage)))

TITRE : génération d'un fichier de données

DEFINTION : calcul à partir de données reçues
de nouvelles données et sauvetage
de celles-ci dans un fichier

FORMAT : ZONRECP(fichier)

ARGUMENT : fichier séquentiel contenant
les nouvelles données

RESULTAT : fichier contient les données
associées au nouveau schéma
graphique généré

PRE_ZONRECP : _

FONCTION :

calcul pour chaque composante
du schéma graphique généré
de sa dimension ainsi que de
sa localisation dans le
schéma graphique.
La localisation et la dimension
correspondent à la position
du premier caractère stockable
dans la composante et la longueur
maximun de la donnée stockable
dans la composante

POST_ZONRECP :
toute nouvelle donnée qui est
associée à une composante correspond
à des valeurs qui sont comprises dans
chaque composante du schéma graphique

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

ETUDE, CONCEPTION ET REALISATION

D'UN ORDINATEUR PEDAGOGIQUE.

(ANNEXE 2)

Mémoire présenté par

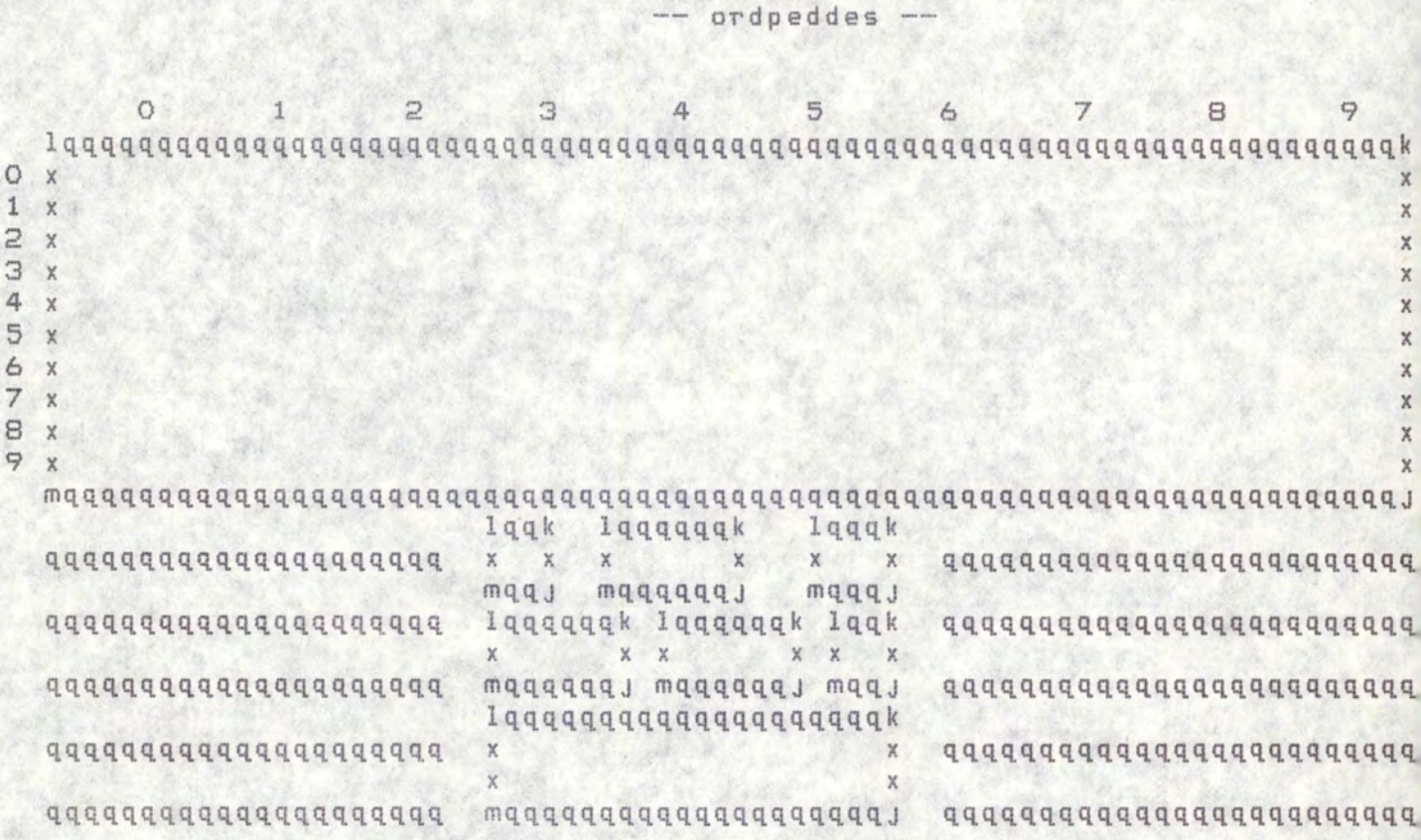
Johnny Vendrix

en vue de l'obtention
du titre de
Licencié et Maître en Informatique

Annee académique 1980-1981

TABLE DES MATIERES.

LES FICHIERS UTILISES.


-- ordpeddes --
0 1 2 3 4 5 6 7 8 9

```
?                       (* message attente d'une donnee *)
ERR_MES_ENT             (* message erreur d'une donnee incorrecte *)
ERR_OPER                (* messages erreur a l'execution *)
ERR_ADR
ARG_NON_VAL
CO_NON_VAL
```

-- ftabcomd --

```
E       (*  commande superviseur *)
M
Q
L
R
I       (* commande editeur compilateur *)
D
P       (* cette table doit etre en concordance avec la liste *)
-       (* TYPE 1_2 dans le fichier fmestype, tout changement *)
S       (* dans l'un doit entrainer un changement dans l'autre*)
T
V
N1/nouvelle_commande
N2/nouvelle_commande
N3/nouvelle_commande
N4/nouvelle_commande
N5/nouvelle_commande
FM      (* commande interpreteur pupitreur *)
?
FI
FA
FC
FE
FL
ZM
LM
SI
EI
EP
SM
NL
SL
IC
LC
NC
-C
1C
M1/nouvelle_commande
M2/nouvelle_commande
M3/nouvelle_commande
M4/nouvelle_commande
M5/nouvelle_commande
```

```
10 012345 LDA    CE FICHIER REPREND LES DIFFERENTES CODES OPERATOIRES
11 12345  STA    DU LANGAGE MACHINE
12 12345  EXC    LES 3 PREMIERS ELEMENTS SONT SIGNIFICATIFS
20 012345 ADD    LE PREMIER ELEMENT = LE CODE OPERATOIRE
21 012345 SUB    LE DEUXIEME REPREND LES MODES D'ADRESSAGES POSSIBLES
22 012345 MUL     SUIVANT LE CODE OPERATOIRE
23 012345 DIV    LE TROISIEME ELEMENT = REPRESENTATION EN CLAIR DU
24 0      NEG     CODE OPERATOIRE
25 0      ABS
26 012345 DBZ    POUR MAJ LA TABLE IL FAUT RESPECTER LA REGLE
27 012345 DBP    - AVOIR LES 3 ELEMENTS
28 012345 DBN    - AVOIR ENTRE CHAQUE ELEMENT UN SEPARATEUR (= BLANC)
30 0      CPA
40 012345 B
41 012345 B=
42 012345 B<
43 012345 B>
44 012345 B<>
45 012345 B<=
46 012345 B>=
47 012345 BOV
48 012345 BER
49 012345 CAL
70 1235   RDN
71 1235   RDC
80 01235  PRN
81 01235  PRC
90 0      NOP
91 0      HLT
92 0      DOV
93 0      EOV
```

LES PROGRAMMES.

**** LES CONSTANTES ****

-- fcarcst --

```
lmaxlig = 80;
lmaxcha = 81;
fdl = '@';
fdf = '^';
blan = ' ';
moins = '-';
plus = ' ';
zero = '0';
```

-- fmescst --

```
lmaxcmd = 2;
lmax1erparam = 10;
lmax2emparam = 4;
lmax3emparam = 4;
maxdonnee = 4;          { nombre maximun de donnees }
                        { dans un message }
pos_f_superv = 3;       { position de la derniere commande }
                        { superviseur dans le tableau des  }
                        { des commandes }
pos_d_edit  =  4;       { position de la  premiere commande }
                        { editeur compilateur dans le tableau }
                        { des commandes }
pos_f_edit = 17;        { position de la derniere commande }
                        { editeur compilateur dans le tableau }
                        { des commandes }
pos_d_interp = 18;      { position de la premiere commande }
                        { interpreteur pupitreur dans le  tableau}
                        { des commandes }
pos_f_interp = 42;      { position de la derniere commande }
                        { interpreteur pupitreur dans le  tableau}
                        { des commandes }
```

```
{*** ecran ***}
longecran =      80;
largecran =      23;

{*** memoire centrale ***}
ligmot1 =       3;
colmot1 =       5;
hautmot =      1;
lmot =       6;
distmot =      1;
nmotligmem =       10;
nmotcolmem =       10;

{*** lecteur de cartes ***}
ligcart1 =      14;
colcart1 =       3;
hautcart =       1;
ncart =       5;
lcart =      20;

{*** compteur ordinal ***}
ligco =      15;
colco =      27;
lco =       2;

{***  registre instruction ***}
ligri =      15;
colri =      33;

{*** code operatoire ***}
ligcop =       15;
colcop =       44;
lcop =       3;

{*** registre extension ***}
ligre =      18;
colre =      27;

{*** accumulateur ***}
liga =      18;
cola =      36;
```

```
{*** indicateur logique ***}
ligil =      18;
colil =      45;
lil =       2;

{*** zone d'affichage ***}
ligza =      22;
colza =      27;
lza =      20;
nlza =       2;

{*** imprimante ***}
ligimp1 =      14;
colimp1 =      50;
hautlig =       1;
nlig =       5;
llig =      24;
```

*** LES TYPES ***

-- fcartype --

```
lmaxlig = 80;
lmaxcha = 81;
fdl = '@';
fdf = '^';
blan = ' ';
moins = '-';
plus = ' ';
zero = '0';
```

```
bon_mauvais = (bon,mauvais);
type1_2 = (TEC,TM,TQ,TL,TR,TI,TD,TP,TMOINS,TS,TT,TV,TE1,TE2,
     TE3,TE4,TE5,TFM,TX,TFI,TFA,TFC,TFE,TFL,TZM,TML,TVI,TEI,TEP
     TMS,TLN,TLS,TCR,TCL,TCPLUS,TCMOINS,TCUN,TM1,TM2,TM3,
     TM4,TM5,TNIL);
nomb_car = (nombre,caractere);
nom_expediteur = (superviseur,interp_pupit,edit_compil);

message_cmd = RECORD
              etat : bon_mauvais;
              cmd_type : type1_2;
              param1 : PACKED ARRAY [1..1max1erparam] OF CHAR;
              param1_long : 1..1max1erparam;
              param1_type : nomb_car;
              param2 : PACKED ARRAY [1..1max2emparam] OF CHAR;
              param2_long : 1..1max2emparam;
              param2_type : nomb_car;
              param3 : PACKED ARRAY [1..1max3emparam] OF CHAR;
              param3_long : 1..1max2emparam;
              param3_type : nomb_car;
             END;
```

```
                                -- fgrtype --

        posint = 1..maxint;
        erasetype = (eol,bol,line,eos,bos,screen);
        nolig_ecran = 1..largecran;
        nocol_ecran = 1..longecran;
```

```
                                -- fordtype --

mot = PACKED ARRAY[1..lmot] OF CHAR;
carte = PACKED ARRAY [1..lcart] OF CHAR;
ligne = PACKED ARRAY [1..llig] OF CHAR;
ordinat_type = RECORD
        memoire : ARRAY [1..nmotligmem,1..nmotcolmem] OF mot;
        cartes  : ARRAY [1..25] OF carte;
        no_carte: 1..25;
        col_cart: 1..lcart;
        vi      : 0..99;
        co      : PACKED ARRAY [1..lco] OF CHAR;
        ri      : mot;
        cop     : PACKED ARRAY [1..lcop] OF CHAR;
        re      : mot;
        re_ov   : BOOLEAN;
        ra      : mot;
        li      : PACKED ARRAY [1..lil] OF CHAR;
        lignes  : ARRAY [1..nlig] OF ligne;
        no_lig  : 1..nlig;
        col_lig : 1..llig;
              END;
```

PROGRAMME DU NIVEAU 0

```
#
  {*-----------------------------------*}
  {*                                   *}
  {* PROGRAMME SUPERVISEUR             *}
  {*                                   *}
  {*-----------------------------------*}

PROGRAM AP_SUP(INPUT,OUTPUT,
               ordpeddes,
               fmesout,
               ftabcomd,fcopma);

{-------------------------------------------------------------------------
*  ENTREE  :     - une commande superviseur;                              *
*                                                                         *
* SORTIE :       - appel du module lecture commande superviseur           *
*                - appel du module selection de modules                   *
*                - sortie du systeme;                                     *
*                                                                         *
* FONCTION :     - si l'utilisateur vient d'entrer dans le                *
*                systeme alors appel du module lecture                    *
*                commande superviseur,  aller a l'etape suivante          *
*                - envoit de la commande superviseur vers le              *
*                module selection de modules ou sortie du systeme         *
-------------------------------------------------------------------------}

CONST

# include "fmescst"

TYPE

# include "fmestype"


VAR
      message : message_cmd;    {* message utilisateur *}

      ordpeddes,          {* dessin de l'ordinateur *}
      fmesout,            {* fichier de messages de sortie *}
      ftabcomd,fcopma :       {* table de commandes    *}
            TEXT;


 {**********************************}
 { effacement du contenu de l'ecran }

 PROCEDURE FC_NETECRAN; EXTERN;

 {******************************************}
 { enregistrement et controle d'une commande }
 {      superviseur                         }

 PROCEDURE PH_ECCSUP(VAR message : message_cmd;
```

```
      VAR fentree,fsortie,fmessage,fcommande : TEXT);
EXTERN;

{**********************************}
{ selection d'une tache du systeme }

PROCEDURE PH_SEL_APP(VAR message : message_cmd;
          VAR fentree,fsortie,ordpeddessin,
         fmessage,fcommande,fcopmachine : TEXT);
EXTERN;

{*-------------------------------------------------------*}

  PROCEDURE INITIALISATION;
     BEGIN
        FC_NETECRAN;
        REPEAT
        PH_ECCSUP(message,INPUT,OUTPUT,fmesout,ftabcomd)
        UNTIL message.cmd_type <> TNIL
     END;

  PROCEDURE TRAITEMENT;

     BEGIN
PH_SEL_APP(message,INPUT,OUTPUT,ordpeddes,fmesout,ftabcomd,fcopm
     END;

  PROCEDURE CLOTURE;

     BEGIN
        FC_NETECRAN;
     END;

{*-------------------------------------------------------*}

BEGIN
   INITIALISATION;
   WHILE message.cmd_type <> TQ DO TRAITEMENT;
   CLOTURE END.
```

PROGRAMMES DU NIVEAU 1

```
#
 {*-----------------------------------------------*}
 {*  ENREGISTREMENT ET CONTROLE COMMANDE          *}
 {*              SUPERVISEUR                      *}
 {*-----------------------------------------------*}

 {-------------------------------------------------------------------
 * ENTREE :      - un fichier contenant le message a lire      *
 *               - un fichier reprenant la liste des commandes*
 *                                                             *
 * SORTIE :      - un message contenant une commande           *
 *               superviseur valide                            *
 *                                                             *
 * FONCTION :    - lecture d'un message                        *
 *               - verification de la validite du message      *
 *               - envoit message erreur si message invalide et *
 *                 recommencement de la fonction;              *
 -------------------------------------------------------------------}

 CONST
 # include "fmescst"
 # include "fcarcst"

 TYPE
 # include "fmestype"
 # include "fcartype"

 {*************************************************}
 { analyse syntaxique d'une commande utilisateur }

 PROCEDURE FC_MESSYNTAX(expediteur : nom_expediteur;
       VAR commande : chaincaract;
       VAR message : message_cmd;
      VAR table_cmd : TEXT);
 EXTERN;


 {*************************************************}
 { analyse semantique d'une commande utilisateur  }

 PROCEDURE FC_MESSEMANT(expediteur : nom_expediteur;
       VAR message : message_cmd);EXTERN;

 {*****************************************}
 { lecture d'une chaine de caracteres dans }
 {  un fichier          }

 PROCEDURE FC_LECTLIGNE(VAR zonlect : chaincaract;
       long_lire : lonlig;
       VAR fentree : TEXT);EXTERN;


 {*****************************************}
 { ecriture d'une chaine de caracteres dans }
 {  un fichier.          }
```

```
PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
          longimp : lonlig;
          VAR fsortie : TEXT);EXTERN;



{*****************************************}
{ lecture d'un symbole dans une chaine de }
{ caracteres  }

PROCEDURE FC_LECTSYMB(VAR chaine,symbole : chaincaract;
       VAR longsymb,pos_d_lect : loncha);
EXTERN;

{*-------------------------------------------------------------*}

PROCEDURE PH_ECCSUP(VAR message : message_cmd;
    VAR INPUT,OUTPUT,fmesout,ftabcomd : TEXT);

VAR
  zon_mes,   {* zone reception de message utilisateur *}
  tat,     {* zone contenant le message attente *}
  tere  :  {* zone contenant le message erreur  *}
       chaincaract;

  PROCEDURE INITIALISATION;

  VAR zonlec : chaincaract;
      pos_d_lect : loncha;
      longsymb : loncha;

  BEGIN
    RESET(fmesout);
    FC_LECTLIGNE(zonlec,lmaxlig,fmesout);
    pos_d_lect := 1;
    FC_LECTSYMB(zonlec,tat,longsymb,pos_d_lect);
    FC_LECTLIGNE(zonlec,lmaxlig,fmesout);
    pos_d_lect := 1;
    FC_LECTSYMB(zonlec,tere,longsymb,pos_d_lect)
  END;


  PROCEDURE FC_ECSUP;  {* enregistrement du message *}

  BEGIN
    FC_ECRLIGNE(tat,lmaxlig,OUTPUT);
    FC_LECTLIGNE(zon_mes,lmaxlig,INPUT)
  END;

  PROCEDURE FC_CCSUP;  {* controle du message *}

  BEGIN
    FC_MESSYNTAX(superviseur,zon_mes,message,ftabcomd);
    IF message.etat = bon THEN
```

```
        FC_MESSEMANT(superviseur,message)
    END;

    PROCEDURE FC_ERRSUP;   {* traitement message erronnee *}

    BEGIN
      FC_ECRLIGNE(tere,lmaxlig,OUTPUT)
    END;

  {*-------------------------------------------------------*}

BEGIN
    INITIALISATION;
    FC_ECSUP;
    FC_CCSUP;

    WHILE message.etat = mauvais DO

    BEGIN
      FC_ERRSUP;
      FC_ECSUP;
      FC_CCSUP
    END
END;
```

```
#
 {*-----------------------------------------*}
 {* SELECTION APPLICATION                  *}
 {*-----------------------------------------*}

 {* ENTREE      : une commande superviseur      *}
 {* SORTIE      : un appel a une procedure      *}
 {* FONCTION    : selectionne la procedure a    *}
 {*               appeler selon la valeur de la *}
 {*               commande recue                *}

 CONST
 # include "fmescst"

 TYPE
 # include "fmestype"

 {*******************************}
 { programme editeur_assembleur }

 PROCEDURE AP_EC(VAR message : message_cmd;
   VAR fentree,fsortie,fmessage,fcommande: TEXT);
 EXTERN;

 {*********************}
 { programme moniteur }

 PROCEDURE AP_IP(VAR message : message_cmd;
     VAR       fentree,fsortie,ordpeddes,fmesout,ftabcomd,fcopma:
TEXT);
 EXTERN;

 {*-------------------------------------------------------------*}

PROCEDURE PH_SEL_APP(VAR message : message_cmd;
    VAR INP,OUT,ordpeddes,fmesout,ftabcomd,fcopma : TEXT);

BEGIN
  CASE message.cmd_type OF

  TEC   : AP_EC(message,INP,OUT,fmesout,ftabcomd);
  TM    : AP_IP(message,INP,OUT,ordpeddes,fmesout,ftabcomd,fcopma)

  END END;

{*-------------------------------------------------------------*}
```

PROGRAMMES DU NIVEAU 2

```
#
 {*-------------------------------------------------*}
 {*      EDITEUR          COMPILATEUR                *}
 {*-------------------------------------------------*}


 CONST
 # include "fmescst"

 TYPE

 # include "fmestype"


 {***********************************************}
 { lecture d'une commande editeur_compilateur }

 PROCEDURE PH_ECCEC(VAR message : message_cmd;
           VAR fentree,fsortie,fmessage,fcommande : TEXT);EXTERN;

 {*************************************************}
 { execution d'une commande du editeur_compilateur }

  PROCEDURE PH_EXCEC(VAR message : message_cmd;
           VAR fentree,fsortie : TEXT);EXTERN;

 (*-------------------------------------------------*)


PROCEDURE AP_EC(VAR message : message_cmd;
        VAR INPUT,OUTPUT,fmesout,fcomd: TEXT);

  PROCEDURE INIT_EC;
  BEGIN
        { 1ere lecture }
        PH_ECCEC(message,INPUT,OUTPUT,fmesout,fcomd)
  END;

  PROCEDURE TRT_EC;

  BEGIN
        { execution d'une commande editeur compilateur }

        PH_EXCEC(message,INPUT,OUTPUT);

        { lecture commande suivante }

        PH_ECCEC(message,INPUT,OUTPUT,fmesout,fcomd)
  END;

  PROCEDURE CLOT_EC;

  BEGIN
```

```
  END;
(*-------------------------------------------------------------------*)

BEGIN
  INIT_EC;
  WHILE NOT (message.cmd_type IN [TEC,TM,TQ]) DO TRT_EC;
  CLOT_EC
END;
```

```
#
 {-----------------------------------------}
 { APPLICATION INTERPRETEUR PUPITREUR }
 {-----------------------------------------}


 CONST
 # include "fgrrcpt"
 # include "fmescst"
 # include "fcarcst"

 TYPE

 # include "fmestype"
 # include "fordtype"



 {*********************************}
 { effacement du contenu de l'ecran }

 PROCEDURE FC_NETECRAN; EXTERN;

 {*************************************************}
 { affichage du dessin de l'ordinateur pedagogique }

 PROCEDURE PH_AFFGRAPH(VAR ordpeddessin, fsortie : TEXT);
 EXTERN;

 {*************************************}
 { execution d'une commande du moniteur }

 PROCEDURE PH_EXCIP(VAR message : message_cmd;
        VAR ordinateur : ordinat_type;
        VAR fentree, fsortie, fmessages, fcopmachine : TEXT); EXTERN;

 {*********************************}
 { lecture d'une commande moniteur }

 PROCEDURE PH_ECCIP(VAR message : message_cmd;
        VAR fentree, fsortie, fmessage, fcommande : TEXT); EXTERN;

 (*-----------------------------------------------------------*)

PROCEDURE AP_IP(VAR message : message_cmd;
    VAR INPUT, OUTPUT, ordpeddes, fmesout, fcomd, fcopma: TEXT);

VAR
  ordinateur : ordinat_type;


  PROCEDURE INIT_IP;

  VAR
```

```
    i, j, l : INTEGER;

  BEGIN
    { initialisation de l'ordinateur }
    ordinateur.re_ov := TRUE;
    ordinateur.no_carte := 0;
    ordinateur.cartes[1,1] := fdl;
    ordinateur.vi := 0;
    FOR i := 1 TO nmotligmem DO
      FOR j := 1 TO nmotcolmem DO
        BEGIN
          ordinateur.memoire[i, j, 1] := ' ';
          FOR l := 2 TO lmot
            DO ordinateur.memoire[i, j, l] := '0'
        END;
    FOR i := 1 TO lco DO ordinateur.co[i] := '0';

    { affichage graphique }
    PH_AFFGRAPH(ordpeddes,OUTPUT);

    { 1ere lecture }

    PH_ECCIP(message, INPUT, OUTPUT, fmesout, fcomd)
  END;

  PROCEDURE TRT_IP;

  BEGIN
    { execution d'une commande interpreteur pupitreur }
    PH_EXCIP(message, ordinateur, INPUT, OUTPUT, fmesout, fcopma);

    { lecture commande suivante }

    PH_ECCIP(message, INPUT, OUTPUT, fmesout, fcomd)
  END;

  PROCEDURE CLOT_IP;

  BEGIN
    FC_NETECRAN
  END;

(*-------------------------------------------------------------*)

BEGIN
  INIT_IP;
  WHILE NOT (message.cmd_type IN [TEC, TM, TQ]) DO TRT_IP;
  CLOT_IP
END;

 (*-------------------------------------------------------------*)
```

PROGRAMMES DU NIVEAU 3

```
#

{*-------------------------------------------------------------*}
{**** PHASE ENREGISTREMENT ET CONTROLE  COMMANDE ***}
{**** EDITEUR    & COMPILATEUR                          ***}
{*-------------------------------------------------------------*}

{* Cette phase :
  1) lit les donnees introduites par l'utilisateur

  2) determine la validite des donnees introduites;

  3) si la donnee est invalide,

     elle affiche un message d'erreur et
     recommence son traitement en 1)        *}

CONST

# include "fcarcst"
# include "fmescst"

TYPE

# include "fmestype"
# include "fcartype"



{***********************************************}
{ analyse syntaxique d'une commande utilisateur }

PROCEDURE FC_MESSYNTAX(expediteur : nom_expediteur;
      VAR commande : chaincaract;
      VAR message : message_cmd;
      VAR table_cmd : TEXT);
EXTERN;


PROCEDURE FC_MESSEMANT(expediteur : nom_expediteur;
      VAR message : message_cmd);EXTERN;

{*****************************************}
{ lecture d'une chaine de caracteres dans }
{  un fichier          }

PROCEDURE FC_LECTLIGNE(VAR zonlect : chaincaract;
      long_lire : lonlig;
      VAR fentree : TEXT);EXTERN;


{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{  un fichier.          }
```

```
PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
          longimp : lonlig;
          VAR fsortie : TEXT);EXTERN;



{*****************************************}
{ lecture d'un symbole dans une chaine de }
{ caracteres  }

PROCEDURE FC_LECTSYMB(VAR chaine,symbole : chaincaract;
      VAR longsymb,pos_d_lect : loncha);
EXTERN;


{* DESCRIPTION DU PROGRAMME *}

PROCEDURE PH_ECCEC(
       VAR message:     {* zone contenant le resulat *}
       message_cmd;  {* de l'enregistrement       *}
        VAR INP,   {* terminal *}
            OUT,   {* terminal *}
       fmesip,     {* fichier message de sortie *}
       ftabsymb:   {* fichier contenant la table *}
           {* des symboles *}
       TEXT);

VAR
  zon_mes,      {* zone de reception du message utilisateur *}
  tat,     {* zone de reception du message "attente"   *}
  tere      {* zone de reception du message "erreur donnees *}
    : chaincaract;

{*-------------------------------------------------------------------------


  PROCEDURE INITERR;  {* stockage des messages d'erreur *}

  VAR
    zonlec : chaincaract;
    pos_d_lect,longsymb : loncha;
  BEGIN
    reset(fmesip);

    FC_LECTLIGNE(zonlec,lmaxlig,fmesip);
    pos_d_lect := 1;
    FC_LECTSYMB(zonlec,tat,longsymb,pos_d_lect);
    FC_LECTLIGNE(zonlec,lmaxlig,fmesip);
    pos_d_lect := 1;
    FC_LECTSYMB(zonlec,tere,longsymb,pos_d_lect)
  END;

  PROCEDURE FC_ECEC;   {* enregistrement commande *}
                       {* editeur compilateur     *}
```

```
  BEGIN

    {* affichage du message "attente" de l'editeur compil. *}

    FC_ECRLIGNE(tat,lmaxlig,OUT);
       REPEAT

    {* lecture du message introduit par l'utilisateur *}

    FC_LECTLIGNE(zon_mes,lmaxcha,INP)

    UNTIL zon_mes[1] <> fdl

  END;


  PROCEDURE FC_CCEC;   {* controle commande editeur      *}
                       {* compilateur                   *}

  BEGIN

    {* analyse syntaxique message introduit *}

    FC_MESSYNTAX(edit_compil,zon_mes,message,ftabsymb);

    {* analyse semantique message introduit *}

    IF message.etat = bon THEN
        FC_MESSEMANT(edit_compil,message)
  END;

  PROCEDURE FC_ERREC;  {* affichage message erreur *}

  BEGIN

    {* ecriture message erreur *}

    FC_ECRLIGNE(tere,lmaxlig,OUT)
  END;

{*-------------------------------------------------------------*}

PROCEDURE INIT_ECCEC;

  BEGIN
    INITERR
  END;

PROCEDURE TRT_ECCEC;  {* traitement *}

  BEGIN

    FC_ECEC;
    FC_CCEC;
```

```
    WHILE message.etat = mauvais DO
      BEGIN
        FC_ERREC;
        FC_ECEC;
        FC_CCEC
      END
  END;

{*-------------------------------------------------------*}

BEGIN
    INIT_ECCEC;
    TRT_ECCEC
END;
```

```
#
CONST
# include "fmescst"

TYPE
# include "fmestype"
{*************************************************}
{ execution d'une commande du editeur_compilateur }

PROCEDURE PH_EXCEC(VAR message : message_cmd;
        VAR INPUT,OUTPUT : TEXT);

BEGIN
  WRITELN(OUTPUT);
  WRITE(OUTPUT,'sorry les commandes de l''editeur');
  WRITELN(OUTPUT);
  WRITE(OUTPUT,'et de l''assembleur ne sont pas');
  WRITELN(OUTPUT);
  WRITE(OUTPUT,' encore implementees ');
  WRITELN(OUTPUT)
END;
```

```
#

(*-------------------------------------------------------------*)
(**** PHASE ENREGISTREMENT ET CONTROLE  COMMANDE ***)
(**** INTERPRETEUR & PUPITREUR                  ***)
(*-------------------------------------------------------------*)

(* Cette phase :
  1) lit les donnees introduites par l'utilisateur
     dans la zone d'affichage de l'ordinateur
     pedagogique;

  2) determine la validite des donnees introduites;

  3) si la donnee est invalide,

     elle affiche un message d'erreur et
     recommence son traitement en 1)       *)

CONST

# include "fgrrcpt"
# include "fcarcst"
# include "fmescst"

TYPE

# include "fmestype"
# include "fcartype"
# include "fgrtype"


{*************************************************}
{ positionnement du curseur sur l'ecran     }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
      col_curs : nocol_ecran);
EXTERN;

{*************************************************}
{ analyse syntaxique d'une commande utilisateur }

PROCEDURE FC_MESSYNTAX(expediteur : nom_expediteur;
      VAR commande : chaincaract;
      VAR message : message_cmd;
      VAR table_cmd : TEXT);
EXTERN;


PROCEDURE FC_MESSEMANT(expediteur : nom_expediteur;
      VAR message : message_cmd);EXTERN;
{*************************************}
{ lecture d'une chaine de caracteres dans }
{  un fichier        }
```

```
PROCEDURE FC_LECTLIGNE(VAR zonlect : chaincaract;
      long_lire : lonlig;
      VAR fentree : TEXT);EXTERN;


{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{  un fichier.          }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
         longimp : lonlig;
         VAR fsortie : TEXT);EXTERN;



{*****************************************}
{ lecture d'un symbole dans une chaine de }
{ caracteres  }

PROCEDURE FC_LECTSYMB(VAR chaine,symbole : chaincaract;
      VAR longsymb,pos_d_lect : loncha);
EXTERN;


(* DESCRIPTION DU PROGRAMME *)

PROCEDURE PH_ECCIP(
      VAR message:    (* zone contenant le resulat *)
      message_cmd;  (* de l'enregistrement       *)
       VAR INP,   (* terminal *)
           OUT,   (* terminal *)
      fmesip,     (* fichier message de sortie *)
      ftabsymb:   (* fichier contenant la table *)
                  (* des symboles *)
      TEXT);
VAR
  zon_mes,      (* zone de reception du message utilisateur *)
  tat,     (* zone de reception du message "attente"   *)
  tere,      (* zone de reception du message "erreur donnees *)
  lig_trav
          : chaincaract;


(*-------------------------------------------------------------*)

  PROCEDURE INITZA;  (* initialisation zone d'affichage *)

  BEGIN
    FC_POSITCURS(ligza,colza);
    FC_ECRLIGNE(lig_trav,lza,OUT)
  END;

  PROCEDURE INITZERR;  (* initialisation de la zone d'erreur *)
```

```
BEGIN

  FC_POSITCURS(ligza - nlza + 1,colza);
  FC_ECRLIGNE(lig_trav,lza,OUT)
END;

PROCEDURE INITERR;  (* stockage des messages d'erreur *)

VAR
  zonlec : chaincaract;
  pos_d_lect,longsymb : loncha;
BEGIN
  reset(fmesip);

  FC_LECTLIGNE(zonlec,lmaxlig,fmesip);
  pos_d_lect := 1;
  FC_LECTSYMB(zonlec,tat,longsymb,pos_d_lect);
  FC_LECTLIGNE(zonlec,lmaxlig,fmesip);
  pos_d_lect := 1;
  FC_LECTSYMB(zonlec,tere,longsymb,pos_d_lect)
END;

PROCEDURE FC_ECIP;  (* enregistrement commande *)
      (* interpreteur pupitreur  *)

BEGIN

  (* initialisation zone d'affichage *)

  INITZA;

  (* affichage du message "attente" de l'interp. pupitreur *)

  FC_POSITCURS(ligza,colza);
  FC_ECRLIGNE(tat,lza,OUT);
     REPEAT

  (* lecture du message introduit par l'utilisateur *)

  FC_POSITCURS(ligza,colza + 2);
  FC_LECTLIGNE(zon_mes,lza - 3,INP)

  UNTIL zon_mes[1] <> fdl

END;


PROCEDURE FC_CCIP;  (* controle commande interpreteur *)
      (* pupitreur          *)

BEGIN

  (* analyse syntaxique message introduit *)

  FC_MESSYNTAX(interp_pupit,zon_mes,message,ftabsymb);
```

```
    (* analyse semantique message introduit *)

    IF message.etat = bon THEN DO
        FC_MESSEMANT(interp_pupit,message)

  END;

  PROCEDURE FC_ERRIP;   (* affichage message erreur *)

  BEGIN


    (* ecriture message erreur *)

    FC_POSITCURS(ligza - nlza + 1,colza);
    FC_ECRLIGNE(tere,lza,OUT)
  END;

(*-------------------------------------------------------------*)
PROCEDURE INIT_ECCIP;

  VAR indza : loncha;

  BEGIN
    INITERR;
    FOR indza := 1 TO lza DO lig_trav[indza] := blan;
    INITZA;
    INITZERR
  END;

PROCEDURE TRT_ECCIP;   (* traitement *)

  BEGIN

    FC_ECIP;
    FC_CCIP;

    WHILE message.etat = mauvais DO

      BEGIN
        FC_ERRIP;
        FC_ECIP;
        FC_CCIP
      END
  END;

(*-------------------------------------------------------------*)

BEGIN
    INIT_ECCIP;
    TRT_ECCIP
END;

(*-------------------------------------------------------------*)
```

```
#
{*-----------------------------------------------*}
{* EXECUTION COMMANDE MONITEUR                   *}
{*-----------------------------------------------*}

CONST
# include "fmescst"
# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fmestype"
# include "fcartype"
# include "fordtype"


{*-------------------------------------------
* ENTREE : un message contenant la      *
*          la commande a executee       *
*                                       *
* SORTIE : apppel du module concernee   *
*                                       *
* FONCTION : ce module realise l'aiguil-*
*            lage des appels suivant le *
*            type de commande recue     *
*-------------------------------------------*}


{*****************************************}
{ sauvetage du contenu de la memoire de }
{ l'ordinateur pedagogique }

PROCEDURE FC_SAUVMEM(VAR message : message_cmd;
          VAR ordinateur : ordinat_type;
          VAR fsortie,fmessage : TEXT);
EXTERN;

{*************************************}
{garnissage et affichage de la zone }
{ memoire de l'ordinateur pedagogique }


PROCEDURE FC_GMEMCENT(VAR ordinateur : ordinat_type;
       VAR fentree,fsortie,fmessages : TEXT);
EXTERN;


{*****************************************}
{ garnissage et affichage de la zone     }
{ registre A de l'ordinateur pedagogique }

PROCEDURE FC_GREGA(VAR message : message_cmd;
        VAR ordinateur : ordinat_type;
        VAR fsortie : TEXT);
EXTERN;
```

```
{*****************************************}
{ garnissage et affichage de la zone      }
{ registre I de l'ordinateur pedagogique }

PROCEDURE FC_GREGI(VAR message : message_cmd;
       VAR ordinateur : ordinat_type;
       VAR fsortie : TEXT);
EXTERN;


{*****************************************}
{ garnissage et affichage de la zone      }
{ registre C de l'ordinateur pedagogique }

PROCEDURE FC_GREGC(VAR message : message_cmd;
       VAR ordinateur : ordinat_type;
       VAR fsortie : TEXT);
EXTERN;



{*****************************************}
{ garnissage et affichage de la zone      }
{ registre E de l'ordinateur pedagogique }

PROCEDURE FC_GREGE(VAR message : message_cmd;
       VAR ordinateur : ordinat_type;
       VAR fsortie : TEXT);
EXTERN;

{*****************************************}
{   procedure information       }

PROCEDURE FC_HELP(VAR fsortie : TEXT);
EXTERN;

{*****************************************}
{ initialisation des zones de l'ordinateur }
{  pedagogique          }

PROCEDURE FC_INITMACH(VAR ordinateur : ordinat_type;
      VAR fsortie : TEXT);EXTERN;

{*************************************************************}
{* chargement de la memoire de l'ordinateur pedagogique *}

PROCEDURE FC_CHMEMCENT(
      VAR message : message_cmd;
      VAR ordinateur : ordinat_type;
      VAR fmesout,OUTPUT : TEXT);
EXTERN;

{*****************************************}
```

```
{ garnissage et affichage de la zone     }
{ registre L de l'ordinateur pedagogique }

PROCEDURE FC_GREGL(VAR message : message_cmd;
       VAR ordinateur : ordinat_type;
       VAR fsortie : TEXT);
EXTERN;



{*************************************}
{ execution d'une instruction machine }
{ de l'ordinateur pedagogique         }

PROCEDURE FC_EXINST(VAR ordinateur : ordinat_type;
        VAR ok : BOOLEAN;
       VAR fsortie,fmessage,fcopmachine : TEXT
       );EXTERN;

{*****************************************************}
{ reglage du temps delai entre chaque instruction }
{ pendant execution d'un programme  }

PROCEDURE FC_RARINST(VAR message : message_cmd;
        VAR ordinateur : ordinat_type);
EXTERN;

{*********************************}
{ execution d'un programme machine }

PROCEDURE FC_EXPRG(VAR message : message_cmd;
       VAR ordinateur : ordinat_type;
       VAR fsortie,fmessage,fcopma : TEXT
       );EXTERN;

{*****************************************}
{ chargement du lecteur de cartes de     }
{ l'ordinateur pedagogique }

PROCEDURE FC_CHLECT(VAR message : message_cmd;
                    VAR ordinateur : ordinat_type;
                    VAR fmesout,fsortie : TEXT);
EXTERN;


{*********************************************}
{ rend courant la premiere carte du paquet   }

PROCEDURE FC_CARTUN(VAR ordinateur : ordinat_type;
                     VAR fmesout,OUTPUT : TEXT);
EXTERN;

{*********************************}
{ affichage de la carte qui suit   }
{ la carte courante               }
```

```
PROCEDURE FC_CARTSUIV(VAR ordinateur : ordinat_type;
                      VAR fmesout,OUTPUT : TEXT);
EXTERN;

{*************************************}
{ affichage de la carte qui precede   }
{ la carte courante                   }

PROCEDURE FC_CARTPREC(VAR ordinateur : ordinat_type;
                      VAR fmesout,OUTPUT : TEXT);
EXTERN;


{*********************************}
{ avancement d'une ligne de l'imprimante }

PROCEDURE FC_AVANCPAP (VAR ordinateur : ordinat_type;
                       VAR OUTPUT : TEXT);EXTERN;

{*************************************************}
{ sauvetage du contenu de l'imprimante }

PROCEDURE FC_SAUVLIST(VAR message : message_cmd;
                      VAR ordinateur : ordinat_type;
                       VAR fmesout,OUTPUT : TEXT);
EXTERN;

{*************************************}
{ initialisation du lecteur de cartes }

PROCEDURE FC_CARTINIT( VAR ordinateur : ordinat_type;
                       VAR OUTPUT : TEXT);
EXTERN;
{*-----------------------------------------------*}
PROCEDURE PH_EXCIP(VAR message : message_cmd;
        VAR ordinateur : ordinat_type;
        VAR INPUT,OUTPUT,fmesout,fcopma: TEXT);


VAR ok : BOOLEAN;

BEGIN
  CASE message.cmd_type OF


  TZM  : FC_INITMACH(ordinateur,OUTPUT);
  TFM  : BEGIN
         FC_GREGC(message,ordinateur,OUTPUT);
         FC_GMEMCENT(ordinateur,INPUT,OUTPUT,fmesout)
       END;
  TFA  : FC_GREGA(message,ordinateur,OUTPUT);
  TFI  : FC_GREGI(message,ordinateur,OUTPUT);
  TFC  : FC_GREGC(message,ordinateur,OUTPUT);
```

```
    TFE   : FC_GREGE(message,ordinateur,OUTPUT);
    TFL   : FC_GREGL(message,ordinateur,OUTPUT);
    TML   : FC_CHMEMCENT(message,ordinateur,fmesout,OUTPUT);

    {* EXECUTION*}
    {*-----------*}

    TVI   : FC_RARINST(message,ordinateur);
    TEI   : FC_EXINST(ordinateur,ok,OUTPUT,fmesout,fcopma);
    TEP   : FC_EXPRG(message,ordinateur,OUTPUT,fmesout,fcopma);

    {* SAUVETAGE MEMOIRE*}
    {*-------------------*}

    TMS   : FC_SAUVMEM(message,ordinateur,OUTPUT,fmesout);

    {* INFORMATION *}
    {*-------------*}

    TX    : FC_HELP(OUTPUT);

    {* GESTION IMPRIMANTE*}
    {*------------------*}

    TLN   : FC_AVANCPAP(ordinateur,OUTPUT);
    TLS   : FC_SAUVLIST(message,ordinateur,fmesout,OUTPUT);

    {* GESTION LECTEUR DE CARTES*}
    {*-------------------------*}

    TCUN   : FC_CARTUN(ordinateur,fmesout,OUTPUT);
    TCL    : FC_CHLECT(message,ordinateur,fmesout,OUTPUT);
    TCPLUS   : FC_CARTSUIV(ordinateur,fmesout,OUTPUT);
    TCMOINS   : FC_CARTPREC(ordinateur,fmesout,OUTPUT);
    TCR    :  FC_CARTINIT(ordinateur,OUTPUT);


    {* groupe nouveaux *}
    {*-----------------*}

    TM1   :;
    TM2   :;
    TM3   :;
    TM4   :;
    TM5   :;
    END

END;

{*-----------------------------------------*}
```

```
#
{----------------------------------------}
{ PHASE gestion graphique de l'ecran }
{----------------------------------------}

CONST

# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fgrtype"
# include "fcartype"

{-------------------------------------------------------------
* ENTREE  : le fichier contenant le schema garphique;*
*                                                     *
* SORTIE  : affichage graphique de l'ordinateur      *
*             pedagogique                             *
*                                                     *
* FONCTION : lecture et ecriture a l'ecran du contenu*
*             du fichier contenant le schema          *
*                  graphique                          *
*-------------------------------------------------------}


{*************************************************}
{ positionnement du curseur sur l'ecran          }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{*********************************}
{ effacement du contenu de l'ecran }

PROCEDURE FC_NETECRAN; EXTERN;


{*****************************************}
{ placer le terminal en mode graphique }

PROCEDURE MODE_GRAPHE; EXTERN;


{*****************************************}
{ placer le terminal en mode caractere }

PROCEDURE MODE_ASCII; EXTERN;


{*****************************************}
{ lecture d'une chaine de caracteres dans }
{      un fichier                         }
```

```
PROCEDURE FC_LECTLIGNE(VAR zonlect : chaincaract;
                       long_lire : lonlig;
                       VAR fentree : TEXT);EXTERN;


{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                        }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;

(*-------------------------------------------------------------*)

PROCEDURE PH_AFFGRAPH(VAR ordpeddes,OUTPUT : TEXT);


  PROCEDURE FC_AFFORDPED;
              { affichage de l'ordinateur pedagogique }

  VAR
    lig_dessin : chaincaract;
    i : nolig_ecran;

  BEGIN
    RESET(ordpeddes);

    FOR i := 1 TO largecran DO
    BEGIN
      FC_POSITCURS(i,1);
      FC_LECTLIGNE(lig_dessin,lmaxlig,ordpeddes);
      MODE_GRAPHE;
      FC_ECRLIGNE(lig_dessin,lmaxlig,OUTPUT)
    END;
    MODE_ASCII
  END;


(*-------------------------------------------------------------*)

BEGIN
  FC_NETECRAN;
  FC_AFFORDPED
END;

(*-------------------------------------------------------------*)
```

PROGRAMMES DU NIVEAU 4

```
#
(*-------------------------------------------*)
(*  GARNISSAGE MEMOIRE                       *)
(*-------------------------------------------*)

CONST
# include "fgrrcpt"
# include "fmescst"
# include "fcarcst"

TYPE
# include "fordtype"
# include "fmestype"
# include "fcartype"
# include "fgrtype"

{*-----------------------------------------------
* ENTREE :                                      *
*                                               *
* SORTIE : garnissage de la memoire centrale*
*                                               *
* FONCTION :    lecture, controle et            *
*               stockage des mots memoires      *
*               intrduits par l'utilisateur     *
*-----------------------------------------------}


{*********************************************}
{ recherche d'une information dans une zone }
{ de l'ordinateur pedagogique                 }

PROCEDURE FC_CONTORD(nomzone : CHAR;
                     nocellule : INTEGER;
                     VAR ordinateur : ordinat_type;
                     VAR contenu : chaincaract;
                     VAR ok : BOOLEAN);EXTERN;

{*********************************************}
{ transformation d'une chaine de caracteres }
{ numeriques en un nombre entier              }

PROCEDURE FC_CARNOMB(repres : chaincaract;
                     VAR nombre : INTEGER);
EXTERN;


{*********************************************}
{ garnissage et affichage d'un mot memoire }
{ de l'ordinateur pedagogique }

PROCEDURE FC_GMOT(VAR val_mot : chaincaract;
                  no_mot : INTEGER;
                  VAR ordinateur : ordinat_type;
                  VAR ok : BOOLEAN;
```

```
                    VAR fsortie : TEXT);EXTERN;

{*****************************************}
{ lecture d'une chaine de caracteres dans }
{      un fichier                         }

PROCEDURE FC_LECTLIGNE(VAR zonlect : chaincaract;
                       long_lire : lonlig;
                       VAR fentree : TEXT);EXTERN;


{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                        }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


{*****************************************}
{ lecture d'un symbole dans une chaine de }
{ caracteres    }

PROCEDURE FC_LECTSYMB(VAR chaine,symbole : chaincaract;
                VAR longsymb,pos_d_lect : loncha);
EXTERN;

{*************************************************}
{ positionnement du curseur sur l'ecran                    }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{*************************************}
{ test de numericite d'une chaine de }
{      caract`res                    }

FUNCTION FC_TESTNUM(VAR symbole : chaincaract;
                    pos_d_symbole : loncha):BOOLEAN;
EXTERN;


(*-------------------------------------------------------------*)
PROCEDURE FC_GMEMCENT(
                       VAR ordinateur : ordinat_type;
                       VAR INPUT,OUTPUT,
                           fmesout     :        TEXT
                     );

VAR   ok,fin   : BOOLEAN;
      (* ok = TRUE si message introduit par l'utilisateur *)
      (* est correct  *)
```

```
      (* fin = TRUE si commande de fin est introduite *)
    tat,     (* zone de reception du message attente *)
    tere,    (* zone de reception du message "erreur donnees"*)
    terc,    (* zone de reception du message"erreur compt. ordinal
    zonlec,     (* zone de reception du message utilisateur  *)
    lig_trav,symbole : chaincaract;
    longsymb : loncha;
    adresse : INTEGER;

  (*-------------------------------------------------------------*)


    PROCEDURE INITZA;  (* initialisation zone d'affichage *)

    BEGIN
      FC_POSITCURS(ligza,colza);
      FC_ECRLIGNE(lig_trav,lza,OUTPUT)
    END;

    PROCEDURE INITZERR;  (* initialisation de la zone d'erreur *)

    BEGIN

      FC_POSITCURS(ligza - nlza + 1,colza);
      FC_ECRLIGNE(lig_trav,lza,OUTPUT)
    END;

    PROCEDURE INITERR;  (* stockage des messages d'erreur *)

    VAR
      pos_d_lect : loncha;
      i : INTEGER;
    BEGIN
      reset(fmesout);

      FC_LECTLIGNE(zonlec,lmaxlig,fmesout);
      pos_d_lect := 1;
      FC_LECTSYMB(zonlec,tat,longsymb,pos_d_lect);
      FC_LECTLIGNE(zonlec,lmaxlig,fmesout);
      pos_d_lect := 1;
      FC_LECTSYMB(zonlec,tere,longsymb,pos_d_lect);

      (* remplissage message erreur compteur ordinal *)

      FOR i := 1 TO 3 DO FC_LECTLIGNE(zonlec,0,fmesout);
      FC_LECTLIGNE(zonlec,lmaxlig,fmesout);
      pos_d_lect := 1;
      FC_LECTSYMB(zonlec,terc,longsymb,pos_d_lect)
    END;

    PROCEDURE LECTURE;  (* enregistrement message *)

    VAR pos_d_lect : loncha;

    BEGIN
```

```
  (* initialisation zone d'affichage *)

  INITZA;

  (* affichage du message "attente" de l'interp. pupitreur *)

  FC_POSITCURS(ligza,colza);
  FC_ECRLIGNE(tat,lza,OUTPUT);

  (* lecture du message introduit par l'utilisateur *)

  FC_POSITCURS(ligza,colza + 2);
  FC_LECTLIGNE(zonlec,lza - 3,INPUT);
  pos_d_lect := 1;
  FC_LECTSYMB(zonlec,symbole,longsymb,pos_d_lect)

END;


PROCEDURE ERREUR(numero_erreur : INTEGER);  (* affichage messa

BEGIN
  (* ecriture message erreur *)

  FC_POSITCURS(ligza - nlza + 1,colza);
  CASE numero_erreur OF
  1 : FC_ECRLIGNE(tere,lza,OUTPUT);
  2 : FC_ECRLIGNE(terc,lza,OUTPUT)
  END
END;

PROCEDURE TRAIT_MOT;
BEGIN
  IF (adresse < (nmotligmem * nmotcolmem))
  THEN BEGIN
  FC_GMOT(symbole,adresse,ordinateur,ok,OUTPUT);
  adresse := adresse + 1
  END
  ELSE ERREUR(2);
END;


PROCEDURE INCREM_CO;
BEGIN
  adresse := adresse + 1
END;

PROCEDURE CONTROLE;

BEGIN
  ok := TRUE;
  IF longsymb > 0
     THEN BEGIN
    IF (longsymb = 1) AND (symbole[1] = 'X')
```

```
        THEN fin := TRUE
        ELSE
          IF longsymb > lmot THEN ok := FALSE
          ELSE BEGIN
          IF symbole[1] = MOINS
          THEN ok := FC_TESTNUM(symbole,2)
          ELSE BEGIN
            IF longsymb > (lmot - 1)
            THEN ok := FALSE
            ELSE ok := FC_TESTNUM(symbole,1)
               END
               END
              END
  END;

(*-------------------------------------------------*)
PROCEDURE INITIALISATION;

  VAR indza : loncha;

  BEGIN
    fin := FALSE;
    FC_CONTORD('O',0,ordinateur,lig_trav,ok);
    FC_CARNOMB(lig_trav,adresse);
    INITERR;
    FOR indza := 1 TO lza DO lig_trav[indza] := blan;
    INITZERR
  END;

PROCEDURE TRAITEMENT;  (* traitement *)

  BEGIN
    ok := TRUE;
    LECTURE;
    CONTROLE;
    INITZERR;

    WHILE ok = FALSE DO
      BEGIN
        ERREUR(1);
        LECTURE;
        INITZERR;
        CONTROLE
      END;

    IF fin <> TRUE
    THEN
      IF longsymb > 0 THEN TRAIT_MOT
          ELSE INCREM_CO

  END;


(*-------------------------------------------------*)
```

```
BEGIN
  INITIALISATION;
  WHILE fin <> TRUE DO TRAITEMENT
END;

(*-------------------------------------------------------------*)
```

```
#
CONST
# include "fgrrcpt"
# include "fmescst"
# include "fcarcst"

TYPE
# include "fordtype"
# include "fmestype"
# include "fcartype"
# include "fgrtype"

{ GARNISSAGE DU REGISTRE A }
{--------------------------}


{*****************************************************}
{ transfert du contenu d'une chaine de caracteres  }
{ dans une autre chaine de caracteres avec remplis-}
{ sages des positions non occupees }

PROCEDURE FC_TRANSFC(VAR emetteur,recepteur : chaincaract;
                     long_emet,lon_recept : loncha;
                     test_signe : BOOLEAN;
                     caract_remplissage : CHAR);EXTERN;


{*************************************************}
{ positionnement du curseur sur l'ecran                }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                         }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


(*-------------------------------------------------------*)
PROCEDURE FC_GREGA(
              VAR message : message_cmd;
              VAR ordinateur : ordinat_type;
              VAR OUTPUT  : TEXT);

VAR j : loncha;
   zon_trav,zonimp : chaincaract;

BEGIN
  FOR j := 1 TO message.param1_long
```

```
    DO zon_trav[j] := message.param1[j];
  FC_TRANSFC(zon_trav, zonimp, message.param1_long, lmot, TRUE, zero)
  FOR j := 1 TO lmot
    DO ordinateur.ra[j] := zonimp[j];
  FC_POSITCURS(liga, cola);
  FC_ECRLIGNE(zonimp, lmot, OUTPUT)
END;
```

```
#
CONST
# include "fgrrcpt"
# include "fmescst"
# include "fcarcst"

TYPE
# include "fordtype"
# include "fmestype"
# include "fcartype"
# include "fgrtype"

{ GARNISSAGE DU COMPTEUR D'ADRESSES }
{-----------------------------------}


{*****************************************************}
{ transfert du contenu d'une chaine de caracteres  }
{ dans une autre chaine de caracteres avec remplis-}
{ sages des positions non occupees }

PROCEDURE FC_TRANSFC(VAR emetteur,recepteur : chaincaract;
                     long_emet,lon_recept : loncha;
                     test_signe : BOOLEAN;
                     caract_remplissage : CHAR);EXTERN;


{*************************************************}
{ positionnement du curseur sur l'ecran                }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                         }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


(*-----------------------------------------------*)
PROCEDURE FC_GREGC(
                VAR message : message_cmd;
                VAR ordinateur : ordinat_type;
                VAR OUTPUT  : text);

VAR j : loncha;
   zon_trav,zonimp : chaincaract;

BEGIN
  FOR j := 1 TO message.parami_long
```

```
    DO zon_trav[j] := message.param1[j];
  FC_TRANSFC(zon_trav,zonimp,message.param1_long,lco,FALSE,zero)
  FOR j := 1 TO lco
    DO ordinateur.co[j] := zonimp[j];
  FC_POSITCURS(ligco,colco);
  FC_ECRLIGNE(zonimp,lco,OUTPUT)
END;
```

```
#
CONST
# include "fgrrcpt"
# include "fmescst"
# include "fcarcst"

TYPE
# include "fordtype"
# include "fmestype"
# include "fcartype"
# include "fgrtype"

{ GARNISSAGE DU REGISTRE INSTRUCTION }
{-------------------------------------}



{*************************************************}
{ transfert du contenu d'une chaine de caracteres  }
{ dans une autre chaine de caracteres avec remplis-}
{ sages des positions non occupees }

PROCEDURE FC_TRANSFC(VAR emetteur,recepteur : chaincaract;
                     long_emet,lon_recept : loncha;
                     test_signe : BOOLEAN;
                     caract_remplissage : CHAR);EXTERN;

{*************************************************}
{ positionnement du curseur sur l'ecran                }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                         }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


(*-----------------------------------------------------------*)
PROCEDURE FC_GREGI(
                VAR message : message_cmd;
                VAR ordinateur : ordinat_type;
                VAR OUTPUT  : text);

VAR j : loncha;
   zon_trav,zonimp : chaincaract;

BEGIN
  FOR j := 1 TO message.param1_long
```

```
    DO zon_trav[j] := message.param1[j];
  FC_TRANSFC(zon_trav,zonimp,message.param1_long,lmot,TRUE,zero)
  FOR j := 1 TO lmot
    DO ordinateur.ri[j] := zonimp[j];
  FC_POSITCURS(ligri,colri);
  FC_ECRLIGNE(zonimp,lmot,OUTPUT)
END;
```

```
#
CONST
# include "fgrrcpt"
# include "fmescst"
# include "fcarcst"

TYPE
# include "fordtype"
# include "fmestype"
# include "fcartype"
# include "fgrtype"

{ GARNISSAGE DU REGISTRE EXTENSION }
{-----------------------------------}



{*************************************************}
{ transfert du contenu d'une chaine de caracteres  }
{ dans une autre chaine de caracteres avec remplis-}
{ sages des positions non occupees }

PROCEDURE FC_TRANSFC(VAR emetteur,recepteur : chaincaract;
                     long_emet,lon_recept : loncha;
                     test_signe : BOOLEAN;
                     caract_remplissage : CHAR);EXTERN;

{************************************************}
{ positionnement du curseur sur l'ecran                }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                         }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


(*-------------------------------------------------------*)
PROCEDURE FC_GREGE(
                VAR message : message_cmd;
                VAR ordinateur : ordinat_type;
                VAR OUTPUT  : text);

VAR j : loncha;
   zon_trav,zonimp : chaincaract;

BEGIN
  FOR j := 1 TO message.param1_long
```

```
    DO zon_trav[j] := message.param1[j];
  FC_TRANSFC(zon_trav,zonimp,message.param1_long,lmot,TRUE,zero)
  FOR j := 1 TO lmot
    DO ordinateur.re[j] := zonimp[j];
  FC_POSITCURS(ligre,colre);
  FC_ECRLIGNE(zonimp,lmot,OUTPUT)
END;
```

```
#
CONST
# include "fgrrcpt"
# include "fmescst"
# include "fcarcst"

TYPE
# include "fordtype"
# include "fmestype"
# include "fcartype"
# include "fgrtype"

{ GARNISSAGE DE L'INDICATEUR LOGIQUE }
{-------------------------------------}



{*************************************************}
{ transfert du contenu d'une chaine de caracteres  }
{ dans une autre chaine de caracteres avec remplis-}
{ sages des positions non occupees }

PROCEDURE FC_TRANSFC(VAR emetteur,recepteur : chaincaract;
                     long_emet,lon_recept : loncha;
                     test_signe : BOOLEAN;
                     caract_remplissage : CHAR);EXTERN;


{************************************************}
{ positionnement du curseur sur l'ecran                }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                         }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


(*-------------------------------------------------------*)
PROCEDURE FC_GREGL(
                VAR message : message_cmd;
                VAR ordinateur : ordinat_type;
                VAR OUTPUT  : text);

VAR j : loncha;
   zon_trav,zonimp : chaincaract;

BEGIN
  FOR j := 1 TO message.param1_long
```

```
    DO zon_trav[j] := message.param1[j];
  FC_TRANSFC(zon_trav,zonimp,message.param1_long,lil,FALSE,blan)
  FOR j := 1 TO lil
    DO ordinateur.il[j] := zonimp[j];
  FC_POSITCURS(ligil,colil);
  FC_ECRLIGNE(zonimp,lil,OUTPUT)
END;
```

```
#

{*-----------------------------------------*}
{*                                         *}
{*   Initialisation de l'ordinateur        *}
{*                                         *}
{*-----------------------------------------*}

CONST
# include "fmescst"
# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fordtype"
# include "fmestype"
# include "fcartype"
# include "fgrtype"

{*****************************************}
{ garnissage et affichage de la zone      }
{ registre A de l'ordinateur pedagogique }

PROCEDURE FC_GREGA(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;

{*****************************************}
{ garnissage et affichage de la zone      }
{ registre I de l'ordinateur pedagogique }

PROCEDURE FC_GREGI(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;

{*****************************************}
{ garnissage et affichage de la zone      }
{ registre C de l'ordinateur pedagogique }

PROCEDURE FC_GREGC(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;

{*****************************************}
{ garnissage et affichage de la zone      }
{ registre E de l'ordinateur pedagogique }

PROCEDURE FC_GREGE(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;
```

```
{*****************************************}
{ garnissage et affichage de la zone      }
{ registre L de l'ordinateur pedagogique }

PROCEDURE FC_GREGL(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;


{*****************************************}
{ garnissage et affichage d'un mot memoire }
{ de l'ordinateur pedagogique }

PROCEDURE FC_GMOT(VAR val_mot : chaincaract;
                  no_mot : INTEGER;
                  VAR ordinateur : ordinat_type;
                  VAR ok : BOOLEAN;
                  VAR fsortie : TEXT);EXTERN;

{*****************************************}
{ granissage et affichage de la zone code }
{ operatoire de l'ordinateur pedagogique  }

PROCEDURE FC_GCOP(VAR contenu : chaincaract;
                  VAR ordinateur : ordinat_type;
                  VAR fsortie : TEXT);
EXTERN;
{*************************************************}
{ positionnement du curseur sur l'ecran                    }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;
{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                         }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;



PROCEDURE FC_INITMACH(VAR ordinateur : ordinat_type;
                      VAR OUTPUT : TEXT);

VAR
       meszero,mesblan : message_cmd;
       chzero,chblan   : chaincaract;

PROCEDURE INITIALISATION;
```

```
VAR
  j : INTEGER;

BEGIN
  FOR j := 1 TO lmot DO
  BEGIN
    meszero.param1[j] := '0';
    mesblan.param1[j] := blan;
    chzero[j] := '0';
    chblan[j] := blan
  END;

  chzero[lmot + 1] := fdl;
  chblan[lmot + 1] := fdl;
END;

PROCEDURE MAZ_MEMOIRE;

VAR
  j : INTEGER;
  ok : BOOLEAN;
BEGIN
  FOR j := 1 TO (nmotligmem * nmotcolmem)
    DO FC_GMOT(chzero, j - 1, ordinateur, ok, OUTPUT)
END;


PROCEDURE MAZ_REG;

BEGIN
  meszero.param1_long := lmot;
  FC_GREGA(meszero, ordinateur, OUTPUT);
  FC_GREGE(meszero, ordinateur, OUTPUT);
  FC_GREGI(meszero, ordinateur, OUTPUT);
  meszero.param1_long := lco;
  FC_GREGC(meszero, ordinateur, OUTPUT)
END;

PROCEDURE MAB_AUTRES;

BEGIN
  mesblan.param1_long := lil;
  FC_GREGL(mesblan, ordinateur, OUTPUT);
  chblan[lcop + 1] := fdl;
  FC_GCOP(chblan, ordinateur, OUTPUT)
END;

PROCEDURE VID_LECT;

VAR
  init_carte : chaincaract;
  i : INTEGER;
  debut_carte : INTEGER;

BEGIN
```

```
  debut_carte := ligcart1;
  ordinateur.no_carte := 0;
  ordinateur.cartes[i,1] := fd1;
  FOR i := 1 TO lcart
    DO init_cart[i] := blan;
  FOR i := 1 TO ncart
  DO
  BEGIN
    FC_POSITCURS(debut_carte,colcart1);
    FC_ECRLIGNE(init_carte,lcart,OUTPUT);
    debut_carte := debut_carte + hautcart + 1
  END
END;


PROCEDURE VID_IMPRIM;

VAR

  debut_ligne : INTEGER;
  init_imp : chaincaract;
  i : INTEGER;

BEGIN

  debut_ligne := ligimp1;
  FOR i := 1 TO llig DO init_imp[i] := blan;
  FOR i := 1 TO nlig
  DO
  BEGIN
     FC_POSITCURS(debut_ligne,colimp1);
     FC_ECRLIGNE(init_imp,llig,OUTPUT);
     debut_ligne := debut_ligne + hautlig + 1
  END
END;

PROCEDURE TRAITEMENT;
BEGIN
  MAZ_MEMOIRE;
  MAZ_REG;
  MAB_AUTRES;
  VID_LECT;
  VID_IMPRIM
END;

{-------------------------------------------------------}

BEGIN
  INITIALISATION;
  TRAITEMENT
END;
```

```
#
{*-------------------------------------------------------------------*}
{*                                                                   *}
{*      CHARGEMENT DE LA MEMOIRE CENTRALE                            *}
{*                                                                   *}
{*-------------------------------------------------------------------*}

CONST
# include "fcarcst"
# include "fmescst"
# include "fgrrcpt"
delimit = '*';

TYPE
# include "fcartype"
# include "fmestype"
# include "fordtype"

{**********************************}
{ lecture du contenu d'un fichier }
{ suivant un nom de fichier donne }

PROCEDURE FLECTURE(VAR nomfichier : chaincaract;
                   VAR ok : BOOLEAN;
                   VAR fichier : TEXT);EXTERN;

{*****************************************}
{ recherche d'une information dans une zone }
{ de l'ordinateur pedagogique               }

PROCEDURE FC_CONTORD(nomzone : CHAR;
                     nocellule : INTEGER;
                     VAR ordinateur : ordinat_type;
                     VAR contenu : chaincaract;
                     VAR ok : BOOLEAN);EXTERN;

{************************************}
{ test de numericite d'une chaine de }
{      caract^res                    }

FUNCTION FC_TESTNUM(VAR symbole : chaincaract;
                    pos_d_symbole : loncha):BOOLEAN;
EXTERN;


{*******************************************}
{ transformation d'une chaine de caracteres }
{ numeriques en un nombre entier            }

PROCEDURE FC_CARNOMB(repres : chaincaract;
                     VAR nombre : INTEGER);
EXTERN;

{*****************************************}
{ lecture d'une chaine de caracteres dans }
```

```
{      un fichier                                }

PROCEDURE FC_LECTLIGNE(VAR zonlect : chaincaract;
                       long_lire : lonlig;
                       VAR fentree : TEXT);EXTERN;

{*********************************}
{ garnissage de la zone d'erreur }
{ du moniteur }

PROCEDURE FC_ERRMON(code_erreur : INTEGER;
                    VAR fmessage,OUTPUT : TEXT);EXTERN;

{*****************************************}
{ garnissage et affichage de la zone      }
{ registre C de l'ordinateur pedagogique }

PROCEDURE FC_GREGC(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;

{*******************************************}
{ garnissage et affichage d'un mot memoire }
{ de l'ordinateur pedagogique }

PROCEDURE FC_GMOT(VAR val_mot : chaincaract;
                  no_mot : INTEGER;
                  VAR ordinateur : ordinat_type;
                  VAR ok : BOOLEAN;
                  VAR fsortie : TEXT);EXTERN;

{*****************************************}
{ incrementation du contenu du compteur }
{ d'adresses de l'ordinateur pedagogique}

PROCEDURE FC_INCREMCO(VAR ordinateur : ordinat_type;
                      VAR anc_co : INTEGER;
                      VAR ok : BOOLEAN;VAR fsortie : TEXT);EXTE



PROCEDURE FC_CHMEMCENT(
                      VAR message : message_cmd;
                      VAR ordinateur : ordinat_type;
                      VAR fmesout,OUTPUT : TEXT
                     );

{*********************************************************
*                                                        *
* ENTREE : l'identification du fichier a garnir dans la *
*          la memoire de l'ordinateur pedagogique;       *
*                                                        *
* SORTIE : le remplissage ou l'affichage des zones      *
*          garnies de la memoire centrale ou l'envoit    *
```

```
*         d'un message d'erreur a l'utilisateur          *
*                                                        *
* FONCTION :    le contenu du fichier est charge dans la *
*              memoire centrale a partir de la valeur    *
*               contenu dans le compteur ordinal ou par  *
*               la valeur debut adresse existant dans le*
*               fichier, ce dernier prevaut sur la (CO)  *
*                                                        *
**********************************************************}

VAR
        code_erreur : INTEGER;
        fichier : TEXT;
        motlu, zonlect : chaincaract;
        deb_lect : loncha;


PROCEDURE OBTMOT(VAR succes : BOOLEAN);

VAR
        longlu : loncha;

BEGIN
        succes := TRUE;
        longlu := 0;
        WHILE  (longlu <> lmot) DO
        BEGIN
        WHILE (zonlect[deb_lect] <> fdl) AND (longlu <> lmot)
        DO
        BEGIN
                longlu := longlu + 1;
                motlu[longlu] := zonlect[deb_lect];
                deb_lect := deb_lect + 1
        END;
        IF longlu <> lmot
        THEN
        BEGIN
                IF NOT EOF(fichier)
                THEN FC_LECTLIGNE(zonlect, lmaxlig, fichier)
                ELSE
                   BEGIN
                        succes := FALSE;
                        longlu := lmot
                   END;
                deb_lect := 1
        END
  END;
  motlu[lmot + 1] := fdl;
END;

PROCEDURE INITIALISATION;

VAR
        j : loncha;
        nomfichier : chaincaract;
```

```
        ok : BOOLEAN;

BEGIN

        code_erreur := 0;
        WITH message DO
        BEGIN
                FOR j := 1 TO param1_long DO
                        nomfichier[j] := param1[j];
                nomfichier[param1_long + 1] := fdl;
        END;
        FLECTURE(nomfichier,ok,fichier);
        IF NOT ok THEN code_erreur := 2
END;

PROCEDURE TRAITEMENT;

VAR
        trav : chaincaract;
        ok,succes : BOOLEAN;
        i,j : loncha;
        adresse : INTEGER;

BEGIN
        RESET(fichier);
        FC_LECTLIGNE(zonlect,lmaxlig,fichier);
        deb_lect := 1;
        j := 1;

        IF zonlect[j] = delimit THEN
        BEGIN
                WHILE zonlect[j] = delimit DO j := j + 1;
                i := 1;
                WHILE zonlect[j] <> delimit DO
                BEGIN
                        trav[i] := zonlect[j];
                        i := i + 1;
                        j := j + 1
                END;
                trav[i] := fdl;
                WHILE zonlect[j] = delimit DO j := j + 1;
                deb_lect := j;
                IF (i <= (lco + 1)) AND (FC_TESTNUM(trav,1) = TRU
                THEN
                BEGIN
                FC_CARNOMB(trav,adresse);
                IF adresse < (nmotligmem * nmotcolmem)
                THEN
                BEGIN
                        FOR j := 1 TO (i - 1) DO
                                message.param1[j] := trav[j];
                        message.param1_long := (i - 1);
                        FC_GREGC(message,ordinateur,OUTPUT)
                        END
                        ELSE code_erreur := 2
```

```
                END
                ELSE code_erreur := 2
        END;

        IF code_erreur = 0
        THEN
        BEGIN
        FC_CONTORD('O',0,ordinateur,trav,ok);
        FC_CARNOMB(trav,adresse);

        OBTMOT(succes);
        WHILE succes  AND (adresse < (nmotligmem * nmotcolmem))
        DO
        BEGIN
                FC_GMOT(motlu,adresse,ordinateur,ok,OUTPUT);
                adresse := adresse + 1;
                OBTMOT(succes)
        END;

        IF succes AND (adresse = (nmotligmem * nmotcolmem))
        THEN code_erreur := 2
        END
END;

PROCEDURE CLOTURE;

BEGIN

        IF code_erreur <> 0
        THEN FC_ERRMON(code_erreur,fmesout,OUTPUT)
END;


{-----------------------------------------------}

BEGIN
        INITIALISATION;
        IF code_erreur = 0 THEN TRAITEMENT;
        CLOTURE
END;
```

```
#
{*-------------------------------------*}
{*   REGLAGE DELAI ENTRE CHAQUE    *}
{*     INSTRUCTION                 *}
{*-------------------------------------*}

CONST
# include "fgrrcpt"
# include "fmescst"
# include "fcarcst"
TYPE
# include "fmestype"
# include "fcartype"
# include "fordtype"

{*********************************************}
{ transformation d'une chaine de caracteres }
{ numeriques en un nombre entier            }

PROCEDURE FC_CARNOMB(repres : chaincaract;
                     VAR nombre : INTEGER);
EXTERN;


PROCEDURE FC_RARINST(VAR message : message_cmd;
                     VAR ordinateur : ordinat_type);

VAR
       zontrav : chaincaract;
       j,vitesse : INTEGER;

BEGIN
  IF message.param1_long > 2 THEN vitesse := 99
  ELSE
    BEGIN
    FOR j := 1 TO message.param1_long
      DO zontrav[j] := message.param1[j];
    zontrav[message.param1_long + 1] := fdl;
    END;
  FC_CARNOMB(zontrav,vitesse);
  ordinateur.vi := vitesse
END;
```

```
#
{*-------------------------------------------*}
{*                                           *}
{*    EXECUTION D'UNE INSTRUCTION            *}
{*              MACHINE                      *}
{*                                           *}
{*-------------------------------------------*}

CONST
halt = 91;
# include "fcarcst"
# include "fgrrcpt"
# include "fmescst"

TYPE
# include "fmestype"
# include "fcartype"
# include "fordtype"
# include "fgrtype"

{-------------------------------------------
* ENTREE : le contenu de l'ordinateur  *
*          pedagogique;                *
*                                      *
* SORTIE : execution de l'instruction  *
*          contenue dans le registre   *
*          instruction ou envoit d'un  *
*          message d'erreurs;          *
*                                      *
* FONCTION : lecture, controle et      *
*            execution de l'instruction*
*            contenu dans le registre  *
*            instruction.              *
-------------------------------------------}


{*****************************************}
{ recherche d'un enregistrement suivant }
{ sa position dans un fichier           }

PROCEDURE FC_VALASS(pos_enregist : INTEGER;
                    VAR enregistrement : chaincaract;
                    VAR fichier : TEXT);
EXTERN;

{*****************************************}
{ lecture d'un symbole dans une chaine de }
{ caracteres   }

PROCEDURE FC_LECTSYMB(VAR chaine,symbole : chaincaract;
                      VAR longsymb,pos_d_lect : loncha);
EXTERN;
{*************************************************}
{ transfert du contenu d'une chaine de caracteres  }
{ dans une autre chaine de caracteres avec remplis-}
```

```
{ sages des positions non occupees }

PROCEDURE FC_TRANSFC(VAR emetteur,recepteur : chaincaract;
                     long_emet,lon_recept : loncha;
                     test_signe : BOOLEAN;
                     caract_remplissage : CHAR);EXTERN;

{*****************************************}
{ lecture d'une chaine de caracteres dans }
{      un fichier                         }

PROCEDURE FC_LECTLIGNE(VAR zonlect : chaincaract;
                       long_lire : lonlig;
                       VAR fentree : TEXT);EXTERN;

{************************************************}
{ positionnement du curseur sur l'ecran                }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;
{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                        }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;

{****************************************}
{ incrementation du contenu du compteur }
{ d'adresses de l'ordinateur pedagogique}

PROCEDURE FC_INCREMCO(VAR ordinateur : ordinat_type;
                VAR anc_co : INTEGER;
                VAR ok : BOOLEAN;VAR fsortie : TEXT);EXTERN;

{*********************************************}
{ recherche d'une information dans une zone }
{ de l'ordinateur pedagogique                 }

PROCEDURE FC_CONTORD(nomzone : CHAR;
                     nocellule : INTEGER;
                     VAR ordinateur : ordinat_type;
                     VAR contenu : chaincaract;
                     VAR ok : BOOLEAN);EXTERN;

{*****************************************}
{ garnissage et affichage de la zone      }
{ registre I de l'ordinateur pedagogique }

PROCEDURE FC_GREGI(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;
```

```
{*****************************************}
{ garnissage et affichage de la zone      }
{ registre C de l'ordinateur pedagogique }

PROCEDURE FC_GREGC(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;

{*****************************************}
{ garnissage et affichage de la zone      }
{ registre L de l'ordinateur pedagogique }

PROCEDURE FC_GREGL(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;

{*****************************************}
{ garnissage et affichage de la zone      }
{ registre A de l'ordinateur pedagogique }

PROCEDURE FC_GREGA(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;
{*****************************************}
{ garnissage et affichage de la zone      }
{ registre E de l'ordinateur pedagogique }

PROCEDURE FC_GREGE(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;

{*****************************************}
{ garnissage et affichage d'un mot memoire }
{ de l'ordinateur pedagogique }

PROCEDURE FC_GMOT(VAR val_mot : chaincaract;
                  no_mot : INTEGER;
                  VAR ordinateur : ordinat_type;
                  VAR ok : BOOLEAN;
                  VAR fsortie : TEXT);EXTERN;

{*************************************}
{ test de numericite d'une chaine de }
{      caract`res                     }

FUNCTION FC_TESTNUM(VAR symbole : chaincaract;
                    pos_d_symbole : loncha):BOOLEAN;
EXTERN;

{*****************************************}
```

```
{ localisation d'un symbole dans une table }
{ de symboles se trouvant dans un fichier  }

PROCEDURE FC_LOCALSYMB(
                       VAR symbole : chaincaract;
                       pos_d_symb : loncha;
                        VAR pos_table : INTEGER;
                        VAR tabsymbole : TEXT);
EXTERN;

{*******************************************}
{ transformation d'une chaine de caracteres }
{ numeriques en un nombre entier            }

PROCEDURE FC_CARNOMB(repres : chaincaract;
                     VAR nombre : INTEGER);
EXTERN;

{*******************************************}
{ transformation d'un nombre entier positif }
{ en une chaine de caracteres               }

PROCEDURE FC_NOMBCAR(nombre : INTEGER;
                     VAR repres : chaincaract);
EXTERN;

{*********************************}
{ garnissage de la zone d'erreur }
{ du moniteur }

PROCEDURE FC_ERRMON(code_erreur : INTEGER;
                    VAR fmessage,OUTPUT : TEXT);EXTERN;

{*****************************************}
{ granissage et affichage de la zone code }
{ operatoire de l'ordinateur pedagogique  }

PROCEDURE FC_GCOP(VAR contenu : chaincaract;
                  VAR ordinateur : ordinat_type;
                  VAR fsortie : TEXT);
EXTERN;


{*-------------------------------------------------------------------*}

PROCEDURE FC_EXINST(VAR ordinateur : ordinat_type;
                {* description interne de l'ordinateur *}
                    VAR ok : BOOLEAN;
                {* valeur resultat du bon fonctionnement de *}
                        {* l'operation *}
                    VAR OUTPUT,fmesout,fcopma : TEXT
                {* fichier representant le terminal *}
                {* fichier des messages de sortie *}
        {* fichier des codes operations du langage machine *}
                   );
```

```
VAR
        nophase,
          {* no de la phase dans le traitement d'une *}
                    {* instruction machine *}
        code_erreur,
                {* no de l'erreur pendant le deroulement *}
                      {* du traitement de l'instruction *}
        nomb12,
                {* valeur du champ 1 & 2 de l'instruction machine
        nomb3,
                {* valeur du champ 3 de l'instruction machine *}
        nomb4,
                {* valeur du champ 4 de l'instruction machine *}
        adresse
                {* valeur de l'adresse de l'argument *}
                : INTEGER;
        ok_increm : BOOLEAN;
                {* valeur signalant la reussite de *}
                {* l'incrementation du CO       *}
        message : message_cmd;
        reg_i : chaincaract;

{-------------------------------------------------------------------
* ENTREE : - la representation interne de l'ordinateur pedagogiq
*
* SORTIE : - une valeur signalant la reussite de l'operation,
*          - affichage du resultat de l'operation (message erreur
*            ou excution de l'instruction);
*
* FONCTION : - recherche de la valeur du reg instruction suivant
*              la valeur contenue dans le compteur ordinal,
*            - analyse de la validite du contenu du reg instructi
*            - si correct alors execution de l'instruction,
*            - affichage du resultat;
-------------------------------------------------------------------

PROCEDURE CARREEL(repres : chaincaract;VAR nbreel : REAL);

        {* transformation chaine de caracteres ---> nombre reel *

VAR
   j : INTEGER;
   signe_negatif : BOOLEAN;

BEGIN
   nbreel := 0;
   signe_negatif := FALSE;
   j := 1;
   WHILE repres[j] = blan DO j := j + 1;
   IF repres[j] = moins THEN
            BEGIN
            signe_negatif := TRUE;
            j := j + 1
            END;
```

```
    WHILE repres[j] = '0' DO j := j + 1;
    WHILE repres[j] <> fdl DO
       BEGIN
       nbreel := (nbreel * 10) + (ORD(repres[j]) - ORD('0'));
       j := j + 1
       END;
    IF signe_negatif = TRUE THEN nbreel := nbreel * (0 - 1)
END;

PROCEDURE REELCAR(nbreel : REAL; VAR repres : chaincaract);

 {* transformation partie entiere du nbre reel -> chaine caracte
    {* 1er caractere de la chaine = signe , blan pour positif et
    {* '-' pour negatif *}

VAR
    ficnomb : TEXT;
    trav : chaincaract;
    i, j : INTEGER;
    auxil : REAL;

BEGIN
    auxil := nbreel;
    IF nbreel < 0 THEN nbreel := nbreel * (0 - 1);
    REWRITE(ficnomb);
    WRITELN(ficnomb, nbreel : 13 : 0);
    RESET(ficnomb);
    FC_LECTLIGNE(trav, 10, ficnomb);
    IF auxil < 0
       THEN repres[1] := moins
       ELSE repres[1] := plus;
    j := 2;
    i := 1;
    WHILE trav[i] = blan DO i := i + 1;
    WHILE trav[i] <> fdl
    DO
    BEGIN
       repres[j] := trav[i];
       j := j + 1;
       i := i + 1
    END;
    repres[j] := fdl
END;

PROCEDURE REMPL_MSG(information : chaincaract);

    {* transfert de l'information dans un format message *}

VAR
    j : INTEGER;

BEGIN
    j := 1;
    WHILE information[j] <> fdl
       DO BEGIN
```

```
            message.param1[j] := information[j];
            j := j + 1
            END;
    message.param1_long := j - 1
END;

PROCEDURE ARG(VAR argument : chaincaract);

    {* recherche de la valeur dans l'ordinateur suivant une adres

VAR
     j : INTEGER;
     ok : BOOLEAN;

BEGIN
    CASE nomb3 OF
    0 : BEGIN
        FOR j := 1 TO 2 DO argument[j] := reg_i[j + 4];
        argument[3] := fd1
         END;
    1,2,4,5 : BEGIN
        FC_CONTORD('M',adresse,ordinateur,argument,ok);
        IF ok = FALSE THEN code_erreur := 4
          END;
    3 : FC_CONTORD('E',0,ordinateur,argument,ok)
    END
END;
PROCEDURE GR_DIVERS;

BEGIN
    CASE nomb12 OF

    90    : ; {NOP}
    91    : ; {HLT}
    92     : {DOV}
          ordinateur.re_ov := FALSE;
    93    : {EOV}
          ordinateur.re_ov := TRUE
    END
END;


PROCEDURE GR_BRANCHEMENT;

VAR
    gotoadr,contenu : chaincaract;
    ok : BOOLEAN;

BEGIN
    gotoadr[1] := fd1;
    CASE nomb12 OF

    40    : ARG(gotoadr);
    41    : BEGIN
          IF ordinateur.il[1il] = '=' THEN ARG(gotoadr)
```

```
        END;
    42  : BEGIN
        IF ordinateur.il[lil] = '<' THEN ARG(gotoadr)
        END;
    43  : BEGIN
        IF ordinateur.il[lil] = '>' THEN ARG(gotoadr)
        END;
    44  :BEGIN
        IF (ordinateur.il[lil] = '<') OR
           (ordinateur.il[lil] = '>') THEN ARG(gotoadr)
        END;
    45  :BEGIN
        IF (ordinateur.il[lil] = '<') OR
           (ordinateur.il[lil] = '=') THEN ARG(gotoadr)
        END;
    46  :BEGIN
        IF (ordinateur.il[lil] = '>') OR
           (ordinateur.il[lil] = '=') THEN ARG(gotoadr)
        END;
    47  :BEGIN
        IF (ordinateur.il[lil] = 'V') AND
           (ordinateur.il[lil] = 'O') THEN ARG(gotoadr)
        END;
    48  :BEGIN
        IF (ordinateur.il[lil] = 'R') AND
           (ordinateur.il[lil] = 'E') THEN ARG(gotoadr)
        END;
    49  : BEGIN
        ARG(gotoadr);
        FC_CONTORD('O',0,ordinateur,contenu,ok);
        REMPL_MSG(contenu);
        FC_GREGA(message,ordinateur,OUTPUT);
        END;
    END;
    IF gotoadr[1] <> fdl
       THEN BEGIN
         REMPL_MSG(gotoadr);
         FC_GREGC(message,ordinateur,OUTPUT)
           END
END;

PROCEDURE GR_COMPARAISON;

VAR
    accumul,operande,chcop : chaincaract;
    oper2,oper1 : REAL;
    ok : BOOLEAN;
BEGIN
    FC_CONTORD('A',0,ordinateur,accumul,ok);
    CARREEL(accumul,oper1);
    ARG(operande);
    IF code_erreur = 0 THEN
    BEGIN
       CARREEL(operande,oper2);
       IF oper1 < oper2 THEN chcop[1] := '<';
```

```
      IF oper1 = oper2 THEN chcop[1] := '=';
      IF oper1 > oper2 THEN chcop[1] := '>';
      chcop[2] := fdl;
      REMPL_MSG(chcop);
      FC_GREGL(message,ordinateur,OUTPUT)
   END
END;

PROCEDURE GR_ARITHMETIQUE;

VAR
   oper1,oper2,result : REAL;
   accumul,operande,resultat,chcop : chaincaract;
   longueur,i,j : INTEGER;

BEGIN
   IF (20 <= nomb12) AND (nomb12 <= 25)
   THEN
   BEGIN
      FC_CONTORD('A',0,ordinateur,accumul,ok);
      CARREEL(accumul,oper1);
      IF (20 <= nomb12) AND (nomb12 <= 23)
      THEN
      BEGIN
         ARG(operande);
         IF code_erreur = 0
         THEN
         BEGIN
         CARREEL(operande,oper2);
         CASE nomb12 OF

         20 : {ADD}
            result := oper1 + oper2;
         21 : {SUB}
            result := oper1 - oper2;
         22 : {MUL}
            result := oper1 * oper2;
         23 : {DIV}
            BEGIN
            IF oper2 = 0.0
            THEN code_erreur := 3
            ELSE result := TRUNC(oper1/oper2);
            END;
         END;
         IF code_erreur = 0
         THEN BEGIN
         REELCAR(result,resultat);
         longueur := 1;
         WHILE resultat[longueur] <> fdl
         DO longueur := longueur + 1;
         longueur := longueur - 1;
         IF ((result >= 0) AND (longueur < lmot))
           OR ((result < 0) AND (longueur <= lmot))
         THEN
         BEGIN
```

```
            REMPL_MSG(resultat);
            FC_GREGA(message,ordinateur,OUTPUT)
         END
         ELSE
         BEGIN
            i := 2;
            FOR j := (longueur - (lmot - 1) + 1) TO longueur
            DO
            BEGIN
            accumul[i] := resultat[j];
            i := i + 1
            END;
            accumul[1] := blan;
            IF ordinateur.re_ov = FALSE
            THEN accumul[1] := resultat[1];
            REMPL_MSG(accumul);
            FC_GREGA(message,ordinateur,OUTPUT);
            IF ordinateur.re_ov = TRUE
            THEN
            BEGIN
              resultat[longueur + 1 - (lmot - 1)] := fdl;
              REMPL_MSG(resultat);
              FC_GREGE(message,ordinateur,OUTPUT)
            END;
         END;
          END;
       END;
    END;
    IF (24 <= nomb12) AND (nomb12 <= 25)
    THEN
    BEGIN
       CASE nomb12 OF

       24    : BEGIN
                result := oper1 * (0 - 1)
               END;
       25    : BEGIN
                result := ABS(oper1)
               END;
       END;
       REELCAR(result,resultat);
       REMPL_MSG(resultat);
       FC_GREGA(message,ordinateur,OUTPUT)
    END;

    IF code_erreur = 0
    THEN
    BEGIN
       {* garnissage de l'indicateur logique *}

       j := 1;
       IF result < 0 THEN chcop[j] := '<';
       IF result = 0 THEN chcop[j] := '=';
       IF result > 0 THEN chcop[j] := '>';
       IF (longueur > lmot) AND
```

```
            (ordinateur.re_ov = FALSE)
         THEN BEGIN
            chcop[j] := 'O';
            j := 2;
            chcop[j] := 'V';
              END;
         chcop[j + 1] := fdl;
         REMPL_MSG(chcop);
         FC_GREGL(message,ordinateur,OUTPUT)
      END
   END
END;

PROCEDURE GR_TRANSFERT;

VAR
   operande,accumul : chaincaract;
   ok : BOOLEAN;

BEGIN
   CASE nomb12 OF

   10 : {* LDA *}
        BEGIN
      ARG(operande);
      IF code_erreur = 0
      THEN BEGIN
         REMPL_MSG(operande);
         FC_GREGA(message,ordinateur,OUTPUT)
           END
        END;

   11 : {* STA *}
        BEGIN
      FC_CONTORD('A',0,ordinateur,accumul,ok);
      IF nomb3 = 3
      THEN BEGIN
         REMPL_MSG(accumul);
         FC_GREGE(message,ordinateur,OUTPUT)
           END
      ELSE BEGIN
         FC_GMOT(accumul,adresse,ordinateur,ok,OUTPUT);
         IF ok = FALSE THEN code_erreur := 4
           END
        END;

   12 : {* EXC *}
        BEGIN
      ARG(operande);
      IF code_erreur = 0
      THEN BEGIN
         FC_CONTORD('A',0,ordinateur,accumul,ok);
         REMPL_MSG(operande);
         FC_GREGA(message,ordinateur,OUTPUT);
         IF nomb3 = 3
```

```
            THEN BEGIN
               REMPL_MSG(accumul);
               FC_GREGE(message,ordinateur,OUTPUT)
                 END
            ELSE BEGIN
               FC_GMOT(accumul,adresse,ordinateur,ok,OUTPUT);
               IF ok = FALSE THEN code_erreur := 4
                 END
              END
         END
   END; {* CASE *}

END;


{*-----------------------------------------------------------------*}

PROCEDURE EXEC_INSTRUCTION;

VAR
   champ1 : CHAR;
BEGIN
   champ1 := reg_i[2];
   CASE champ1 OF
   '1'    : GR_TRANSFERT;
   '2'    : GR_ARITHMETIQUE;
   '3'    : GR_COMPARAISON;
   '4'    : GR_BRANCHEMENT;
   '9'    : GR_DIVERS;
   END

END;

PROCEDURE REC_INSTRUCTION;

VAR
   ok : BOOLEAN;
   val_co : INTEGER;

BEGIN
   {* obtention du contenu du CO + incrementation de ce dernier

   FC_INCREMCO(ordinateur,val_co,ok_increm,OUTPUT);
   {* obtention contenu mot memoire adresse par val_co *}
   FC_CONTORD('M',val_co,ordinateur,reg_i,ok);
   IF ok = FALSE THEN code_erreur := 6
   ELSE BEGIN
      {* garnissage reg instruction *}
      REMPL_MSG(reg_i);
      FC_GREGI(message,ordinateur,OUTPUT)
       END
END;

PROCEDURE SYNT_INSTRUCTION;
```

```
BEGIN
      {* test numerique *}
      IF FC_TESTNUM(reg_i,2) = FALSE THEN code_erreur := 2
END;

PROCEDURE REC_ADOPER;

VAR
   champ4,contenu : chaincaract;
   long : loncha;
   ok : BOOLEAN;
   j : loncha;

BEGIN
   FOR j := 1 TO 2 DO champ4[j] := reg_i[j+4];
   champ4[3] := fdl;
   FC_CARNOMB(champ4,nomb4);

   CASE nomb3 OF
   0,3: ;
   1: adresse := nomb4;
   2,4,5 : BEGIN
      {* recherche contenu de la cellule defini par champ4 *}
      FC_CONTORD('M',nomb4,ordinateur,contenu,ok);
      IF ok = FALSE THEN code_erreur := 4
      ELSE BEGIN
         {* test de validite du contenu *}
         IF FC_TESTNUM(contenu,2) = FALSE
         THEN code_erreur := 4
         ELSE BEGIN
            j := 1;
            WHILE contenu[j] = blan DO j := j + 1;
            WHILE contenu[j] = '0' DO j := j + 1;
            FC_LECTSYMB(contenu,contenu,long,j);
            IF long > 2 THEN code_erreur := 4
            ELSE BEGIN
               FC_CARNOMB(contenu,adresse);
               IF nomb3 = 4
               THEN
               BEGIN
               adresse := adresse - 1;
               IF adresse < 0 THEN
               code_erreur := 4;
               FC_NOMBCAR(adresse,contenu);
         FC_GMOT(contenu,nomb4,ordinateur,ok,OUTPUT)
                  END
                    END
                END
          END
      END
   END {* case *}
END;

PROCEDURE SEMANT_INSTRUCTION;
```

```
VAR
   champ12,valeur,symbole,mode_adressage : chaincaract;
   pos_tab,j : INTEGER;
   pos_d_lect,longsymb : loncha;
   champ3 : CHAR;

BEGIN
   FOR j := 1 TO 2 DO champ12[j] := reg_i[j + 1];
   champ12[3] := fd1;
   champ3 := reg_i[4];
   valeur[1] := champ3;
   valeur[2] := fd1;
   FC_CARNOMB(valeur,nomb3);
   FC_LOCALSYMB(champ12,1,pos_tab,fcopma);
   IF pos_tab = 0
   THEN code_erreur := 3
   ELSE BEGIN
      FC_CARNOMB(champ12,nomb12);
      {* test du mode d'adressage *}
      FC_VALASS(pos_tab,valeur,fcopma);
      pos_d_lect := 1;
      {* le mode adressage se trouve dans le 2eme element de val
      FC_LECTSYMB(valeur,symbole,longsymb,pos_d_lect);
      FC_LECTSYMB(valeur,mode_adressage,longsymb,pos_d_lect);
      {* recherche du champ3 dans les modes adressages possibles
      mode_adressage[longsymb + 1] := champ3;
      j := 1;
      WHILE mode_adressage[j] <> champ3 DO
               j := j + 1;
      IF j = (longsymb + 1)   {* cas modes adr. de RI introuvabl
      THEN code_erreur := 2;
      {* affichage du symbole COP ,il est le 3eme element dans *
      {* dans la valeur *}
      FC_LECTSYMB(valeur,symbole,longsymb,pos_d_lect);
      FC_GCOP(symbole,ordinateur,OUTPUT);
      END
END;
{*-------------------------------------------------*}

PROCEDURE INITIALISATION;

BEGIN
   nophase := 1;
   code_erreur := 0
END;


PROCEDURE TRAITEMENT;

BEGIN
   WHILE code_erreur = 0 DO
      BEGIN
      CASE nophase OF
      1: REC_INSTRUCTION;
```

```
        2: SYNT_INSTRUCTION;
        3: SEMANT_INSTRUCTION;
        4: REC_ADOPER;
        5: EXEC_INSTRUCTION;
        6: code_erreur := 99
        END;
        nophase := nophase + 1;
        END
END;

PROCEDURE TERMINAISON;


BEGIN
   ok := TRUE;

   {* test debordement de la memoire *}
   IF ((code_erreur = 99) AND (ok_increm = FALSE) AND (nomb12 <>
   THEN code_erreur := 6;
   IF code_erreur <> 99
   THEN BEGIN
      ok := FALSE;
      FC_ERRMON(code_erreur, fmesout, OUTPUT)
        END
END;

{*-------------------------------------------------*}

BEGIN
   INITIALISATION;
   TRAITEMENT;
   TERMINAISON
END;
```

```
#
CONST
# include "fmescst"
# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fmestype"
# include "fcartype"
# include "fordtype"

chaincop = PACKED ARRAY [1..lcop] OF CHAR;
{*************************************************
*                                                *
* ENTREE : le nombre d'instrucion a executer     *
*                                                *
* SORTIE : l'affichage du resultat de l'execu-   *
*              tion de ces instructions          *
*                                                *
* FONCTION : cette procedure execute les instruc-*
*              tion a partir de l'adresse        *
*              contenue dans le compteur         *
*              ordinal jusqu'a la rencontre      *
*              de l'instruction HALT ou apres    *
*              avoir execute le nombre d'instr.  *
*              demande ou lors de la rencontre   *
*              d'une erreur.                     *
*                                                *
*************************************************}


{*************************************}
{ execution d'une instruction machine }
{ de l'ordinateur pedagogique         }

PROCEDURE FC_EXINST(VAR ordinateur : ordinat_type;
          VAR ok : BOOLEAN;
          VAR fsortie,fmessage,fcopmachine : TEXT);EXTERN;

{***************************}
{ attente d'un lap de temps }

PROCEDURE FC_ATTENTE(secondes : INTEGER);
EXTERN;

{*******************************************}
{ transformation d'une chaine de caracteres }
{ numeriques en un nombre entier            }

PROCEDURE FC_CARNOMB(repres : chaincaract;
                     VAR nombre : INTEGER);
EXTERN;

{-------------------------------------------------}
```

```
PROCEDURE FC_EXPRG(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                    VAR OUTPUT,fmessage,fcopma : TEXT
                  );

VAR
       nbre_inst : INTEGER;
       fin_prg : chaincop;

PROCEDURE INITIALISATION;

VAR
       j : loncha;
       zontrav : chaincaract;

BEGIN
       IF message.param1_long > 0
       THEN
       BEGIN
       FOR j := 1 TO message.param1_long
              DO zontrav[j] := message.param1[j];
       zontrav[message.param1_long + 1] := fdl;
       FC_CARNOMB(zontrav,nbre_inst);
       END
       ELSE nbre_inst := 99;
       fin_prg[lcop] := 'T';
       fin_prg[lcop - 1] := 'L';
       fin_prg[lcop - 2] := 'H';
       FOR j := 1 TO (lcop - 3) DO fin_prg[j] := blan
END;

PROCEDURE TRAITEMENT;

VAR
       ok : BOOLEAN;
       no_inst : INTEGER;

BEGIN
       no_inst := 1;
       REPEAT
              FC_EXINST(ordinateur,ok,OUTPUT,fmessage,fcopma);
              IF ordinateur.vi > 0
                     THEN FC_ATTENTE(ordinateur.vi);
              no_inst := no_inst + 1
       UNTIL ((no_inst) > nbre_inst) OR
              ( ok = FALSE) OR
              ( ordinateur.cop = fin_prg)
END;

BEGIN
       INITIALISATION;
       TRAITEMENT
END;
```

```
#
CONST
# include "fcarcst"
# include "fmescst"
# include "fgrrcpt"
TYPE
# include "fcartype"
# include "fmestype"
# include "fordtype"

{*************************************************************
*                                                            *
* ENTREE :  le contenu de l'ordinateur pedagogique,          *
*           le fichier des messages d'erreurs,               *
*           la commande de l'utilisateur.                    *
*                                                            *
* SORTIE :  un fichier contenant le contenu de la            *
*              memoire de l'ordinateur pedagogique           *
*           ou un message d'erreur si la copie n'a pu        *
*              etre realise.                                 *
*                                                            *
* FONCTION : cette procedure copie le contenu de la          *
*            memoire de l'ordinateur pedagogique a           *
*            partir de l'adresse d'une cellule donnee        *
*            jusqu'a l'adresse d'une autre cellule donne*
*            la copie est realise suivant un nom de          *
*            fichier introduit par l'utilisateur.            *
*                                                            *
*************************************************************}

{*******************************************}
{ transformation d'une chaine de caracteres }
{ numeriques en un nombre entier            }

PROCEDURE FC_CARNOMB(repres : chaincaract;
                     VAR nombre : INTEGER);
EXTERN;

{*******************************************}
{ recherche d'une information dans une zone }
{ de l'ordinateur pedagogique               }

PROCEDURE FC_CONTORD(nomzone : CHAR;
                     nocellule : INTEGER;
                     VAR ordinateur : ordinat_type;
                     VAR contenu : chaincaract;
                     VAR ok : BOOLEAN);EXTERN;

{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                         }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;
```

```
{****************************************}
{ ecriture du contenu d'un fichier dans }
{ un fichier externe }

PROCEDURE FECRITURE(VAR nomfichier : chaincaract;
                    VAR ok : BOOLEAN;
                    VAR fichier : TEXT);EXTERN;


{*********************************}
{ garnissage de la zone d'erreur }
{ du moniteur }

PROCEDURE FC_ERRMON(code_erreur : INTEGER;
                    VAR fmessage,OUTPUT : TEXT);EXTERN;

{-------------------------------------------------------------------}

PROCEDURE FC_SAUVMEM(VAR message : message_cmd;
                     VAR ordinateur : ordinat_type;
                     VAR OUTPUT,fmessage : TEXT);

VAR
     adr_debut,adr_fin : INTEGER;
     nomfichier : chaincaract;
     fichier : TEXT;

PROCEDURE INITIALISATION;

VAR
       j : INTEGER;
       debut,fin : chaincaract;

BEGIN
       WITH message DO
       BEGIN
       { remplissage du nom de fichier }
       FOR j := 1 TO param1_long DO
              nomfichier[j] := param1[j];
       nomfichier[param1_long + 1] := fdl;

       { adresse debut de la memoire a consideree }
       FOR j := 1 TO param2_long DO
              debut[j] := param2[j];
       debut[param2_long + 1] := fdl;
       FC_CARNOMB(debut,adr_debut);

       { adresse fin de la memoire a consideree }
       FOR j := 1 TO param3_long DO
              fin[j] := param3[j];
       fin[param3_long + 1] := fdl;
       FC_CARNOMB(fin,adr_fin)
```

```
        END
END;

PROCEDURE TRAITEMENT;

VAR
        ok : BOOLEAN;
        contenu : chaincaract;
        j : INTEGER;

BEGIN

        REWRITE(fichier);
        FOR j := adr_debut TO adr_fin DO
        BEGIN
                FC_CONTORD('M', j, ordinateur, contenu, ok);
                FC_ECRLIGNE(contenu, lmot, fichier)
        END
END;

PROCEDURE CLOTURE;

VAR
        ok : BOOLEAN;

BEGIN
        FECRITURE(nomfichier, ok, fichier);
        IF ok = FALSE
                THEN FC_ERRMON(2, fmessage, OUTPUT)
END;


BEGIN
        INITIALISATION;
        TRAITEMENT;
        CLOTURE
END;
```

```
#
CONST
# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fordtype"
# include "fcartype"
# include "fgrtype"

{************************************************}
{ positionnement du curseur sur l'ecran          }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                         }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;

{-------------------------------------------------------}
{ ENTREE : etat de l'imprimante, ce dernier est   }
{              contenu dans la description de     }
{              l'ordinateur pedagogique           }
{                                                 }
{ SORTIE : chaque ligne avance d'une position sur }
{         l'imprimante, la premiere ligne est     }
{         de caractere blanc                      }
{                                                 }
{ FONCTION : shift de chaque ligne de l'imprimante}
{           d'une ligne (vers le haut) et ecriture }
{           d'une ligne blanche                   }
{-------------------------------------------------------}

PROCEDURE FC_AVANCPAP (VAR ordinateur : ordinat_type;
                       VAR OUTPUT : TEXT);

VAR
       debut_ligne,i,j : INTEGER;
       ligne_remplie : chaincaract;

BEGIN
       debut_ligne := ligimp1;

       FOR i := (nlig - 1) DOWNTO 1
       DO
       BEGIN
           ordinateur.lignes[i + 1] := ordinateur.lignes[i];
          FOR j := 1 TO llig
          DO ligne_remplie[j] := ordinateur.lignes[i,j];
```

```
            FC_POSITCURS(debut_ligne,colimp1);
            FC_ECRLIGNE(ligne_remplie,llig,OUTPUT);
            debut_ligne := debut_ligne + hautlig + 1
         END;

         FOR j := 1 TO llig DO ordinateur.lignes[1,j] := blan;
         FOR j := 1 TO llig DO ligne_remplie[j] := blan;
            FC_POSITCURS(debut_ligne,colimp1);
            FC_ECRLIGNE(ligne_remplie,llig,OUTPUT)
END;
```

```
#
CONST
# include "fcarcst"
# include "fmescst"
# include "fgrrcpt"
TYPE
# include "fcartype"
# include "fmestype"
# include "fordtype"

{*************************************************************
*                                                            *
* ENTREE :  le contenu de l'ordinateur pedagogique,          *
*          le fichier des messages d'erreurs,                *
*          la commande de l'utilisateur.                     *
*                                                            *
* SORTIE :  un fichier contenant le contenu de               *
*           de l'imprimante                                  *
*          ou un message d'erreur si la copie n'a pu         *
*              etre realise.                                 *
*                                                            *
* FONCTION : cette procedure copie le contenu de             *
*           l'imprimante de l'ordinateur pedagogique a       *
*           partir de l'adresse d'une cellule donnee         *
*           jusqu'a l'adresse d'une autre cellule donne*
*           la copie est realise suivant un nom de           *
*           fichier introduit par l'utilisateur.             *
*                                                            *
*************************************************************}


{*******************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                          }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


{*****************************************}
{ ecriture du contenu d'un fichier dans }
{ un fichier externe }


PROCEDURE FECRITURE(VAR nomfichier : chaincaract;
                    VAR ok : BOOLEAN;
                    VAR fichier : TEXT);EXTERN;


{*********************************}
{ garnissage de la zone d'erreur }
{ du moniteur }
```

```
PROCEDURE FC_ERRMON(code_erreur : INTEGER;
                    VAR fmessage,OUTPUT : TEXT);EXTERN;

{-------------------------------------------------------------}

PROCEDURE FC_SAUVLIST(VAR message : message_cmd;
                      VAR ordinateur : ordinat_type;
                      VAR OUTPUT,fmessage : TEXT);

VAR
     nomfichier : chaincaract;
     fichier : TEXT;

PROCEDURE INITIALISATION;

VAR
       j : INTEGER;

BEGIN
       WITH message DO
       BEGIN
       { remplissage du nom de fichier }
       FOR j := 1 TO param1_long DO
              nomfichier[j] := param1[j];
       nomfichier[param1_long + 1] := fdl;

       END
END;

PROCEDURE TRAITEMENT;

VAR
       contenu : chaincaract;
       i,j : INTEGER;

BEGIN

       REWRITE(fichier);
       FOR i := 1 TO nlig DO
       BEGIN
              FOR j := 1 TO llig DO
              contenu[j] := ordinateur.lignes[i,j];
              FC_ECRLIGNE(contenu,llig,fichier)
       END
END;

PROCEDURE CLOTURE;

VAR
       ok : BOOLEAN;

BEGIN
       FECRITURE(nomfichier,ok,fichier);
       IF ok = FALSE
              THEN FC_ERRMON(2,fmessage,OUTPUT)
```

```
END;


BEGIN
        INITIALISATION;
        TRAITEMENT;
        CLOTURE
END;
```

```
#
CONST
# include 'fgrcst'

TYPE
# include 'fgrtype'


{************************************************}
{ positionnement du curseur sur l'ecran                }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;
{***************************}
{ attente d'un lap de temps }

PROCEDURE FC_ATTENTE(secondes : INTEGER);
EXTERN;


PROCEDURE FC_HELP(VAR OUTPUT : TEXT);


BEGIN

        FC_POSITCURS(8,30);
        WRITELN(OUTPUT,' MEMOIRE CENTRALE ');
        FC_POSITCURS(14,3);
        WRITELN(OUTPUT,' LECTEUR DE CARTES');
        FC_POSITCURS(15,27);
        WRITELN(OUTPUT,'CO');
        FC_POSITCURS(15,35);
        WRITELN(OUTPUT,'RI');
        FC_POSITCURS(15,44);
        WRITELN(OUTPUT,'COP');
        FC_POSITCURS(18,29);
        WRITELN(OUTPUT,'RE');
        FC_POSITCURS(18,38);
        WRITELN(OUTPUT,'A ');
        FC_POSITCURS(18,45);
        WRITELN(OUTPUT,'IL');
        FC_POSITCURS(22,29);
        WRITELN(OUTPUT,'ZONE D''AFFICHAGE');
        FC_POSITCURS(14,55);
        WRITELN(OUTPUT,' IMPRIMANTE ');
        FC_ATTENTE(10);
        FC_POSITCURS(8,30);
        WRITELN(OUTPUT,'                    ');
        FC_POSITCURS(14,3);
        WRITELN(OUTPUT,'                      ');
        FC_POSITCURS(15,27);
        WRITELN(OUTPUT,'  ');
        FC_POSITCURS(15,35);
```

```
        WRITELN(OUTPUT,'   ');
        FC_POSITCURS(15,44);
        WRITELN(OUTPUT,'    ');
        FC_POSITCURS(18,29);
        WRITELN(OUTPUT,'   ');
        FC_POSITCURS(18,38);
        WRITELN(OUTPUT,'   ');
        FC_POSITCURS(18,45);
        WRITELN(OUTPUT,'   ');
        FC_POSITCURS(22,29);
        WRITELN(OUTPUT,'                         ');
        FC_POSITCURS(14,55);
        WRITELN(OUTPUT,'                ')
END;
```

```
#
{*-------------------------------------------------------------*}
{*                                                             *}
{*      CHARGEMENT DU LECTEUR DE CARTES                        *}
{*                                                             *}
{*-------------------------------------------------------------*}

CONST
# include "fcarcst"
# include "fmescst"
# include "fgrrcpt"

TYPE
# include "fcartype"
# include "fmestype"
# include "fordtype"

{*********************************}
{ lecture du contenu d'un fichier }
{ suivant un nom de fichier donne }

PROCEDURE FLECTURE(VAR nomfichier : chaincaract;
                   VAR ok : BOOLEAN;
                   VAR fichier : TEXT);EXTERN;


{*****************************************}
{ lecture d'une chaine de caracteres dans }
{      un fichier                         }

PROCEDURE FC_LECTLIGNE(VAR zonlect : chaincaract;
                       long_lire : lonlig;
                       VAR fentree : TEXT);EXTERN;

{*********************************}
{ garnissage de la zone d'erreur }
{ du moniteur }

PROCEDURE FC_ERRMON(code_erreur : INTEGER;
                    VAR fmessage,OUTPUT : TEXT);EXTERN;


{*****************************************}
{ cette procedure affiche l'environnement }
{ de la carte courante a l'ecran }

PROCEDURE FC_AFFCARTE(VAR ordinateur : ordinat_type;
                      VAR OUTPUT : TEXT);
EXTERN;


PROCEDURE FC_CHLECT(
                      VAR message : message_cmd;
                      VAR ordinateur : ordinat_type;
                      VAR fmesout,OUTPUT : TEXT
```

```
                    );

{*************************************************************
*                                                            *
* ENTREE : l'identification du fichier a garnir dans la *
*          la memoire de l'ordinateur pedagogique;       *
*                                                            *
* SORTIE : le remplissage et l'affichage des zones     *
*          garnies du lecteur de cartes ou l'envoit     *
*          d'un message d'erreur a l'utilisateur        *
*                                                            *
* FONCTION :   le contenu du fichier est charge dans la *
*            dans le lecteur de cartes, la 1ere carte  *
*            du fichier est la carte courante. *
*                                                            *
*************************************************************}

VAR
        code_erreur : INTEGER;
        fichier : TEXT;
        zonlect : chaincaract;
        cartelu : carte;
        deb_lect : loncha;


PROCEDURE OBTCARTE(VAR succes : BOOLEAN);

VAR
        longlu : loncha;

BEGIN
        succes := TRUE;
        longlu := 0;
        WHILE  (longlu <> lcart) DO
        BEGIN
        WHILE (zonlect[deb_lect] <> fdl) AND (longlu <> lcart)
        DO
        BEGIN
                longlu := longlu + 1;
                cartelu[longlu] := zonlect[deb_lect];
                deb_lect := deb_lect + 1
        END;
        IF longlu <> lcart
        THEN
        BEGIN
                IF NOT EOF(fichier)
                THEN FC_LECTLIGNE(zonlect,lmaxlig,fichier)
                ELSE
                   BEGIN
                        succes := FALSE;
                        longlu := lcart
                   END;
                deb_lect := 1
        END
        END;
```

```
END;

PROCEDURE INITIALISATION;

VAR
        j : loncha;
        nomfichier : chaincaract;
        ok : BOOLEAN;

BEGIN

        code_erreur := 0;
        WITH message DO
        BEGIN
                FOR j := 1 TO param1_long DO
                        nomfichier[j] := param1[j];
                nomfichier[param1_long + 1] := fd1;
        END;
        FLECTURE(nomfichier,ok,fichier);
        IF NOT ok THEN code_erreur := 2
END;

PROCEDURE TRAITEMENT;

VAR
        succes : BOOLEAN;
         nbcarte_lu : INTEGER;

BEGIN
        RESET(fichier);
        FC_LECTLIGNE(zonlect,lmaxlig,fichier);
        deb_lect := 1;

         ordinateur.no_carte := 1;
         nbcarte_lu := 0;

        OBTCARTE(succes);
        WHILE succes AND (nbcarte_lu < 24)
        DO
        BEGIN
                nbcarte_lu := nbcarte_lu + 1;
                ordinateur.cartes[nbcarte_lu] := cartelu;
                OBTCARTE(succes)
        END;

        IF NOT succes THEN ordinateur.cartes[nbcarte_lu + 1,1] :=
        IF succes AND (nbcarte_lu = 24) THEN code_erreur := 2;
END;

PROCEDURE CLOTURE;

BEGIN

        IF code_erreur <> 0
        THEN FC_ERRMON(code_erreur,fmesout,OUTPUT)
```

```
        ELSE FC_AFFCARTE(ordinateur,OUTPUT)
END;

{-------------------------------------------------------------------------------}

BEGIN
        INITIALISATION;
        IF code_erreur = 0 THEN TRAITEMENT;
        CLOTURE
END;
```

```
#
CONST
# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fordtype"

{-------------------------------------------------}
{ ENTREE : l'etat du lecteur de cartes, celui}
{         -ci est contenu dans l'ordinateur }
{          pedagogique                        }
{                                             }
{ SORTIE : affichage eventuel d'une nouvelle }
{         carte ou envoit d'un message       }
{          d'erreur                           }
{                                             }
{ FONCTION : cette procedure rend courant la }
{           carte qui precede la carte       }
{           courante, si elle n'existe pas   }
{           alors il y a envoit d'un         }
{           message d'erreur                  }
{-------------------------------------------------}


{*********************************}
{ garnissage de la zone d'erreur }
{ du moniteur }

PROCEDURE FC_ERRMON(code_erreur : INTEGER;
                    VAR fmessage,OUTPUT : TEXT);EXTERN;

{*****************************************}
{ cette procedure affiche l'environnement }
{ de la carte courante a l'ecran }

PROCEDURE FC_AFFCARTE(VAR ordinateur : ordinat_type;
                      VAR OUTPUT : TEXT);
EXTERN;


{-------------------------------------------------------------}

PROCEDURE FC_CARTPREC(VAR ordinateur : ordinat_type;
                     VAR fmesout,OUTPUT : TEXT);

BEGIN
       WITH ordinateur DO
       BEGIN
              IF no_carte <= 1
              THEN FC_ERRMON(2,fmesout,OUTPUT)
              ELSE
              BEGIN
                     no_carte := no_carte - 1;
                     col_cart := 1;
```

```
                                FC_AFFCARTE(ordinateur,OUTPUT)
                        END
                END
        END;
```

```
#
CONST
# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fordtype"

{-----------------------------------------------}
{ ENTREE : l'etat du lecteur de cartes, celui}
{         -ci est contenu dans l'ordinateur }
{          pedagogique                       }
{                                            }
{ SORTIE : affichage eventuel d'une nouvelle }
{         carte ou envoit d'un message       }
{          d'erreur                          }
{                                            }
{ FONCTION : cette procedure rend courant la }
{            carte qui suit la carte      }
{            courante, si elle n'existe pas  }
{            alors il y a envoit d'un        }
{            message d'erreur                }
{-----------------------------------------------}


{*********************************}
{ garnissage de la zone d'erreur }
{ du moniteur }

PROCEDURE FC_ERRMON(code_erreur : INTEGER;
                    VAR fmessage,OUTPUT : TEXT);EXTERN;

{*****************************************}
{ cette procedure affiche l'environnement }
{ de la carte courante a l'ecran }

PROCEDURE FC_AFFCARTE(VAR ordinateur : ordinat_type;
                      VAR OUTPUT : TEXT);
EXTERN;


{-------------------------------------------------------------}

PROCEDURE FC_CARTSUIV(VAR ordinateur : ordinat_type;
                      VAR fmesout,OUTPUT : TEXT);

BEGIN
        WITH ordinateur DO
        BEGIN
                IF cartes[no_carte + 1,1] = fdl
                THEN FC_ERRMON(2,fmesout,OUTPUT)
                ELSE
                BEGIN
                        no_carte := no_carte + 1;
                        col_cart := 1;
```

```
                                    FC_AFFCARTE(ordinateur,OUTPUT)
                            END
                    END
            END;
```

```
#
CONST
# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fordtype"

{-------------------------------------------------}
{ ENTREE : l'etat du lecteur de cartes, celui}
{         -ci est contenu dans l'ordinateur }
{          pedagogique                       }
{                                            }
{ SORTIE : affichage eventuel d'une nouvelle }
{         carte ou envoit d'un message       }
{          d'erreur                          }
{                                            }
{ FONCTION : cette procedure rend courant la }
{            premiere carte du paquet, si    }
{            elle n'existe pas alors il y a  }
{            envoit d'un message d'erreur    }
{-------------------------------------------------}


{*********************************}
{ garnissage de la zone d'erreur }
{ du moniteur }

PROCEDURE FC_ERRMON(code_erreur : INTEGER;
                    VAR fmessage,OUTPUT : TEXT);EXTERN;

{*****************************************}
{ cette procedure affiche l'environnement }
{ de la carte courante a l'ecran }

PROCEDURE FC_AFFCARTE(VAR ordinateur : ordinat_type;
                      VAR OUTPUT : TEXT);
EXTERN;


{-------------------------------------------------------------}

PROCEDURE FC_CARTUN(VAR ordinateur : ordinat_type;
                    VAR fmesout,OUTPUT : TEXT);

BEGIN
       WITH ordinateur DO
       BEGIN
              IF no_carte <= 1
              THEN FC_ERRMON(2,fmesout,OUTPUT)
              ELSE
              BEGIN
                     no_carte := 1;
                     col_cart := 1;
                     FC_AFFCARTE(ordinateur,OUTPUT)
```

```
                    END
                END
            END;
```

```
#

{*-----------------------------------------*}
{*                                         *}
{*   Initialisation du lecteur de          *}
{*                cartes                   *}
{*                                         *}
{*-----------------------------------------*}

CONST
# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fordtype"
# include "fcartype"
# include "fgrtype"

{*************************************************}
{ positionnement du curseur sur l'ecran           }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;
{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                        }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


{*************************************}
{ initialisation du lecteur de cartes }

PROCEDURE FC_CARTINIT( VAR ordinateur : ordinat_type;
                       VAR OUTPUT : TEXT);


VAR
  init_carte : chaincaract;
  i : INTEGER;
  debut_carte : INTEGER;

BEGIN
  debut_carte := ligcart1;
  ordinateur.no_carte := 0;
  ordinateur.cartes[1,1] := fd1;
  FOR i := 1 TO lcart
    DO init_cart[i] := blan;
  FOR i := 1 TO ncart
  DO
  BEGIN
    FC_POSITCURS(debut_carte,colcart1);
```

```
      FC_ECRLIGNE(init_carte,lcart,OUTPUT);
      debut_carte := debut_carte + hautcart + 1
    END
  END;
```

PROGRAMMES DU NIVEAU 5

```
#
CONST
# include "fgrrcpt"
# include "fcarcst"
TYPE
# include "fordtype"
# include "fcartype"
# include "fgrtype"

(*-------------------------------------------------*)
(*              TRANSFERT CONTENU MOT --->         *)
(*   ZONE MEMOIRE DE L'ORDINATEUR PEDAGOGIQUE      *)
(*-------------------------------------------------*)

{************************************************}
{ transfert du contenu d'une chaine de caracteres  }
{ dans une autre chaine de caracteres avec remplis-}
{ sages des positions non occupees }

PROCEDURE FC_TRANSFC(VAR emetteur,recepteur : chaincaract;
                     long_emet,lon_recept : loncha;
                     test_signe : BOOLEAN;
                     caract_remplissage : CHAR);EXTERN;


{*******************************************}
{ lecture d'un symbole dans une chaine de }
{ caracteres   }

PROCEDURE FC_LECTSYMB(VAR chaine,symbole : chaincaract;
                      VAR longsymb,pos_d_lect : loncha);
EXTERN;


{*******************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                          }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


{************************************************}
{ positionnement du curseur sur l'ecran     }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{-------------------------------------------------
* ENTREE : - une chaine de caracteres contenant *
*            suite de caracteres numeriques     *
*            representant la valeur du mot.     *
*            cette valeur occupe les premieres  *
```

```
*            positions de la chaine et est suivi   *
*          d'un caractere special (sentinelle)     *
*        - numero de la cellule qui doit contenir  *
*          le mot recu                             *
*                                                  *
* SORTIE : - une valeur booleenne signalant si     *
*          garnissage  s'est bien deroule          *
*        - si pas d'erreur affichage et            *
*          remplissage du contenu du mot dans      *
*          dans la cellule concernee;              *
*                                                  *
* FONCTION : - transformation de   la position de  *
*              la cellule en coordonnees lignes    *
*              et colonne de la memoire centrale   *
*            - si les valeurs de lignes et colonnes *
*              appartiennent aux bornes de la      *
*              memoire centrale alors affichage    *
*              et remplissage de la cellule        *
*              appropri'e de l'ordinateur          *
*              pedagogique                         *
*                                                  *
----------------------------------------------------}

PROCEDURE FC_GMOT(VAR val_mot : chaincaract;
        (* chaine de caracteres contenant le mot        *)
                  no_mot : INTEGER;
        (* numero de la cellule memoire                 *)
                  VAR ordinateur : ordinat_type;
        (* representation interne de l'ordinateur       *)
        (* pedagogique *)
                  VAR ok : BOOLEAN;
        (* resultat du bon fonctionnement               *)
                  VAR OUTPUT : TEXT
        (* ecran terminal                               *)
                 );

VAR
        j,pos_d_lect,
        longmot
        (* longueur du mot contenu dans la chaine de    *)
        (* caracteres recues    *)
                 : loncha;
        zonimp : chaincaract;
        lig_mem,col_mem : INTEGER;
        lig_gr : nolig_ecran;
        col_gr : nocol_ecran;

(*-------------------------------------------------------------*)
BEGIN
   ok := FALSE;

   (* CALCUL DES COORDONNEES *)

   lig_mem := ( no_mot DIV nmotcolmem) + 1;
   col_mem := (no_mot MOD nmotcolmem) + 1;
```

```
    IF (lig_mem IN [1..nmotligmem]) AND
       (col_mem IN [1..nmotcolmem])
    THEN BEGIN
       ok := TRUE;

       (* REMPLISSAGE *)

       pos_d_lect := 1;
       (* calcul longueur du mot recu *)
       FC_LECTSYMB(val_mot,val_mot,longmot,pos_d_lect);
       (* representation formalisee du mot *)
       FC_TRANSFC(val_mot,zonimp,longmot,lmot,TRUE,zero);
       (* remplisage du mot *)
       FOR j := 1 TO lmot DO
          ordinateur.memoire[lig_mem,col_mem,j] := zonimp[j];

       (* affichage *)

       lig_gr := ligmot1 + (lig_mem - 1) * hautmot;
       col_gr := colmot1 + (col_mem - 1) * (lmot + distmot);
       FC_POSITCURS(lig_gr,col_gr);
       FC_ECRLIGNE(zonimp,lmot,OUTPUT)
    END
END;
```

```
#
CONST
# include "fcarcst"
# include "fmescst"
# include "fgrrcpt"
TYPE
# include "fordtype"
# include "fcartype"
# include "fgrtype"
# include "fmestype"


{-------------------------------------------}
{ INCREMENTATION DU CONTENU DU COMPTEUR }
{      D'INSTRUCTIONS (CO)                  }
{-------------------------------------------}



{*******************************************}
{ transformation d'un nombre entier positif }
{ en une chaine de caracteres                }

PROCEDURE FC_NOMBCAR(nombre : INTEGER;
                     VAR repres : chaincaract);
EXTERN;

{*****************************************}
{ garnissage et affichage de la zone     }
{ registre C de l'ordinateur pedagogique }

PROCEDURE FC_GREGC(VAR message : message_cmd;
                   VAR ordinateur : ordinat_type;
                   VAR fsortie : TEXT);
EXTERN;
{*******************************************}
{ transformation d'une chaine de caracteres }
{ numeriques en un nombre entier            }

PROCEDURE FC_CARNOMB(repres : chaincaract;
                     VAR nombre : INTEGER);
EXTERN;

{*****************************************}
{ lecture d'un symbole dans une chaine de }
{ caracteres   }

PROCEDURE FC_LECTSYMB(VAR chaine,symbole : chaincaract;
                      VAR longsymb,pos_d_lect : loncha);
EXTERN;


{-------------------------------------------------------------------------
* ENTREE  :  - representation interne de l'ordinateur pedagogiqu
*         contenant le contenu du compteur ordinal sous forme   *
```

```
*        caracteres;                                                    *
*                                                                       *
* SORTIE :   - une valeur booleenne signalant si l'incrementation      *
*                a se faire correctement                                *
*          - si pas erreur alors :                                      *
*         + renvoit de la valeur (sous forme entiere) contenue          *
*           dans le compteur d'instructions avant l'incremen-           *
*           tation,                                                     *
*         + remplissage et affichage de la nouvelle valeur du           *
*           compteur d'instructions;                                    *
* FONCTION :- si valeur contenu dans CO (compteur ordinal)              *
*             appartient a [0..nombre maximun de cellules] alors        *
*                CO <--- CO + 1.                                        *
-------------------------------------------------------------------------

PROCEDURE FC_INCREMCO(VAR ordinateur : ordinat_type;
       (* representation interne de l'ordinateur pedagogique *)
        VAR anc_co : INTEGER;   (* ancienne valeur du CO *)
         VAR ok : BOOLEAN;      (* resultat du fonctionnement *)
         VAR OUTPUT : TEXT      (* ecran terminal *)
              );

VAR
  j,pos_d_lect,longsymb : loncha;
  lig_trav      : chaincaract;
  nouvel_co     : INTEGER;
  message               : message_cmd;

(*-----------------------------------------------------------------------*)
BEGIN
  ok := FALSE;

  (* transformation de la valeur contenu dans CO en un  *)
  (* nombre entier *)

  FOR j := 1 TO lco DO lig_trav[j] := ordinateur.co[j];
  lig_trav[lco + 1] := fdl;
  FC_CARNOMB(lig_trav,anc_co);

  IF anc_co < ((nmotligmem * nmotcolmem ) - 1)
  THEN BEGIN
    ok := TRUE;
    (* incrementation *)
    nouvel_co := anc_co + 1;
    FC_NOMBCAR(nouvel_co,lig_trav);

    (* affichage et remplissage de la nouvelle valeur  *)
    (* du CO on utilise la fonction *)
    (*   existante garnissage du registre C *)

    pos_d_lect := 1;
    FC_LECTSYMB(lig_trav,lig_trav,longsymb,pos_d_lect);
    FOR j := 1 TO longsymb DO message.param1[j] := lig_trav[j];
    message.param1_long := longsymb;
    FC_GREGC(message,ordinateur,OUTPUT)
```

```
#
(*-------------------------------------------------------------------*)
(*                                                                   *)
(*  RECHERCHE D'UNE INFORMATION DANS UNE ZONE DE                     *)
(*     L'ORDINATEUR    PEDAGOGIQUE                                   *)
(*                                                                   *)
(*-------------------------------------------------------------------*)

CONST
# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fcartype"
# include "fordtype"

PROCEDURE FC_CONTORD(nomzone : CHAR;
        (* A = Accumulateur, E = registre E, I = registre
        Instruction, L = indicateur Logique, M = Memoire,
        O = compteur Ordinal,C = carte courante *)
             nocellule : INTEGER;
        (* si on desire le contenu d'un mot de la memoire *)
             VAR ordinateur : ordinat_type;
        (* representation interne de l'ordinateur pedag. *)
             VAR contenu : chaincaract;
        (* contenu de la zone concernee *)
             VAR ok : BOOLEAN
        (* variable resultat du bon foncionnement de
           l'operation *)
                    );

{-------------------------------------------------------------------
* ENTREE : -nom de la zone identifiant la donnee a rechercher,     *
*          - le numero de la cellule memoire si nom de la zone     *
*            = memoire;                                            *
*          - la representation interne de l'ordinateur             *
*              pedagogique                                         *
*                                                                  *
* SORTIE : - une variable signalant la reussite de l'operation,    *
*          - le contenu de la zone consideree si operation est     *
*              OK                                                  *
*                                                                  *
* FONCTION : - recherche et rangement de l'information recherche
*            - detection de la validite de la demande dans le cas
*              ou la zone n'existe pas et dans le cas ou le        *
*              no de cellule n'existe pas (si nomzone = memoire);
-------------------------------------------------------------------}

VAR
        j,col_mem,lig_mem : INTEGER;

BEGIN
  ok := TRUE;

  IF NOT (nomzone IN ['A','E','I','L','M','O','C'])
```

```
    THEN ok := FALSE
    ELSE
      CASE nomzone OF

      'M' :BEGIN
        lig_mem := (nocellule DIV nmotcolmem) + 1;
        col_mem := (nocellule MOD nmotcolmem) + 1;
        IF (lig_mem IN [1..nmotligmem]) AND
            (col_mem IN [1..nmotcolmem])
        THEN BEGIN
          FOR j := 1 TO lmot DO
          contenu[j] := ordinateur.memoire[lig_mem,col_mem,j];
          contenu[lmot + 1] := fdl
             END
        ELSE ok := FALSE
          END;
      'A' :BEGIN
        FOR j := 1 TO lmot DO
          contenu[j] := ordinateur.ra[j];
          contenu[lmot + 1] := fdl
          END;
      'E' :BEGIN
        FOR j := 1 TO lmot DO
          contenu[j] := ordinateur.re[j];
          contenu[lmot + 1] := fdl
          END;
      'I' :BEGIN
        FOR j := 1 TO lmot DO
          contenu[j] := ordinateur.ri[j];
          contenu[lmot + 1] := fdl
          END;
      'L' :BEGIN
        FOR j := 1 TO lil DO
          contenu[j] := ordinateur.il[j];
          contenu[lil + 1] := fdl
          END;
      'O' :BEGIN
        FOR j := 1 TO lco DO
          contenu[j] := ordinateur.co[j];
          contenu[lil + 1] := fdl
          END;
      'C' :BEGIN
          WITH ordinateur DO
          FOR j := 1 TO lcart DO
          contenu[j] := cartes[no_carte,j];
          contenu[lil + 1] := fdl
          END
      END
END;
```

```
#
CONST
# include "fgrrcpt"
# include "fcarcst"

TYPE
# include "fordtype"
# include "fcartype"
# include "fgrtype"

{-------------------------------------}
{ GARNISSAGE DU CODE OPERATOIRE }
{-------------------------------------}


{*************************************************}
{ transfert du contenu d'une chaine de caracteres  }
{ dans une autre chaine de caracteres avec remplis-}
{ sages des positions non occupees }

PROCEDURE FC_TRANSFC(VAR emetteur,recepteur : chaincaract;
                     long_emet,lon_recept : loncha;
                     test_signe : BOOLEAN;
                     caract_remplissage : CHAR);EXTERN;


{*************************************************}
{ positionnement du curseur sur l'ecran                }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{      un fichier.                         }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


(*-------------------------------------------------------*)
PROCEDURE FC_GCOP(
                VAR contenu : chaincaract;
                VAR ordinateur : ordinat_type;
                VAR OUTPUT : TEXT);

VAR j : loncha;
    zonimp : chaincaract;

BEGIN
    j := 1;
    WHILE contenu[j] <> fdl DO j := j + 1;
```

```
    FC_TRANSFC(contenu,zonimp,j - 1,lcop,FALSE,blan);
    FOR j := 1 TO lcop
       DO ordinateur.cop[j] := zonimp[j];
    FC_POSITCURS(ligcop,colcop);
    FC_ECRLIGNE(zonimp,lcop,OUTPUT)
END;
```

```
#
CONST
# include "fcarcst"
# include "fgrrcpt"

TYPE
# include "fcartype"
# include "fordtype"
# include "fgrtype"

{-------------------------------------------------------}
{ ENTREE : description du lecteur de cartes      }
{          contenu dans l'ordinateur pedagog.    }
{                                                }
{ SORTIE : affichage du contenu de la carte      }
{          courante avec les cartes preceden-    }
{          tes ou alors affichage d'un message   }
{          si le lecteur est vide,               }
{                                                }
{ FONCTION : a partir du numero de carte cour-   }
{            ante, la procedure affiche le       }
{            contenu de celle-ci, les autres     }
{            cartes qui le precedaient sont mis  }
{            avant.                              }
{-------------------------------------------------------}


{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{       un fichier.                        }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;

{*************************************************}
{ positionnement du curseur sur l'ecran                  }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;


{-------------------------------------------------------}

PROCEDURE FC_AFFCARTE (VAR ordinateur : ordinat_type;
                       VAR OUTPUT : TEXT);

VAR
        i,j : INTEGER;
        no_cc,no_sp,debut_cartes : INTEGER;
        { no_cc = carte courante;
          no_sp = carte au sommet du paquet;
          debut_cartes = adresse de remplissage de la carte }
        carte_blanche,carte_remplie : chaincaract;
```

```
BEGIN
        no_cc := ordinateur.no_carte;
        no_sp := no_cc;
        WHILE (ordinateur.cartes[no_sp + 1,1] <> fdl)
                AND ((no_sp - no_cc) < (ncart - 1))
        DO no_sp := no_sp + 1;
        debut_cartes := ligcart1;

        IF (no_sp - no_cc) < (ncart - 1)
        THEN
        BEGIN

                { preparation d'une carte vide }
                FOR j := 1 TO lcart DO
                        carte_blanche[j] := blan;

                FOR j := ((no_sp - no_cc) + 2) TO ncart
                DO
                BEGIN
                   FC_POSITCURS(debut_cartes,colcart1);
                   FC_ECRLIGNE(carte_blanche,lcart,OUTPUT);
                   debut_cartes := debut_cartes + hautcart + 1
                END
        END;

        WITH ordinateur DO
        BEGIN
                FOR i := no_sp DOWNTO no_cc
                DO
                BEGIN
                  FOR j := 1 TO lcart
                  DO carte_remplie[j] := cartes[i,j];
                  FC_POSITCURS(debut_cartes,colcart1);
                  FC_ECRLIGNE(carte_remplie,lcart,OUTPUT);
                  debut_cartes := debut_cartes + hautcart + 1
                END
        END
END;
```

```
#
CONST
# include "fgrrcpt"
# include "fcarcst"

TYPE
# include "fgrtype"
# include "fcartype"

{*************************************************************
*                                                            *
* ENTREE : cette procedure recoit :                          *
*              - un code erreur                              *
*              - un fichier contenant les messages           *
*                    d'erreurs.                              *
*                                                            *
* SORTIE : affiche le message d'erreur dans la zone          *
*             d'affichage de l'ordinateur pedagogique        *
*                                                            *
* FONCTION : a partir du code erreur, la procedure va        *
*             rechercher le message d'erreur corres-         *
*             pondant et ensuite l'afficher a l'ecran        *
*                                                            *
*************************************************************}

{*****************************************}
{ recherche d'un enregistrement suivant }
{ sa position dans un fichier           }

PROCEDURE FC_VALASS(pos_enregist : INTEGER;
                    VAR enregistrement : chaincaract;
                    VAR fichier : TEXT);
EXTERN;


{*****************************************************}
{ transfert du contenu d'une chaine de caracteres  }
{ dans une autre chaine de caracteres avec remplis-}
{ sages des positions non occupees }

PROCEDURE FC_TRANSFC(VAR emetteur,recepteur : chaincaract;
                     long_emet,lon_recept : loncha;
                     test_signe : BOOLEAN;
                     caract_remplissage : CHAR);EXTERN;

{*****************************************}
{ lecture d'un symbole dans une chaine de }
{ caracteres   }

PROCEDURE FC_LECTSYMB(VAR chaine,symbole : chaincaract;
                      VAR longsymb,pos_d_lect : loncha);
EXTERN;

{*****************************************}
{ ecriture d'une chaine de caracteres dans }
```

```
{      un fichier.                                    }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);EXTERN;


{************************************************}
{ positionnement du curseur sur l'ecran                    }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;

{-------------------------------------------------------}

PROCEDURE FC_ERRMON(code_erreur : INTEGER;
                    VAR fmessage,OUTPUT : TEXT);

VAR
        erreur,zonimp : chaincaract;
        pos_d_lect,long : loncha;

BEGIN
        FC_VALASS(code_erreur,erreur,fmessage);
        pos_d_lect := 1;
        FC_LECTSYMB(erreur,erreur,long,pos_d_lect);
        FC_TRANSFC(erreur,zonimp,long,lza,FALSE,blan);
        FC_POSITCURS(ligza - nlza + 1,colza);
        FC_ECRLIGNE(zonimp,lza,OUTPUT)
END;
```

PROGRAMMES DU NIVEAU 6

```
#

(*-------------------------------------------------------*)
(*    LECTURE D'UNE CHAINE DE CARACTERES                 *)
(*-------------------------------------------------------*)


(* ENTREE :L'adresse d'un fichier existant et la position *)
(*      de la ligne courante; ce dernier etant implicite  *)
(* SORTIE : Le contenu de la ligne courante et le         *)
(*       positionnement a la ligne courante suivante du   *)
(*      fichier dans le cas ou elle existe                *)
(* FONCTION :  Lecture d'une chaine de caractere dans     *)
(*             la ligne courante                          *)
(*              du fichier jusqu'a la fin de la ligne     *)
(*             courante ou suivant une longueur determinee*)
(*-------------------------------------------------------------------*)

CONST

# include "fcarcst"

TYPE

# include "fcartype"


(* desription des proc. et fonct. externes *)

{*****************************************}
{ lecture d'un caractere dans un fichier }

FUNCTION FC_LECTC(VAR fentree : TEXT):CHAR;
EXTERN;


{*********************************************}
{ lecture d'une chaine de caracteres dans }
{      un fichier                           }

PROCEDURE FC_LECTLIGNE(VAR zonlec : chaincaract;
                       nc : lonlig;
                       VAR fentree : TEXT);


VAR
     i : lonlig;
     caract : CHAR;

(*-------------------------------------------------------------*)

BEGIN
    i := 0;
```

```
    IF EOLN(fentree) THEN READLN(fentree);

    WHILE (NOT EOLN(fentree)) AND ( i <= nc) DO
        BEGIN

          i := i + 1;
          zonlec[i] := FC_LECTC(fentree)

        END;

    zonlec[i + 1] := fdl;

  (* les instructions qui suivent positionnent la position *)
  (* courante du fichier a la nouvelle ligne               *)

    WHILE NOT EOLN(fentree)  DO
      caract := FC_LECTC(fentree);

END;

(*-------------------------------------------------------*)
```

```
#

{ ANALYSE SYNTAXIQUE DE MESSAGES }
{-----------------------------------}

CONST
# include "fcarcst"
# include "fmescst"

TYPE
# include "fcartype"
# include "fmestype"

{*****************************************}
{ lecture d'un symbole dans une chaine de }
{ caracteres   }

PROCEDURE FC_LECTSYMB(VAR chaine,symbole : chaincaract;
                      VAR longsymb,pos_d_lect : loncha);
EXTERN;

{*************************************}
{ test de numericite d'une chaine de }
{      caract`res                    }

FUNCTION FC_TESTNUM(VAR symbole : chaincaract;
                    pos_d_symbole : loncha):BOOLEAN;
EXTERN;

{*******************************************}
{ localisation d'un symbole dans une table }
{ de symboles se trouvant dans un fichier  }

PROCEDURE FC_LOCALSYMB(
                       VAR symbole : chaincaract;
                       pos_d_symb : loncha;
                        VAR pos_table : INTEGER;
                        VAR tabsymbole : TEXT);
EXTERN;



PROCEDURE FC_MESSYNTAX( expediteur : nom_expediteur;
              VAR zon_mes : chaincaract;
              (* chaine a analyser *)
              VAR message : message_cmd;
              VAR tabcmd : TEXT);
      (* zone resultat de l'analyse syntaxique *)
      (* table des commandes *)

VAR
  nodonnee : INTEGER;  (* numero de la donnee a tester *)
  symbole : chaincaract;
  pos_d_lect,longsymb,pos_d_symb : loncha;
```

```
(*-------------------------------------------------------------*)

  PROCEDURE LECTSYMB;  (* lecture d'un symbole *)

  BEGIN
    FC_LECTSYMB(zon_mes,symbole,longsymb,pos_d_lect)
  END;


  PROCEDURE INITIALISATION;

  VAR i_trans : loncha;

  BEGIN

    nodonnee := 0;
    pos_d_lect := 1;
    message.etat := bon;
    longsymb := lmaxcha;
    message.param1_long := 0;
    message.param2_long := 0;
    message.param3_long := 0;

    (* transformation du delimiteur ',' par un blanc *)

    i_trans := 1;
    WHILE zon_mes[i_trans] <> fdl  DO
      BEGIN
        IF zon_mes[i_trans] = ',' THEN
          zon_mes[i_trans] := blan;
        i_trans := i_trans + 1;
      END

  END;


  PROCEDURE TEST_LONGUEUR;  (* test de longueur du symbole lu *)

  (* le symbole lu est soit une commande *)
  (*                  un parametre *)

  BEGIN
    IF ((nodonnee = 1) AND (longsymb > lmaxcmd)) OR
          ((nodonnee = 2) AND (longsymb > lmax1erparam)) OR
          ((nodonnee = 3) AND (longsymb > lmax2emparam)) OR
                ((nodonnee = 4) AND (longsymb > lmax3emparam))
      THEN message.etat := mauvais
  END;


  PROCEDURE CMD_TEST;  (* test de la commande *)

  VAR i : INTEGER;
      pos_tab : INTEGER;
```

```
BEGIN
  pos_d_symb := 1;
  FC_LOCALSYMB(symbole,pos_d_symb,pos_tab,tabcmd);
  IF pos_tab = 0
  THEN message.etat := mauvais
  ELSE
  BEGIN
    IF pos_tab > pos_f_superv THEN
    CASE expediteur OF
    superviseur : message.etat := mauvais;
    edit_compil : IF (pos_tab < pos_d_edit) OR
         (pos_tab > pos_f_edit)
              THEN message.etat := mauvais;
    interp_pupit : IF (pos_tab < pos_d_interp) OR
          (pos_tab > pos_f_interp)
              THEN message.etat := mauvais
    END
  END;

  IF message.etat = bon
    THEN
    BEGIN
      message.cmd_type := TEC;
      FOR i := 2 TO pos_tab DO
      message.cmd_type := succ(message.cmd_type)
    END
END;

PROCEDURE PREMPARAM_TEST;  (* test du 1er parametre *)

VAR i : INTEGER;

BEGIN
  (* remplissage du 1er param *)

  FOR i := 1 TO longsymb DO message.param1[i] := symbole[i];
  FOR i := (longsymb + 1) TO lmax1erparam DO message.param1[i]

  message.param1_long := longsymb;
  message.param1_type := nombre;
  IF FC_TESTNUM(symbole,1) = FALSE
    THEN message.param1_type := caractere
END;


PROCEDURE DEUPARAM_TEST;  (* test du 2em parametre *)

VAR i : INTEGER;

BEGIN
  (* remplissage du 2eme param *)

  FOR i := 1 TO longsymb DO message.param2[i] := symbole[i];
  FOR i := (longsymb + 1) TO lmax2emparam DO message.param2[i]
```

```
  message.param2_long := longsymb;
  message.param2_type := nombre;
  IF FC_TESTNUM(symbole,1) = FALSE
    THEN message.param2_type := caractere
END;

PROCEDURE TROIPARAM_TEST;  (* test du 3em parametre *)

VAR i : INTEGER;

BEGIN
  (* remplissage du 3eme param *)

  FOR i := 1 TO longsymb DO message.param3[i] := symbole[i];
  FOR i := (longsymb + 1) TO lmax3emparam DO message.param3[i]

  message.param3_long := longsymb;
  message.param3_type := nombre;
  IF FC_TESTNUM(symbole,1) = FALSE
    THEN message.param3_type := caractere
END;

PROCEDURE SELECTION_TEST;

BEGIN
  CASE nodonnee OF
  1: CMD_TEST;
  2: PREMPARAM_TEST;
  3: DEUPARAM_TEST;
  4: TROIPARAM_TEST
  END
END;

PROCEDURE TEST_DONNEE;

  (* test du nombre de donnees dans le message *)

BEGIN
  nodonnee := nodonnee + 1;
  IF nodonnee > maxdonnee
    THEN message.etat := mauvais
    ELSE BEGIN
      TEST_LONGUEUR;
      IF message.etat = bon THEN SELECTION_TEST
         END
END;

PROCEDURE CLOTURE;

  (* test de message vide *)

BEGIN
  IF nodonnee = 0
    THEN message.cmd_type := TNIL
```

```
  END;

(*-------------------------------------------------------------------*)

BEGIN

  INITIALISATION;

  WHILE (longsymb > 0) AND (message.etat = bon)
  DO  BEGIN
      LECTSYMB;
      IF longsymb > 0 THEN TEST_DONNEE
    END;

  CLOTURE
END;

(*-------------------------------------------------------------------*)
```

```
#
(*-----------------------------------------*)
(*  ANALYSE SEMANTIQUE DE MESSAGES         *)
(*-----------------------------------------*)

CONST
# include "fmescst"

TYPE
# include "fmestype"

(*-------------------------------------------------------------*)

PROCEDURE FC_MESSEMANT(expediteur : nom_expediteur;
                       VAR message : message_cmd);

BEGIN

END;

(*-------------------------------------------------------------*)
```

```
#

(*-------------------------------------------------------*)
(*  ECRITURE D'UNE CHAINE DE CARACTERES                  *)
(*-------------------------------------------------------*)

{-------------------------------------------------------
* ENTREE : - une chaine de caracteres                   *
*          - longueur de la ligne d'impression          *
*                                                       *
* SORTIE : - ecriture des caracteres concernes          *
*                                                       *
* FONCTION : ecriture de la chaine dans le              *
*            fichier concerne sur la longueur           *
*            = min [position du delimiteur dans         *
*            chaine, longueur demande l'impres-         *
*               sion voulue                             *
*-------------------------------------------------------}

CONST

# include "fcarcst"

TYPE

# include "fcartype"


{*****************************************}
{ ecriture d'un caractere dans un fichier }

PROCEDURE FC_ECRITC(caractere : CHAR;
                    VAR fsortie : TEXT);EXTERN;



(* description de la procedure *)

{*****************************************}
{ ecriture d'une chaine de caracteres dans }
{       un fichier.                        }

PROCEDURE FC_ECRLIGNE(VAR zonimp : chaincaract;
                      longimp : lonlig;
                      VAR fsortie : TEXT);

(* description des variables *)

VAR
   i : loncha;
   car : CHAR;


(*-------------------------------------------------------------------------*)
```

```
BEGIN
   i := 1;
   zonimp[longimp + 1] := fdl;
      (* cela permet d'economiser le test *)
      (* si i depasse la borne longimp *)
   WHILE zonimp[i] <> fdl DO

      BEGIN
         car := zonimp[i];
         FC_ECRITC(car,fsortie);
         i := i + 1
      END;

        WRITELN(fsortie)
END;

(*-------------------------------------------------------*)
```

```
  #
(*-------------------------------------------------------------------------*
(*                                                                         *
(* TRANSFORMATION CHAINE DE CARACTERES ---> NOMBRE ENTIER POSITIF          *
(*                                                                         *
(*-------------------------------------------------------------------------*

  CONST
  # include "fcarcst"

  TYPE
  # include "fcartype"


  PROCEDURE FC_CARNOMB(repres : chaincaract;
                       VAR nombre : INTEGER);


  {-----------------------------------------------------------------
  * ENTREE : une chaine de caracteres contenant une       *
  *         suite de caracteres numeriques representant un*
  *         nombre entier positif et inferieur a la valeur*
  *           maximale entiere MAXINT;                    *
  *                                                       *
  * SORTIE : la valeur entiere representee par la suite   *
  *          de caracteres;                               *
  *                                                       *
  * FONCTION : transformation de chaque caractere numerique*
  *              en un chiffre et combinaison de ces      *
  *              chiffres pour obtenir le nombre          *
  *                        represente.                    *
  -----------------------------------------------------------------}

  VAR
        index : INTEGER;
        chiffre : 0..9;

  BEGIN
    index := 1;
    nombre := 0;

    WHILE (repres[index] <> blan) AND (repres[index] <> fdl) DO
      BEGIN
        chiffre := ORD(repres[index]) - ORD('0');
        nombre := nombre * 10 + chiffre;
        index := index + 1
      END
  END;
```

```
#
(*-------------------------------------------------------------*)
(*                                                             *)
(* TRANSFORMATION ENTIER  --->  CHAINE DE CARACTERES           *)
(*                                                             *)
(*-------------------------------------------------------------*)

CONST
# include "fcarcst"

TYPE
# include "fcartype"

PROCEDURE FC_NOMBCAR(nombre :   (* nombre a representer *)
                        INTEGER;
                     VAR repres : (* zone de reception de la chaine
                                     caracteres *)
                          chaincaract);

{-------------------------------------------------------------------
* ENTREE : nombre entier appartenant a [-MAXINT..MAXINT],
*          adresse zone de reception de la representation du nom
*
* SORTIE : contenu de la zone de reception = representation du n
*
* FONCTION : extraction des chiffres du nombre et codage sous fo
*              caracteres.
*              si le nombre est negatif alors le premier caracte
*              contient le symbole "-"
-------------------------------------------------------------------

VAR     entier,index_i,index_J  : INTEGER;
        chiffre : 0..9;
        repres_renv : chaincaract;

BEGIN
  entier := ABS(nombre);
  index_i := 1;

  REPEAT
    chiffre := entier MOD 10;
    repres_renv[index_i] := CHR(ORD('0') + chiffre);
    index_i := index_i + 1;
    entier := entier DIV 10
  UNTIL entier = 0;

  (* la representation du nombre est renverse, il faut retablir
  (* la situation *)

  IF nombre < 0
  THEN
  BEGIN
    repres_renv[index_i] := moins;
    index_i := index_i + 1
  END;
```

```
    index_j := 1;
    WHILE (index_i - 1) > 0 DO
    BEGIN
      index_i := index_i - 1;
      repres[index_j] := repres_renv[index_i];
      index_j := index_j + 1;
    END;
    repres[index_j] := fdl
END;
```

```
#

(*-----------------------------------------*)
(*     LOCALISATION D'UN SYMBOLE           *)
(*-----------------------------------------*)

{-------------------------------------------------------------
* ENTREE :   - une chaine de caracteres contenant          *
*              un symbole localisee par la position du     *
*              premier caractere, et precedent un          *
*              delimiteur (sentinelle)                     *
*            - une table de symboles se trouvant dans un   *
*              fichier, chaque symbole est une chaine      *
*              de caracteres; les elements de la tables ne*
*              sont pas ordonnes                           *
*                                                          *
* SORTIE :  - la position du symbole dans la table si le   *
*             symbole a ete trouve sinon la valeur 0 est   *
*             est retournee,                               *
*                                                          *
* FONCTION :  - la recherche du symbole recu dans la       *
*               table de symboles                          *
*----------------------------------------------------------}
CONST

# include "fcarcst"

TYPE

# include "fcartype"

{*****************************************}
{ lecture d'une chaine de caracteres dans }
{      un fichier                         }

PROCEDURE FC_LECTLIGNE(VAR zonlect : chaincaract;
                       long_lire : lonlig;
                       VAR fentree : TEXT);EXTERN;

{*****************************************}
{ lecture d'un symbole dans une chaine de }
{ caracteres   }

PROCEDURE FC_LECTSYMB(VAR chaine,symbole : chaincaract;
                      VAR longsymb,pos_d_lect : loncha);
EXTERN;


{*****************************************}
{ localisation d'un symbole dans une table }
{ de symboles se trouvant dans un fichier  }

PROCEDURE FC_LOCALSYMB(
      VAR chaine_symb : chaincaract;
```

```
        (* chaine de caracteres contenant un symbole *)
        (* a rechercher dans la table des symboles   *)
        pos_d_symb : loncha;
        (* position du symbole dans la chaine        *)
        VAR pos_tab : INTEGER;
        (* localisation du symbole dans la table     *)
        (* = 0 si pas localise                       *)
        (* = position dans table si localise         *)
        VAR tabsymb : TEXT
        (* fichier contenant la table des symboles *)
                    );


VAR trouve : BOOLEAN;
    symb_tab, symbole, zonlec : chaincaract;
    longueur : loncha;

(*-------------------------------------------------------------------*)

  PROCEDURE INITIALISATION;

  VAR posit : loncha;

  BEGIN
    posit := pos_d_symb;
    FC_LECTSYMB(chaine_symb, symbole, longueur, posit);

    (* cette lecture est realisee pour : *)
    (* - placer un delimiteur a la fin du symbole *)
    (* - faciliter la comparaison                 *)

    RESET(tabsymb);
    pos_tab := 0;
    trouve := FALSE
  END;


  PROCEDURE LECTURE; (* consultation de la table *)

  VAR posit : loncha;

  BEGIN
    posit := 1;
    FC_LECTLIGNE(zonlec, lmaxlig, tabsymb);
    FC_LECTSYMB(zonlec, symb_tab, longueur, posit);
    pos_tab := pos_tab + 1
  END;


  PROCEDURE COMPARAISON;  (* comparaison symbole avec element *)
        (* de la table des symboles      *)

  VAR i_comp : loncha;
```

```
  BEGIN
    i_comp := 1;

    WHILE (symbole[i_comp] <> fdl)  AND
      (symbole[i_comp] = symb_tab[i_comp]) DO
      i_comp := i_comp + 1;
    IF symbole[i_comp] = symb_tab[i_comp]
    THEN trouve := TRUE
  END;

  PROCEDURE CLOTURE;

  BEGIN
    IF trouve = FALSE THEN pos_tab := 0
  END;

(*-------------------------------------------------------------------*)

BEGIN
  INITIALISATION;
  WHILE (NOT EOF(tabsymb)) AND (trouve = FALSE) DO
  BEGIN
    LECTURE;
    COMPARAISON
  END;
  CLOTURE
END;
```

```
#
(*-------------------------------------------------*)
(*                                                 *)
(*  RECHERCHE VALEUR DANS UN FICHIER SUIVANT       *)
(*              UNE VALEUR DE CLE.                 *)
(*                                                 *)
(*-------------------------------------------------*)

CONST
# include "fcarcst"

TYPE
# include "fcartype"

{*****************************************}
{ lecture d'une chaine de caracteres dans }
{       un fichier                        }

PROCEDURE FC_LECTLIGNE(VAR zonlect : chaincaract;
                       long_lire : lonlig;
                       VAR fentree : TEXT);EXTERN;


(*-------------------------------------------------*)

{*****************************************}
{ recherche d'un enregistrement suivant   }
{ sa position dans un fichier             }

PROCEDURE FC_VALASS(cle : INTEGER;
                    VAR enregist : chaincaract;
                    VAR fentree : TEXT);

{-------------------------------------------------------------
* ENTREE : - un fichier de type "TEXTE",                      *
*          - une cle donnant la position relative d'un       *
*                    enregistrement                          *
*              par rapport au debut de fichier               *
*              (la position determinee par la cle est        *
*               supposee existante dans le fichier           *
*                                                            *
* SORTIE : - valeur de l'enregistrement recherche;           *
*                                                            *
* FONCTION : valeur de l'enregistrement = fichier(cle)       *
-------------------------------------------------------------}

VAR
        j : INTEGER;

BEGIN
        RESET(fentree);
        j := 1;
        WHILE j < cle
        DO   BEGIN
                FC_LECTLIGNE(enregist,0,fentree);
```

```
                j := j + 1
            END;
        FC_LECTLIGNE(enregist, lmaxlig, fentree)
END;
```

```
#
{$c+}

CONST
# include "fextcst"
# include "fcarcst"

TYPE
# include "fexttype"
# include "fcartype"
TEXTE = FILE OF CHAR;

{*****************************************************************
*
* ENTREE : - une chaine de caracteres contenant un nom de fichie
*
* SORTIE : - le fichier defini par "nom de fichier" copie dans
*                le fichier recu en entree,
*          - un valeur signalant le succes de l'operation
*
* FONCTION : - lecture  du fichier externe,
*            - copie du fichier externe dans le fichier donne
*
*****************************************************************

FUNCTION open(nom : string; m : openmode):fdok;EXTERN;
FUNCTION uread(f : fd; VAR b : chaincaract; s : loncha):bsizeok;
FUNCTION close (f : fd):ok;EXTERN;
FUNCTION strbuf(VAR b : chaincaract):string;EXTERN;

{-------------------------------------------------------------}

PROCEDURE FLECTURE(VAR nomfichier : chaincaract;
                   VAR succes : BOOLEAN;
                   VAR fichier : TEXTE);

VAR
   strnomf : string;
   nofichier : fdok;

PROCEDURE INITIALISATION;

VAR
   j : loncha;

BEGIN
   succes := TRUE;
   j := 1;
   WHILE nomfichier[j] <> fdl DO j := j + 1;
   nomfichier[j] := CHR(0);
   strnomf:= strbuf(nomfichier);
   nofichier := open(strnomf,R);
   IF nofichier < 0 THEN succes := FALSE
END;
```

```
PROCEDURE TRAITEMENT;

VAR
   j : loncha;
   longueur : bsizeok;
   ligne : chaincaract;

BEGIN
   REWRITE(fichier);
   longueur := uread(nofichier,ligne,lmaxlig);
   WHILE longueur  > 0
   DO BEGIN
      FOR j := 1 TO longueur DO WRITE(fichier,ligne[j]);
      longueur := uread(nofichier,ligne,lmaxlig)
   END;

   IF longueur = -1 THEN succes := FALSE
END;

PROCEDURE CLOTURE;

BEGIN
   IF close(nofichier) < 0 THEN succes := FALSE
END;


{-------------------------------------------------}

BEGIN
   INITIALISATION;
   IF succes THEN TRAITEMENT;
   IF succes THEN CLOTURE
END;
```

```
#
{$c+}

CONST
# include "fextcst"
# include "fcarcst"

TYPE
# include "fexttype"
# include "fcartype"
uncaract = PACKED ARRAY [1..1] OF CHAR;
TEXTE = FILE OF CHAR;

{*****************************************************************
*
* ENTREE : - une chaine de caracteres contenant un nom de fichie
*          - le fichier a recopie dans le fichier externe
*
* SORTIE : - le fichier defini par "nom de fichier" contenant le
*            le contenu du fichier recu en entree
*          - un valeur signalant le succes de l'operation
*
* FONCTION : - creation du fichier externe
*            - copie du fichier donne dans le fichier externe
*
*****************************************************************

FUNCTION open(nom : string; m : openmode):fdok;EXTERN;
FUNCTION creat(nom : string; m : set of modebits):fdok;EXTERN;
FUNCTION uwrite(f : fd; VAR b : uncaract; s : loncha):bsizeok;EX
FUNCTION close (f : fd):ok;EXTERN;
FUNCTION strbuf(VAR b : chaincaract):string;EXTERN;

{---------------------------------------------------------------

PROCEDURE FECRITURE(VAR nomfichier : chaincaract;
        VAR succes : BOOLEAN;
        VAR fichier : TEXTE);

VAR
  strnomf : string;
  nofichier : fdok;
  protection : set of modebits;

PROCEDURE INITIALISATION;

VAR
  j : loncha;

BEGIN
  succes := TRUE;
  j := 1;
  WHILE nomfichier[j] <> fdl DO j := j + 1;
  nomfichier[j] := CHR(0);
  strnomf:= strbuf(nomfichier);
```

```
  protection := [WHIM,RHIM,WYOU,RYOU,WME,RME];
  nofichier := creat(strnomf,protection);
  IF nofichier < 0 THEN succes := FALSE
END;

PROCEDURE TRAITEMENT;

VAR
  longueur : bsizeok;
  caractere : uncaract;

BEGIN
  RESET(fichier);
  IF NOT EOF(fichier)
  THEN BEGIN
       READ(fichier,caractere[1]);
    longueur := uwrite(nofichier,caractere,1)
       END;

  WHILE (NOT EOF(fichier)) AND (longueur > 0)
  DO BEGIN
    READ(fichier,caractere[1]);
    longueur := uwrite(nofichier,caractere,1)
     END;

  IF longueur = -1 THEN succes := FALSE
END;

PROCEDURE CLOTURE;

BEGIN
  IF close(nofichier) < 0 THEN succes := FALSE
END;


{-------------------------------------------------------}

BEGIN
  INITIALISATION;
  IF succes THEN TRAITEMENT;
  CLOTURE
END;
```

PROGRAMMES DU NIVEAU 7

```
#
CONST
# include "fgrrcpt"

TYPE
# include "fgrtype"

# include "fgrproc"

PROCEDURE FC_NETECRAN;


        { cette procedure efface ce qui est presente }
        { a l'ecran du terminal }
BEGIN
        vt100;
        erase(screen);
        vt52
END;
```

```
#

CONST

# include "fgrrcpt"

TYPE
# include "fgrtype"

# include"fgrproc"

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);

        { cette procedure place le curseur a une certaine }
        { position de l'ecran du terminal }

BEGIN
        vt100;
        posit(lig_curs,col_curs);
        vt52
END;
```

```
{*-----------------------------------------*}
{* ARRET de l'execution du programme *}
{* pour un delai FIXE                *}
{*-----------------------------------------*}

PROCEDURE SLEEP(seconds : INTEGER);EXTERN;

PROCEDURE FC_ATTENTE(secondes : INTEGER);

BEGIN
        SLEEP(secondes)
END;
```

```
{-----------------------------}
{ LECTURE D'UN CARACTERE }
{-----------------------------}

FUNCTION FC_LECTC (VAR fentree : TEXT)  : CHAR;

VAR c : CHAR;

(*-------------------------------------------------------------------------*)
(* ENTREE: cette procedure recoit l'adresse d'un fichier*)
(*         "texte" et l'adresse de la position courante  *)
(*         dans ce fichier; ce dernier etant implicite   *)
(* SORTIE: un caractere du fichier dont la position      *)
(*         est determinee par la position courante       *)
(* FONCTION: cette procedure realise la lecture d'un     *)
(*         caractere appartenant a un fichier            *)
(*-------------------------------------------------------------------------*)

BEGIN
          READ(fentree,c);
          FC_LECTC := c
END;
```

```
(*-----------------------------------*)
(*       ECRITURE CARACTERE          *)
(*-----------------------------------*)

{-------------------------------------------------------
* ENTREE : l'adresse d'un fichier "TEXTE"              *
*          la position courante dans ce fichier        *
*          (ce dernier etant implicite)                *
*                                                      *
* SORTIE : un caractere ecrit sur le fichier           *
*                                                      *
* FONCTION : ecriture a la position courante           *
*            du fichier du caractere considere         *
*                                                      *
-------------------------------------------------------}

PROCEDURE FC_ECRITC(c : CHAR;VAR fsortie : TEXT);
          BEGIN
               WRITE (fsortie,c)
          END;
```

```
#


(*-----------------------------------*)
(*       TEST NUMERIQUE              *)
(*-----------------------------------*)

{-------------------------------------------------------------
* ENTREE : un symbole se trouvant dans une chaine           *
*          a partir d'une position determinee               *
*                                                           *
* SORTIE : la fonction recoit la valeur booleenne           *
*               TRUE si le symbole est numerique            *
*              FALSE si le symbole n'est pas numerique      *
*                                                           *
* FONCTION : verification de chaque caractere du            *
*              symbole si elle est un chiffre decimal       *
*                                                           *
-------------------------------------------------------------}

CONST
# include "fcarcst"

TYPE
# include "fcartype"

FUNCTION FC_TESTNUM (VAR symbole : chaincaract;
                (* chaine contenant le symbole a tester *)
                     pos_d_symbole : loncha
                (* position du symbole dans la chaine *)
                     ):BOOLEAN;


VAR
        numerique : BOOLEAN;

BEGIN

numerique := TRUE;

WHILE (symbole[pos_d_symbole] <> fdl) AND (numerique = TRUE)
DO
BEGIN
  IF (NOT (symbole[pos_d_symbole] IN ['1','2','3','4','5',
      '6','7','8','9','0']))
  THEN numerique := FALSE;
  pos_d_symbole := pos_d_symbole + 1
END;

FC_TESTNUM := numerique

END;
```

```
#
(*-------------------------------------------------------*)
(*       TRANSFERT DE CARACTERES                         *)
(*-------------------------------------------------------*)


{-------------------------------------------------------
* ENTREE : une chaine de caracteres contenant   *
*                un symbole;                     *
*                                                *
* SORTIE : une chaine de caracteres contenant   *
*                un symbole;                     *
*                                                *
* FONCTION : transfert d'une partie de la chaine*
*            de caracteres recus en entree dans *
*            une zone sur une longueur determine*
*            et garnissage eventuel des positions *
*           non occupees et possibilite de tester *
*           le signe avec placement du signe     *
*           eventuel a la premiere position dans  *
*           la zone de reception, ce signe = '-'  *
*           si la partie de la chaine consideree  *
*           commence par un '-' sinon il y aura   *
*          un blanc                               *
-------------------------------------------------------}
CONST
# include "fcarcst"

TYPE
# include "fcartype"

PROCEDURE FC_TRANSFC(
                VAR
                   emetteur,
          (* zone contenant la chaine de caracteres *)
          (* la zone consideree commence en position*)
          (* 1 de la chaine recue                   *)
                   recepteur
          (* zone de reception de la chaine de car-*)
          (* acteres considerees                    *)
                        : chaincaract;
                   long_emet,
          (* longueur de la zone consideree *)
                   long_recept
          (* longueur de la zone desiree apres *)
          (* transfert *)
                  : loncha;
                   test_signe : BOOLEAN;
          (* si le signe de la chaine doit etre *)
          (* teste          *)
                   car_rempl : CHAR
          (* caractere de remplissage *)
                 );

(*-------------------------------------------------------*)
```

```
PROCEDURE TRAITEMENT;

VAR
  i, j, deb_rempl : loncha;

BEGIN
  deb_rempl := 1;
  (* transfert de la valeur recue *)

  i := long_recept;
  FOR j := long_emet DOWNTO 1 DO
    BEGIN
      recepteur[i] := emetteur[j];
      i := i - 1
    END;

  (* test du signe *)

  IF test_signe = TRUE
    THEN  BEGIN
      deb_rempl := 2;
      IF recepteur[i + 1] = moins
      THEN BEGIN
        recepteur[1] := moins;
        i := i + 1
           END
      ELSE BEGIN
           recepteur[1] := plus;
           IF recepteur[i + 1] = plus
           THEN i := i + 1
      END;
    END;

  (* remplissage des positions non occupees *)
  (* du recepteur                           *)


  FOR j := i DOWNTO deb_rempl
    DO recepteur[j] := car_rempl
END;
(*-------------------------------------------------------------*)

BEGIN
  IF long_emet <= long_recept THEN TRAITEMENT
END;
```

```
#
CONST
# include "fgrrcpt"

TYPE

# include "fgrtype"

# include "fgrproc"

PROCEDURE MODE_ASCII;

 (* cette procedure place le terminal en mode caractere *)

BEGIN
        ascii;
        vt52
END;
```

```
#
CONST
# include "fgrrcpt"

TYPE
# include "fgrtype"

# include "fgrproc"

PROCEDURE MODE_GRAPHE;

  (* cette procedure place le terminal en mode graphique *)

BEGIN
        vt100;
        graph
END;
```

PROGRAMMES DE MAINTENANCE.

DONNEES UTILISEES.

```
{ Information generale }

  ndessin = 10;
  longecran = 80;       { largeur de l'ecran }
  largecran = 23;       { largeur de l'ecran - 1}
  blan = ' ';

{ Description des informations sur la representation }
{ graphique de l'ordinateur pedagogique              }

{*** la memoire centrale ***}

  ligdmem = 2;          { ligne debut du cadre de la memoire centrale }
  coldmem = 3; { colonne debut du cadre de la memoire centrale }
  hautmot = 1; { hauteur d'un mot de la memoire centrale }
  lmot = 6;     { longueur d'un mot de la memoire centrale }
  distmot = 1;          { distance entre chaque mot de la memoire centr
  nmotligmem = 10; { nombre de mots sur une ligne de la memoire }
                { centrale affichee }
  nmotcolmem = 10; { nombre de mots sur une colonne de la memoire }
                { centrale affichee }
   dindcmem = 1;        { distance entre cadre memoire et indice colon
                { de la memoire centrale }
   dindlmem = 1;        { distance entre cadre memoire et indice ligne
                { de la memoire centrale }

 {*** le lecteur de cartes ***}

   dlectmem = 1;        { distance entre le lecteur  et le cadre de la
                { memoire centrale }
   coldlect = 3;        { colonne debut du lecteur de cartes }
   hautcart = 1;        { hauteur d'une carte }
   ncart = 5;    { nombre de cartes visibles a l'ecran }
   lcart = 20;   { nombres de caracteres sur une carte }

 {*** les registres de niveau 1 ***}
   dregmem = 1; { distance entre la premiere rangee de registres }
                { et le cadre de la memoire centrale }
 { le compteur ordinal }
   dcolect = 3; {distance entre le compteur ordinal et le lecteur }
                { de cartes }
   lco = 2;     { longueur du compteur ordinal }
   { le registre instruction }
     drico = 3;   { distance entre le registre I et le compteur }
                  { ordinal }
```

```
{ le code operatoire }
  dcopri = 4;   { distance entre le code operatoire et le registre
                { instruction }
  lcop = 3;     { longueur du code operatoire }

{*** les registres de niveau 2 ***}
  dregreg = 1; { distance entre registre de niveau 1 et les }
               { registres situees sur la deuxieme rangee }
{ le registre extension }

{ le registre accumulateur }
  dare = 2;     { distance entre registre A et le registre }
                { extension }
{ l'indicateur logique }
  dila = 2;     { distance entre l'indicateur logique et le }
                { registre A }
  lil = 2;      { longueur de l'indicateur logique }

{*** la zone d'affichage ***}

  dzalect = 3; { distance entre la zone d'affichage et le }
               { lecteur de cartes }
  dzareg = 1;  { distance entre la zone d'affichage et la }
               { deuxieme rangee de registres }
  lza = 20;    { longueur de la zone d'affichage }

  hautza = 2;  { hauteur de la zone d'affichage }
               { ligne inferieur = ligne message utilisateur }
               { ligne superieur = ligne message erreur }

{*** l'imprimante ***}

  dimpmem = 1;    { distance entre l'imprimante et le cadre }
               { de la memoire centrale }
  dimpza = 3;  { distance entre l'imprimante et la zone }
               { d'affichage }
  hautlig = 1; { hauteur d'une ligne }
  nlig = 5;    { nombre de lignes de l'imprimante visible a l'ecran
  llig = 24;   { longueur d'une ligne de l'imprimante }

{-------------------------------------------------------------------}
 { PARTICULARITE GRAPHIQUE DEPENDANT DU TERMINAL UTILISEE.    }
 {-------------------------------------------------------------------}

   lighoriz = 'q';       { ligne horizontale }
   inf_dr_coin = 'j';    { coin inferieur droit }
   sup_dr_coin = 'k';    { coin superieur droit }
   sup_ga_coin = 'l';    { coin superieur gauche }
   inf_ga_coin = 'm';    { coin inferieur gauche }
   ligvert = 'x';        { ligne verticale }
```

LES PROGRAMMES UTILISES.

```
#

{-------------------------------------------------------------
* ENTREE : un fichier contenant les donnees     *
*          relatives de chaque composantes      *
*                                               *
* SORTIE : un message d'erreur ou alors :       *
*          le schema graphique de l'ordinateur  *
*          et les donnees de chaque composante en*
*          valeur absolue;                      *
*                                               *
* FONCTION : calcul et verification pour chaque *
*            composante des coordonnees en      *
*            valeur absolue, um code_erreur     *
*            sera envoye si une coordonnee ne   *
*               se touve pas dans les bornes    *
*               de l'ecran.                     *
*                                               *
-------------------------------------------------------------}

PROGRAM GRAPH(INPUT,OUTPUT,ordpeddes,fgrrcpt);

CONST

        oui = 'O';
        non = 'N';

# include "fgrcst"

TYPE

# include "fgrtype"
  dim_ecran = PACKED ARRAY [1..largecran,1..longecran] OF CHAR;

VAR

  ordpeddes : TEXT; { fichier de reception du graphique }
  fgrrcpt : TEXT; { fichier des adresses des zones de  reception
}
      { du graphique "ordinateur pedagogique }

  ligcart,colcart : INTEGER;  { indices  pour  la  creation  du
lecteur  }
          { de cartes }
  ligimp,colimp  :  INTEGER;  { indices  pour  la  creation  de
l'imprimante }
  i,j : INTEGER;     { indices divers }
  ecran : dim_ecran;
  code_param : INTEGER;
  code_erreur : INTEGER;
  rep_o_n : CHAR;

  { variable graphique }

  ldmem,lfmem,cdmem,cfmem : INTEGER;
```

```
  ldlect,lflect,cdlect,cflect : INTEGER;
  ldco,lfco,cdco,cfco : INTEGER;
  ldri,lfri,cdri,cfri : INTEGER;
  ldcop,lfcop,cdcop,cfcop : INTEGER;
  ldre,lfre,cdre,cfre : INTEGER;
  lda,lfa,cda,cfa : INTEGER;
  ldil,lfil,cdil,cfil : INTEGER;
  ldza,lfza,cdza,cfza : INTEGER;
  ldimp,lfimp,cdimp,cfimp : INTEGER;

  { variable pour les indices de la memoire centrale }

  indlmem,indcmem : INTEGER; { ligne des indices colonnes, }
        { et colonne des indices lignes }
  pos_i_mem, pos_j_mem : INTEGER;  { position debut des : }
              { - indices de ligne }
              { - indices de colonne }
  unite,dizain : INTEGER;

# include "fgrproc"


PROCEDURE zonrecp(VAR fgrrcpt : TEXT);EXTERN;

  { cette procedure cree un fichier de constante contenant les }
  { adresses des zones de reception des donnees dans le
graphique }
  { de l'ordinateur pedagogique }

{*************************************}
{ placer le terminal en mode graphique }

PROCEDURE MODE_GRAPHE;EXTERN;

{*************************************}
{ placer le terminal en mode caractere }

PROCEDURE MODE_ASCII;EXTERN;

{*************************************************}
{ positionnement du curseur sur l'ecran                }

PROCEDURE FC_POSITCURS(lig_curs : nolig_ecran;
                       col_curs : nocol_ecran);
EXTERN;
{*********************************}
{ effacement du contenu de l'ecran }

PROCEDURE FC_NETECRAN;EXTERN;


PROCEDURE   cadre(VAR   lig_debut,lig_fin,col_debut,col_fin    :
INTEGER;
    VAR tab_ecran : dim_ecran );
```

```
 { cette procedure realise le dessin  d'un  cadre  a  partir  des
coordonnees }
 { suivantes : - la premiere ligne du cadre  }
 {             - la derniere ligne du cadre  }
 {             - la premiere colonne du cadre }
 {             - la derniere colonne du cadre }

    VAR
   lim_colon,lim_ligne :  INTEGER;
   i,j :  INTEGER;

   BEGIN


     { realisation des coins du cadre }

     tab_ecran[lig_debut,col_debut] := sup_ga_coin;
     tab_ecran[lig_debut,col_fin] := sup_dr_coin;
     tab_ecran[lig_fin,col_debut] := inf_ga_coin;
     tab_ecran[lig_fin,col_fin] := inf_dr_coin;

     { cotes horizontaux du cadre }


     lim_colon := col_fin - 1;
     FOR j := col_debut + 1 TO lim_colon DO
       BEGIN
        tab_ecran[lig_debut,j] := lighoriz;
        tab_ecran[lig_fin,j] := lighoriz
       END;

     { cotes verticaux du cadre }


     lim_ligne := lig_fin - 1;
     FOR i := lig_debut + 1 TO lim_ligne DO
       BEGIN
        tab_ecran[i,col_debut] := ligvert;
        tab_ecran[i,col_fin] := ligvert
       END
     END;

 BEGIN { PROGRAMME PRINCIPAL }

 {* 1. test et calcul des coordonnees des dessins *}

   code_param := 1;        { code qui selectionne le  parametre  a
tester }
   code_erreur := 0;        { code  qui  contient  le  numero  de
L'erreur }
   cdmem := coldmem;
   ldmem := ligdmem;
   cdlect := coldmem;

   WHILE code_erreur = 0 DO
```

```
BEGIN
  CASE code_param OF

  { memoire centrale }

  1: IF (ldmem > largecran) OR (ldmem < 2) THEN
     code_erreur := 1;
  2: BEGIN
     lfmem := ldmem + (hautmot * nmotcolmem) + 1;
     IF lfmem > largecran THEN code_erreur := 2
     END;
  3: IF (cdmem > longecran) OR (cdmem < 3) THEN
     code_erreur := 3;
  4: BEGIN
     cfmem := cdmem + (lmot + distmot) * nmotligmem + 1;
     IF cfmem > longecran THEN code_erreur := 4
     END;

  { lecteur de cartes }

  5: BEGIN
     ldlect := lfmem + dlectmem;
     IF ldlect > largecran THEN code_erreur := 5
     END;
  6: BEGIN
     lflect := ldlect + (ncart * (hautcart + 1)) - 1;
     IF lflect > largecran THEN code_erreur := 6
     END;
  7: IF cdlect > longecran THEN code_erreur := 7;
  8: BEGIN
     cflect := cdlect + lcart;
     IF cflect > longecran THEN code_erreur := 8
     END;

  { compteur ordinal }

  9: BEGIN
     ldco := lfmem + dregmem;
     IF ldco > largecran THEN code_erreur := 9
     END;
  10: BEGIN
      lfco := ldco + hautmot + 1;
      IF lfco > largecran THEN code_erreur := 10
      END;
  11: BEGIN
      cdco := cflect + dcolect;
      IF cdco > longecran THEN code_erreur := 11
      END;
  12: BEGIN
      cfco := cdco + lco + 1;
      IF cfco > longecran THEN code_erreur := 12
      END;

  { registre instruction }
```

```
13: BEGIN
    ldri := lfmem + dregmem;
    IF ldri > largecran THEN code_erreur := 13
    END;
14: BEGIN
    lfri := ldri + hautmot + 1;
    IF lfri > largecran THEN code_erreur := 14
    END;
15: BEGIN
    cdri := cfco + drico;
    IF cdri > longecran THEN code_erreur := 15
    END;
16: BEGIN
    cfri := cdri + lmot + 1;
    IF cfri > longecran THEN code_erreur := 16
    END;

{ code operatoire }

17: BEGIN
    ldcop := lfmem + dregmem;
    IF ldcop > largecran THEN code_erreur := 17
    END;
18: BEGIN
    lfcop := ldcop + hautmot + 1;
    IF lfcop > largecran THEN code_erreur := 18
    END;
19: BEGIN
    cdcop := cfri + dcopri;
    IF cdcop > longecran THEN code_erreur := 19
    END;
20: BEGIN
    cfcop := cdcop + lcop + 1;
    IF cfcop > longecran THEN code_erreur :=20
    END;

{ registre extension }

21: BEGIN
    ldre := lfco + dregreg;
    IF ldre > largecran THEN code_erreur := 21
    END;
22: BEGIN
    lfre := ldre + hautmot + 1;
    IF lfre > largecran THEN code_erreur := 22
    END;
23: BEGIN
    cdre := cflect + dcolect;
    IF cdre > longecran THEN code_erreur := 23
    END;
24: BEGIN
    cfre := cdre + lmot + 1;
    IF cfre > longecran THEN code_erreur := 24
    END;
```

```
{ accumulateur }

25: BEGIN
    lda := lfco + dregreg;
    IF lda > largecran THEN code_erreur := 25
    END;
26: BEGIN
    lfa := lda + hautmot + 1;
    IF lfa > largecran THEN code_erreur := 26
    END;
27: BEGIN
    cda := cfre + dare;
    IF cda > longecran THEN code_erreur := 27
    END;
28: BEGIN
    cfa := cda + lmot + 1;
    IF cfa > longecran THEN code_erreur := 28;
    END;


{ l'indicateur logique }

29: BEGIN
    ldil := lfco + dregreg;
    IF ldil > largecran THEN code_erreur := 29
    END;
30: BEGIN
    lfil := ldil + hautmot + 1;
    IF lfil > largecran THEN code_erreur := 30
    END;
31: BEGIN
    cdil := cfa + dila;
    IF cdil > longecran THEN code_erreur := 31
    END;
32: BEGIN
    cfil := cdil + lil + 1;
    IF cfil > longecran THEN code_erreur := 32
    END;

{ la zone d'affichage }

33: BEGIN
    ldza := lfre + dzareg;
    IF ldza > largecran THEN code_erreur := 33
    END;
34: BEGIN
    lfza := ldza + hautza + 1;
    IF lfza > largecran THEN code_erreur := 34
    END;
35: BEGIN
    cdza := cflect + dzalect;
    IF cdza > longecran THEN code_erreur := 35
    END;
36: BEGIN
    cfza := cdza + lza + 1;
```

```
        IF cfza > longecran THEN code_erreur := 36
        END;

    { l'imprimante }

    37: BEGIN
        ldimp := lfmem + dimpmem;
        IF ldimp > largecran THEN code_erreur := 37
        END;
    38: BEGIN
        lfimp := ldimp + (nlig * (hautlig + 1)) - 1;
        IF lfimp > largecran THEN code_erreur := 38
        END;
    39: BEGIN
        cdimp := cfza + dimpza;
        IF cdimp > longecran THEN code_erreur := 39
        END;
    40: BEGIN
        cfimp := cdimp + llig;
        IF cfimp > longecran THEN code_erreur := 40
        END;
    41: code_erreur := 99

    END;
    code_param := code_param + 1
  END;
IF code_erreur <> 99 THEN
     WRITELN('erreur code = ',code_erreur)
ELSE BEGIN

  { initialisation de l'ecran }

  WRITELN('init');

  FOR i := 1 TO largecran DO
       FOR j := 1 TO longecran DO  ecran[i,j] := blan;

  { remplissage de la zone ecran }

  { 1. dessin memoire centrale }

  cadre(ldmem,lfmem,cdmem,cfmem,ecran);

  { 1.1 ecriture des indices de la memoire centrale }

  { 1.1.1 indice colonne }

  indlmem := ldmem - 1;
  pos_j_mem := cdmem + 1 + (lmot + 1 + distmot) DIV 2;

  ecran[indlmem,pos_j_mem] := '0';

  FOR j := 1 TO nmotligmem - 1 DO
  BEGIN
    pos_j_mem := pos_j_mem + lmot + distmot;
```

```
  IF j < 10 THEN ecran[indlmem,pos_j_mem] :=
                              CHR(ORD('0') + j)
       ELSE BEGIN
            unite := j MOD 10;
            dizain := j DIV 10;
            ecran[indlmem,pos_j_mem] :=
                              CHR(ORD('0') + dizain);
            ecran[indlmem,pos_j_mem + 1] :=
                              CHR(ORD('0') + unite)
            END
END;

{ 1.1.2 indice ligne }

indcmem := cdmem - 2;
pos_i_mem := ldmem + (hautmot + 1) DIV 2;

ecran[pos_i_mem,indcmem] := '0';

FOR i := 1 TO nmotcolmem - 1 DO
BEGIN
  pos_i_mem := pos_i_mem + hautmot;
  FC_POSITCURS(pos_i_mem,indcmem);
  IF i < 10 THEN ecran[pos_i_mem,indcmem] :=
                              CHR(ORD('0') + i)
     ELSE BEGIN
            unite := i MOD 10;
            dizain := i DIV 10;
            ecran[pos_i_mem,indcmem] :=
                    CHR(ORD('0') + dizain );
            ecran[pos_i_mem,indcmem + 1] :=
                    CHR(ORD('0') + unite )
          END
END;


{ 2. dessin du lecteur de cartes }

FOR i :=1 TO ncart DO
  BEGIN
  ligcart := ldlect - 1 + (i * (hautcart + 1));
  FOR colcart := cdlect TO cflect DO
   ecran[ligcart,colcart] := lighoriz
  END;

{ 3. dessin du compteur ordinal }

cadre(ldco,lfco,cdco,cfco,ecran);

{ 4. dessin du registre instruction }

cadre(ldri,lfri,cdri,cfri,ecran);

{ 5. dessin du code operatoire }
```

```
  cadre(ldcop,lfcop,cdcop,cfcop,ecran);

  { 6. dessin du registre extension }

  cadre(ldre,lfre,cdre,cfre,ecran);

  { 7. dessin de l'accumulateur }

  cadre(lda,lfa,cda,cfa,ecran);

  { 8. dessin de l'indicateur logique }

  cadre(ldil,lfil,cdil,cfil,ecran);

  { 9. dessin de la zone d'affichage }

  cadre(ldza,lfza,cdza,cfza,ecran);

  { 10. dessin de l'imprimante }


  FOR i := 1 TO nlig DO
    BEGIN
    ligimp := ldimp - 1 + (i * (hautlig + 1));
    FOR colimp := cdimp TO cfimp DO
     ecran[ligimp,colimp] := lighoriz
    END;

{ affichage du graphique a l'ecran }

MODE_GRAPHE;
FC_NETECRAN;


FOR i := 1 TO largecran DO
  BEGIN

    FC_POSITCURS(i,1);
    FOR j := 1 TO longecran DO WRITE(ecran[i,j]);
  END;

ascii;


{ demande d'acceptation du graphique }

REPEAT
  FC_POSITCURS(ligdmem + 1,coldmem + 1);
  WRITE('DESSIN OK ? ( O = OUI, N = NON)    ');
  back(1);
  READ(rep_o_n)
UNTIL (rep_o_n = oui) OR (rep_o_n = non);

FC_POSITCURS(ligdmem + 3,coldmem + 3);
```

```
  IF rep_o_n = oui THEN
      BEGIN
       { ecriture du dessin dans le fichier ordpeddes }
       REWRITE(ordpeddes);
       FOR i := 1 TO largecran DO
              BEGIN
                     FOR j := 1 TO longecran DO
                     WRITE(ordpeddes, ecran[i, j]);
                     WRITELN(ordpeddes)
              END;
       WRITE(' GRAPHE PRESENT DANS FICHIER ',
                                  'ordpeddes ');
       zonrecp(fgrrcpt);
       FC_POSITCURS(ligdmem + 5, coldmem + 3);
       WRITE(' ADRESSES DES ZONES DE RECEPTION ',
              ' DANS FICHIER fgrrcpt ')

      END
      ELSE
       WRITELN(' DESSIN PRESENTE N''EST PAS SAUVE ');
  FC_POSITCURS(largecran, 1);
  MODE_GRAPHE
  END
END.
```

```
#

CONST

# include "fgrcst"

PROCEDURE zonrecp(VAR fconst : TEXT);


        { cette procedure cree la liste des constantes
definissant }
        { les adresses de reception des donnees dans le graphique
}
        { de l'ordinateur pedagogique }


VAR


        ldmem, lfmem, cdmem, cfmem : INTEGER;
        ldlect, lflect, cdlect, cflect : INTEGER;
        ldco, lfco, cdco, cfco : INTEGER;
        ldri, lfri, cdri, cfri : INTEGER;
        ldcop, lfcop, cdcop, cfcop : INTEGER;
        ldre, lfre, cdre, cfre : INTEGER;
        lda, lfa, cda, cfa : INTEGER;
        ldil, lfil, cdil, cfil : INTEGER;
        ldza, lfza, cdza, cfza : INTEGER;
        ldimp, lfimp, cdimp, cfimp : INTEGER;




BEGIN { PROGRAMME PRINCIPAL }


        cdmem := coldmem;
        ldmem := ligdmem;
        cdlect := coldmem;


        lfmem := ldmem + (hautmot * nmotcolmem) + 1;
        cfmem := cdmem + (lmot + distmot) * nmotligmem + 1;
        ldlect := lfmem + dlectmem;
        lflect := ldlect + (ncart * hautcart);
        cflect := cdlect + lcart;
        ldco := lfmem + dregmem;
        lfco := ldco + hautmot + 1;
        cdco := cflect + dcolect;
        cfco := cdco + lco + 1;
        ldri := lfmem + dregmem;
        lfri := ldri + hautmot + 1;
```

```
cdri := cfco + drico;
cfri := cdri + lmot + 1;
ldcop := lfmem + dregmem;
lfcop := ldcop + hautmot + 1;
cdcop := cfri + dcopri;
cfcop := cdcop + lcop + 1;
ldre := lfco + dregreg;
lfre := ldre + hautmot + 1;
cdre := cflect + dcolect;
cfre := cdre + lmot + 1;
lda := lfco + dregreg;
lfa := lda + hautmot + 1;
cda := cfre + dare;
cfa := cda + lmot +1;
ldil := lfco + dregreg;
lfil := ldil + hautmot + 1;
cdil := cfa + dila;
cfil := cdil + lil + 1;
ldza := lfre + dzareg;
lfza := ldza + hautza + 1;
cdza := cflect + dzalect;
cfza := cdza + lza + 1;
ldimp := lfmem + dimpmem;
lfimp := ldimp + nlig;
cdimp := cfza + dimpza;
cfimp := cdimp + llig;

REWRITE(fconst);
WRITELN(fconst,'{*** ecran ***}');
WRITELN(fconst,'longecran = ',longecran,';');
WRITELN(fconst,'largecran = ',largecran,';');
WRITELN(fconst);
WRITELN(fconst,'{*** memoire centrale ***}');
WRITELN(fconst,'ligmot1 = ',ligdmem + hautmot,';');
WRITELN(fconst,'colmot1 = ',coldmem + distmot + 1,';');
WRITELN(fconst,'hautmot =',hautmot,';');
WRITELN(fconst,'lmot = ',lmot,';');
WRITELN(fconst,'distmot =',distmot,';');
WRITELN(fconst,'nmotligmem = ',nmotligmem,';');
WRITELN(fconst,'nmotcolmem = ',nmotcolmem,';');
WRITELN(fconst);
WRITELN(fconst,'{*** lecteur de cartes ***}');
WRITELN(fconst,'ligcart1 = ',ldlect,';');
WRITELN(fconst,'colcart1 = ',cdlect,';');
WRITELN(fconst,'hautcart = ',hautcart,';');
WRITELN(fconst,'ncart = ',ncart,';');
WRITELN(fconst,'lcart = ',lcart,';');
WRITELN(fconst);
WRITELN(fconst,'{*** compteur ordinal ***}');
WRITELN(fconst,'ligco = ',ldco + hautmot,';');
WRITELN(fconst,'colco = ',cdco + 1,';');
WRITELN(fconst,'lco = ',lco,';');
WRITELN(fconst);
WRITELN(fconst,'{***  registre instruction ***}');
WRITELN(fconst,'ligri = ', ldri + hautmot,';');
```

```
        WRITELN(fconst,'colri = ', cdri + 1,';');
        WRITELN(fconst);
        WRITELN(fconst,'{*** code operatoire ***}');
        WRITELN(fconst,'ligcop = ', ldcop + hautmot,';');
        WRITELN(fconst,'colcop = ', cdcop + 1,';');
        WRITELN(fconst,'lcop = ',lcop,';');
        WRITELN(fconst);
        WRITELN(fconst,'{*** registre extension ***}');
        WRITELN(fconst,'ligre = ',ldre + hautmot,';');
        WRITELN(fconst,'colre = ',cdre + 1,';');
        WRITELN(fconst);
        WRITELN(fconst,'{*** accumulateur ***}');
        WRITELN(fconst,'liga = ',lda + hautmot,';');
        WRITELN(fconst,'cola = ',cda + 1,';');
        WRITELN(fconst);
        WRITELN(fconst,'{*** indicateur logique ***}');
        WRITELN(fconst,'ligil = ',ldil + hautmot,';');
        WRITELN(fconst,'colil = ',cdil + 1,';');
        WRITELN(fconst,'lil = ',lil,';');
        WRITELN(fconst);
        WRITELN(fconst,'{*** zone d''affichage ***}');
        WRITELN(fconst,'ligza = ',ldza + hautza,';');
        WRITELN(fconst,'colza = ',cdza + 1,';');
        WRITELN(fconst,'lza = ',lza,';');
        WRITELN(fconst,'nlza =',hautza,';');
        WRITELN(fconst);
        WRITELN(fconst,'{*** imprimante ***}');
        WRITELN(fconst,'ligimp1 = ',ldimp,';');
        WRITELN(fconst,'colimp1 = ',cdimp,';');
        WRITELN(fconst,'hautlig = ',hautlig,';');
        WRITELN(fconst,'nlig = ',nlig,';');
        WRITELN(fconst,'llig = ',llig,';');
END;
```