**MASTER EN SCIENCES INFORMATIQUES**

# Conception et réalisation d'un gestionnaire d'agendas

Gillard, Jean-Luc

*Award date:*
1983

*Awarding institution:*
Universite de Namur

Facultès universitaires Notre-Dame de la Paix (Namur)

Institut d'Informatique

# CONCEPTION ET REALISATION

# D'UN GESTIONNAIRE D'AGENDAS

Mèmoire prèsentè par

Jean-Luc GILLARD

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

Annèe acadèmique 1982 - 1983

## AVANT-PROPOS.

Je tiens à remercier tous ceux qui m'ont aidé durant mes études et lors de la réalisation du présent mémoire.

Je remercie tout d'abord mes parents qui m'ont donné la possibilité de réaliser les études que j'avais choisies et qui m'ont soutenu pendant les moments difficiles.

Je remercie ensuite Monsieur Roland Lesuisse, promoteur de ce mémoire, pour sa constante disponibilité et ses conseils qui ont permis la réalisation de celui-ci.

Je remercie également Messieurs Jean-Luc Hainaut et Axel van Lamsweerde pour leurs conseils judicieux.

J'exprime enfin ma reconnaissance à Mademoiselle Béatrice Scoyer pour sa collaboration lors de l'édition de ce mémoire.

TABLE DES MATIERES.

# Chapitre 0: INTRODUCTION.

"TIME IS MONEY !" disent les anglo-saxons...

Ce dicton, si souvent entendu, est à la base de l'insertion d'outils informatiques dans les bureaux (aujourd'hui appelés "systèmes de bureautique").

En effet, une bonne occupation du temps de travail est une condition fondamentale de la productivité des travailleurs de bureau.

Parmi les activités qui absorbent leur temps de travail, il en est une particulière : celle pendant laquelle les travailleurs de bureau gèrent leur temps de travail.

Ainsi, certaines études ont été réalisées afin de déterminer le temps pendant lequel la direction et la secrétaire sont occupées à gérer leur temps de travail (planification de réunions, rendez-vous, etc...). En voici quelques résultats tirés de [ENG80] :

- une secrétaire passe 2,6 % de son temps de travail à la tenue d'agendas;
- un directeur de haut niveau passe 13,1 % de son temps de travail à fixer des réunions et 8,5 % à déplacer des réunions.
- un directeur de plus bas niveau passe 3,8 % de son temps de travail à fixer des réunions et 3,4 % à déplacer des réunions.

Pour résoudre ce problème, on se propose de mettre au point un système d'agendas électronique (ou gestionnaire d'agendas) qui, pour être accepté dans une entreprise, devra répondre à deux conditions fondamentales :

1. Il devra être facile à utiliser. En effet, l'agenda manuel présente le grand avantage d'être facile à manipuler : on le feuillette, on insère l'information au crayon, on gomme, on l'emporte avec soi,... Ces gestes habituels nécessitent peu de réflexion ! Ainsi un système d'agendas électronique ne sera vraiment apprécié que lorsqu'il répondra à un maximum de facilités proches de celles de l'agenda manuel comme l'explique très judicieusement Robert B. White : "We can't expect people to accept a system that changes their responsabilities or complicates their lives. If we make it more difficult for a manager to look at and modify his calendar electronically than manually, he just won't use the system."

2. Il devra conserver un grand degré de confidentialité. En effet, il s'avère que les responsables, surtout de haut niveau, sont apparemment très réticents à ce que l'occupation de leur temps soit connue de tous.

Le plan de notre travail se présentera, dès lors, ainsi :

1. Spécifications fonctionnelles.
2. Description et implémentation des données.
3. Description des traitements.
4. Manuel utilisateur.
5. Conclusions.

# Chapitre 1: SPECIFICATIONS FONCTIONNELLES.

## 1. Introduction.

On veut mettre ici en relief toutes les fonctions que doit remplir un gestionnaire d'agendas pour répondre aux souhaits des personnes destinées à s'en servir.

A cet effet, on définit d'abord quelques concepts fondamentaux : système d'agendas, abonné, administrateur, ressource, agenda, activité; ensuite, on essaie de déterminer les différentes fonctions à mettre en oeuvre pour consulter et modifier un ou plusieurs agendas.

## 2. Concepts fondamentaux.

### 2.1. Système d'agendas.

On appelle système d'agendas l'ensemble des agendas des abonnés et des ressources connues du système.

Le système d'agendas est gérè par un administrateur et est accessible à tout abonnè.

### 2.2. Abonnè.

On appelle abonnè toute personne reconnue par le gestionnaire d'agendas à l'aide d'un nom et d'un mot de passe, et possèdant un agenda dans le système.

### 2.3. Administrateur.

On appelle administrateur l'abonnè ayant pour tâche de gérer le système d'agendas.

Il a accès à l'ensemble des agendas des abonnés et des ressources connues du système, et est le seul abonnè ayant la possibilitè d'ajouter un abonnè ou une ressource ou de changer le responsable d'une ressource.

### 2.4. Ressource.

On appelle ressource tout objet dont le temps d'utilisation peut être gèrè par le gestionnaire d'agendas.

Exemples : un local, un sèminaire, un rètro-projecteur, un magnètophone, un ordinateur,...

Toute ressource connue du système aura pour responsable un abonnè.

### 2.5. Agenda.

On appelle agenda d'un abonnè ou d'une ressource l'ensemble des activitès concernant cet abonnè ou consommant cette ressource.

Un agenda est caractèrisè par :
- le nom de l'abonnè ou de la ressource,
- la liste d'accès pour les agendas appartenant à un abonnè.

L'agenda d'un abonné est accessible à celui-ci ainsi qu'aux abonnés appartenant à la liste d'accès de l'agenda.

Exemple : la liste d'accès de l'agenda de M. Durand = 'Dupont', 'Peeters', 'Bouvier' ==> MM. Dupont, Peeters, Bouvier et Durand peuvent consulter l'agenda de ce dernier.

L'agenda d'une ressource est accessible à tout abonné.

## 2.6. Activité.

On appelle activité une tâche concernant un ou plusieurs abonnés et/ou consommant une ou plusieurs ressources pendant un laps de temps déterminé.

Une activité se caractérise par :

- la date de cette activité (sous la forme jour/mois/année);
- l'heure de début (sous la forme heure.minute);
- la durée estimée (sous la forme heure.minute);
- un mot-clé déterminant son type (exemples : réunion, conférence, cours, privé, étude,...);
- un texte, indiquant la nature de l'activité;
- un caractère de confidentialité : la valeur de ce caractère marque que l'activité est soit confidentielle (uniquement visible par les abonnés concernés par l'activité), soit publique (visible par tout abonné ayant acces à l'un des agendas des abonnés concernés);
- le nom de la personne ayant enregistré cette activité, c'est-à-dire de l'abonné qui est en tout premier lieu concerné par l'activité;
- le nom de la personne responsable de cette activité, c'est-à-dire de l'abonné qui, avec la personne ayant enregistré l'activité, peut la consulter, la modifier ou la supprimer à loisir;
- une liste des personnes concernées, c'est-à-dire un ensemble d'abonnés qui, en plus de la personne ayant enregistré l'activité et de la personne responsable, sont également concernés par cette activité;
- une liste des codes-réponses : à chaque personne de la liste ci-dessus, correspond un code-réponse qui indique si l'activité est acceptée ou non par celle-ci;
- une liste des ressources consommées, c'est-à-dire un ensemble de ressources connues du système qui sont consommées par cette activité.

Exemple :

M. Dupont a enregistrè une rèunion organisèe par M. Durand et concernant MM. Bouvier et Larivière. Cette rèunion est confidentielle et a lieu dans le local 01 le 1er octobre 1983 de 10 h 30 à 12 h.

```
date = '1/10/83'
heure = '10.30'
durèe = '1.30'
type = 'rèunion'
texte = 'dèbat sur les systèmes experts'
caractère de confidentialitè = 'X'
nom de la personne ayant enregistrè l'activitè = 'Dupont'
nom de la personne responsable = 'Durand'
liste des personnes concernèes = 'Bouvier', 'Larivière'
liste des codes rèponses = ' ', ' '
liste des ressources consommèes = 'local 01'
```

Supposons qu'après avoir pris connaissance de cette activitè, M. Bouvier ait acceptè l'invitation tandis que M. Larivière, ètant dèjà occupè pendant cette pèriode, n'ait pu l'accepter; alors la liste des codes sera modifièe ainsi :

```
liste des codes rèponses = 'O', 'N'
```

## 3. Fonctions de consultation d'un agenda.

### 3.1. Vue d'un jour.

Cette fonction permet de visualiser de facon détaillée les activités d'un abonné ou d'une ressource connue du système correspondant à un type donné (facultatif) et à une date donnée.

Elle est accessible à tout abonné se trouvant dans la liste d'accès de l'agenda s'il s'agit de l'agenda d'un abonné, et par tout abonné s'il s'agit de l'agenda d'une ressource.

Exemple :

M. Dupont consulte son agenda à la date du 1er octobre 1983 et y sélectionne ses réunions.

Dupont réunion le samedi 1 octobre 1983

| heure | durée | description |
|---|---|---|
| 10.30 | 1.30 | débat sur les systèmes experts |
| 16.00 | 3.00 | réunion avec M. Lebrun |

### 3.2. Vue d'une semaine.

Cette fonction permet de visualiser les moments d'une semaine donnée pendant lesquels un abonné ou une ressource connue du système est occupée, à un type d'activité donné (facultatif).

Elle est accessible à tout abonné se trouvant dans la liste d'accès de l'agenda s'il s'agit de l'agenda d'un abonné, et par tout abonné s'il s'agit de l'agenda d'une ressource.

Exemple :

M. Dupont consulte son agenda à la semaine du 1er octobre 1983.

Dupont — du samedi 1 octobre 1983 au vendredi 7 octobre 1983

| heure | samedi | dimanche | lundi | mardi | mercredi | jeudi | vendredi |
|---|---|---|---|---|---|---|---|
| 0. | | | | | | | |
| 2. | | | | | | | |
| 4. | | | | | | | |
| 6. | | | | | | | |
| 8. | | | | | | | |
| 10. | xxxxxx | | | xxxxxx | | | xxxx |
| 12. | | | | | | | |
| 14. | | | | | | | |
| 16. | xxxxxxxx | | | | xxxx | | |
| 18. | xxxx | | | | | | |
| 20. | | | | | | | |
| 22. | | | | | | | |

x = 1/4 h d'occupation

## 3.3. Vue d'un mois.

Cette fonction permet de visualiser les jours d'un mois donné pendant lesquels un abonné ou une ressource connue du système est occupée à au moins une activité, d'un type donné (facultatif).

Elle est accessible à tout abonné se trouvant dans la liste d'accès de l'agenda s'il s'agit de l'agenda d'un abonné, et par tout abonné s'il s'agit de l'agenda d'une ressource.

Exemple :

M. Dupont consulte son agenda au mois d'octobre 1983.

Dupont — octobre 1983

| samedi | dimanche | lundi | mardi | mercredi | jeudi | vendredi |
|---|---|---|---|---|---|---|
| 1 X | 2 | 3 | 4 X | 5 X | 6 | 7 X |
| 8 | 9 | 10 X | 11 | 12 | 13 | 14 |
| 15 | 16 X | 17 | 18 | 19 X | 20 | 21 |
| 22 | 23 | 24 | 25 X | 26 | 27 X | 28 |
| 29 | 30 X | 31 | | | | |

### 3.4. Visualisation des nouvelles activités.

Cette fonction permet à tout abonné de visualiser successivement les activités le concernant et pour lesquelles il n'aurait pas encore donné de réponse.

Exemple :

M. Bouvier est mis au courant de la réunion proposée par M. Dupont.

```
activité communiquée par : Dupont
date : samedi 1 octobre 1983
heure : 10 h 30          durée : 1 h 30
type : réunion
description : débat sur les systèmes experts
responsable : Durand
```

## 4. Fonctions de consultation d'un ensemble d'agendas.

### 4.1. Recherche de plages de temps libre.

Cette fonction permet à tout abonné de visualiser les plages de temps libre pendant une semaine donnée pour un ensemble de personnes et/ou de ressources connues du système.

Exemple :

M. Lebrun visualise les moments de temps libre de MM. Bouvier et Dupont pour la semaine du 1er octobre 1983.

Bouvier Dupont — du samedi 1 octobre 1983 au vendredi 7 octobre 1983

| heure | samedi | dimanche | lundi | mardi | mercredi | jeudi | vendredi |
|---|---|---|---|---|---|---|---|
| 0. | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx |
| 2. | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx |
| 4. | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx |
| 6. | xxxxxxxx | xxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx |
| 8. | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx |
| 10. | xx | xxxxxxxx | xxxxxxxx | xx | xxxxxxxx | | xxxx |
| 12. | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | | xxxxxxxx |
| 14. | xxxxxxxx | xxxxxxxx | | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx |
| 16. | | xxxxxxxx | | xxxxxx | xxxx | xxxxxxxx | xxxxxxxx |
| 18. | xxxx | xxxxxxxx | xxxxxxxx | xx | xxxxxxxx | xxxxxxxx | xxxxxxxx |
| 20. | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx |
| 22. | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx |

x = 1/4 h de temps libre pour l'ensemble d'abonnés et/ou de ressources

### 4.2. Consultation des réponses à une proposition d'activité.

Cette fonction permet à toute personne ayant enregistré une activité, ainsi qu'à la personne responsable, de visualiser la liste des autres personnes concernées par cette activite ainsi que leur code réponse.

Exemple :

M. Dupont consulte les réponses de MM. Bouvier et Larivière à sa proposition d'activité.

liste des personnes concernées :
Bouvier O Larivière N

## 5. Fonctions de modification d'agenda(s).

### 5.1. Ajout d'une activité.

Cette fonction permet à tout abonné d'inscrire une nouvelle activité dans les agendas en indiquant :

- la date;
- l'heure;
- la durée;
- le type (facultatif);
- le texte descriptif (facultatif);
- le nom de la personne responsable (facultatif);
- la liste des personnes concernées (facultatif);
- la liste des ressources consommées (facultatif);

Exemple :

M. Dupont enregistre une réunion avec M. Lebrun à la date du 1er octobre 1983 de 16 à 19 h.

```
date = '1/10/83'
heure = '16.'
durée = '3.'
type = 'réunion'
texte = 'réunion avec Mr. Lebrun'
liste des personnes concernées = 'Lebrun'
```

### 5.2. Ajout d'une activité en série.

Cette fonction permet à tout abonné d'inscrire une série de nouvelles activités dans les agendas en indiquant :

- la date;
- l'heure;
- la durée;
- le type (facultatif);
- le texte descriptif (facultatif);
- le nom de la personne responsable (facultatif);
- la liste des personnes concernées (facultatif);
- la liste des ressources consommées (facultatif);
- le nombre de fois qu'il faut répéter l'activité (en jours);
- le nombre de jours séparant deux de celles-ci.

Exemple :

M. Dupont enregistre une série de 3 cours de "structure des ordinateurs" qu'il donnera de 10 à 12 h dans le local 08 les lundis 3, 10 et 17 octobre 1983.

```
date = '3/10/83'
heure = '10.'
durée = '2.'
type = 'cours'
texte = 'structure des ordinateurs'
liste des ressources consommées = 'local 08'
nombre de fois qu'il faut répéter l'activité = '2'
nombre de jours séparant deux de celles-ci = '7'
```

5.3. Modification d'une activité.

Cette fonction permet à tout abonné de modifier une activité qu'il a lui-même enregsitrée ou de laquelle il est responsable

Peuvent être modifiés :

- la date;
- l'heure;
- la durée;
- le type;
- le texte descriptif;
- le nom de la personne responsable;
- la liste des personnes concernées;
- la liste des ressources consommées;

Exemple :

M. Dupont invite M. Larivière en plus de M. Lebrun à la réunion du 1er octobre 1983.

```
liste des personnes concernées = 'Lebrun', 'Larivière'
```

5.4. Suppression d'une activité.

Cette fonction permet à tout abonné de supprimer une activité qu'il a lui-même enregistrée ou de laquelle il est responsable.

Exemple :

M. Durand, responsable de l'activité du 1er octobre 1983 enregistrée par M. Dupont, annule celle-ci.

L'activité est retirée des agendas de MM. Dupont, Durand, Bouvier et Larivière.

## 6. Fonctions de consultation liées au système d'agendas.

### 6.1. Visualisation de la liste des abonnés.

Cette fonction permet à tout abonné de visualiser la liste des abonnés présents dans le système ainsi que la liste des abonnés pouvant accéder à son agenda.

Exemple :

M. Durand visualise la liste des abonnés.

```
LISTE DES ABONNES
nom       ! abonnés y ayant accès
----------!-----------------------------------
Felingue  !
Dupont    ! Felingue Durand  Bouvier   Lebrun
Durand    ! Felingue Bouvier
Bouvier   ! Felingue Durand  Larivière
Lebrun    ! Felingue Bouvier Larivière
Larivière! Felingue Bouvier Durand    Lebrun
```

### 6.2. Visualisation de la liste des ressources.

Cette fonction permet à tout abonné de visualiser la liste des ressources connues du système ainsi que le nom de leur responsable respectif.

Exemple :

M. Dupont visualise la liste des ressources.

```
LISTE DES RESSOURCES
nom       ! responsable
----------!-------------
local 01  ! Dupont
local 02  ! Dupont
local 03  ! Dupont
pdp       ! Durand
dec       ! Bouvier
```

## 7. Fonctions de modification liées au système d'agendas.

### 7.1. Ajout d'un abonné.

Cette fonction permet à l'administrateur d'ajouter un abonné au système en lui spécifiant son nom, son mot de passe et la liste des personnes qui pourront accéder à son agenda (celles-ci devant être des abonnés).

Exemple :

M. Felingue, administrateur du système d'agendas, ajoute M. Geerts à la liste des abonnés.

```
nom = 'Geerts'
mot de passe = 'voiture'
liste des abonnés ayant accès à son agenda =
'Lebrun', 'Dupont', 'Larivière', 'Bouvier'
```

### 7.2. Modification d'un mot de passe.

Cette fonction permet à tout abonné de modifier son mot de passe.

Exemple :

M. Geerts modifie son mot de passe.

```
nouveau mot de passe = 'renault'
```

### 7.3. Modification de la liste d'accès.

Cette fonction permet à tout abonné de modifier la liste des abonnés ayant accès à son agenda.

Exemple :

M. Geerts modifie la liste des abonnés ayant accès à son agenda.

```
ancienne liste des abonnés ayant accès à son
agenda = Lebrun, Dupont, Larivière, Bouvier
nouvelle liste des abonnés ayant accès à son
agenda = 'Lebrun', 'Larivière', 'Durand'
```

## 7.4. Ajout d'une ressource.

Cette fonction permet à l'administrateur d'ajouter une ressource à l'ensemble des ressources connues du système en lui spécifiant son nom et le nom de son responsable (celui-ci devant être un abonné).

Exemple :

M. Felingue, administrateur du système d'agendas, ajoute un rétro-projecteur dans la liste des ressources. M. Dupont en sera responsable.

```
nom = 'rétro 03'
nom du responsable = 'Dupont'
```

## 7.5. Modification d'un responsable.

Cette fonction permet à l'administrateur de modifier le nom du responsable d'une ressource connue du système.

Exemple :

M. Felingue, administrateur du système d'agendas, change le responsable du rétro-projecteur no. 3. M. Durand en est le nouveau responsable.

```
nom = 'rétro 03'
ancien responsable = Dupont
nouveau responsable = 'Durand'
```

# Chapitre 2: DESCRIPTION ET IMPLEMENTATION DES DONNEES.

## 1. Introduction.

On décrit ici les données globales nécessaires à notre application.

La structure de ces données (ou schéma conceptuel) est tout d'abord présentée selon le modèle "entité - association".

Les différents éléments du schéma conceptuel sont ensuite définis; l'ensemble de ces définitions forme le dictionnaire des données.

Enfin, on reprend le schéma conceptuel et on en déduit un schéma des accès logiques qu'on convertit en fichiers.

# 2. Schéma conceptuel.

## 2.1. Concepts fondamentaux.

Le modèle "entité - association" utilise les concepts suivants : entité, relation, propriété et contraintes d'intégrité [BOD81].

### 2.1.1. L'entité.

L'entité est ce qu'on percoit comme un tout ayant une existence propre. Une entité est caractérisée par un ensemble de propriétés quantitatives ou qualitatives.

Le type d'entités est l'ensemble des entités caractérisées par les mêmes propriétés.

Chaque occurrence d'un type d'entités doit être unique et pour cela, il faut la distinguer des autres occurrences de ce même type d'entité.

La ou les propriété(s) permettant cette distinction est (sont) appelée(s) identifiante(s).

On conviendra qu'un type d'entités est représenté par son nom en caractères majuscules dans un rectangle. Dans celui-ci, figure également le nom des propriétés, de ce type, en caractères minuscules.

### 2.1.2. La relation.

La relation est définie comme une association entre entités où chacune assume un rôle donné. Une relation peut aussi avoir des propriétés.

Le type de relations est l'ensemble des relations caractérisées par les mêmes entités. Une occurrence du type de relations implique toujours une occurrence de chaque type d'entités définissant ce type de relations.

On conviendra qu'un type de relations est représenté par un hexagone dans lequel figure son nom en caractères majuscules et celui de ses propriétés en caractères minuscules. On relie le type de relations aux types d'entités qui le composent par des arcs.

### 2.1.3. La propriété.

Une propriété appartenant à une entité ou à une relation est une qualité de cette entité ou de cette relation.

L'existence d'une propriété est contingente à l'existence de l'entité ou de la relation concernée.

### 2.1.4. Les contraintes d'intégrité.

Les contraintes d'intégrité sont des prédicats relatifs aux éléments du schéma conceptuel qui doivent être vérifiés pendant une période de temps donné. On distingue les contraintes d'intégrité statiques qui doivent être vraies à tout moment et les contraintes d'intégrité dynamiques qui doivent être vérifiées durant des périodes données. Un certain nombre de ces contraintes peuvent être exprimées sur le schéma conceptuel, les contraintes restantes seront énoncées en francais. Les contraintes intégrées dans le modèle sont de deux types : les contraintes de cardinalité et les contraintes d'existence.

#### 2.1.4.1. Les contraintes de cardinalité.

Pour un type de relation défini entre les types d'entités X et Y, on peut avoir :

- une relation (1 - 1), dans laquelle à une occurrence de X correspond toujours exactement une occurrence de la relation vers exactement une occurrence de Y, et inversément.
- une relation (1 - N), dans laquelle à une occurrence de X peut correspondre une ou plusieurs relations, chacune vers exactement une occurrence de Y, et à chaque occurrence de Y ne correspond qu'exactement une relation vers une occurrence de X.
- une relation (M - N), dans laquelle à une occurrence de X peut correspondre une ou plusieurs relations, chacune vers une occurrence de Y, et inversément.

#### 2.1.4.2. Les contraintes d'existence.

Un type de relations est fort pour un type d'entités si une entité de celui-ci ne peut exister que si elle appartient à, au moins, une relation de ce type.

Un type de relations est faible pour un type d'entité si l'existence d'une entité de ce type est indépendante de toute correspondance à une relation de ce type.

2.1.4.3. La combinaison de ces deux types de contraintes.

Pour tout type d'entité participant à un type de relation, il est possible de regrouper les contraintes d'existence et de cardinalité qui s'y rapportent.

Cela se fait par le couple (c-min, c-max) où :

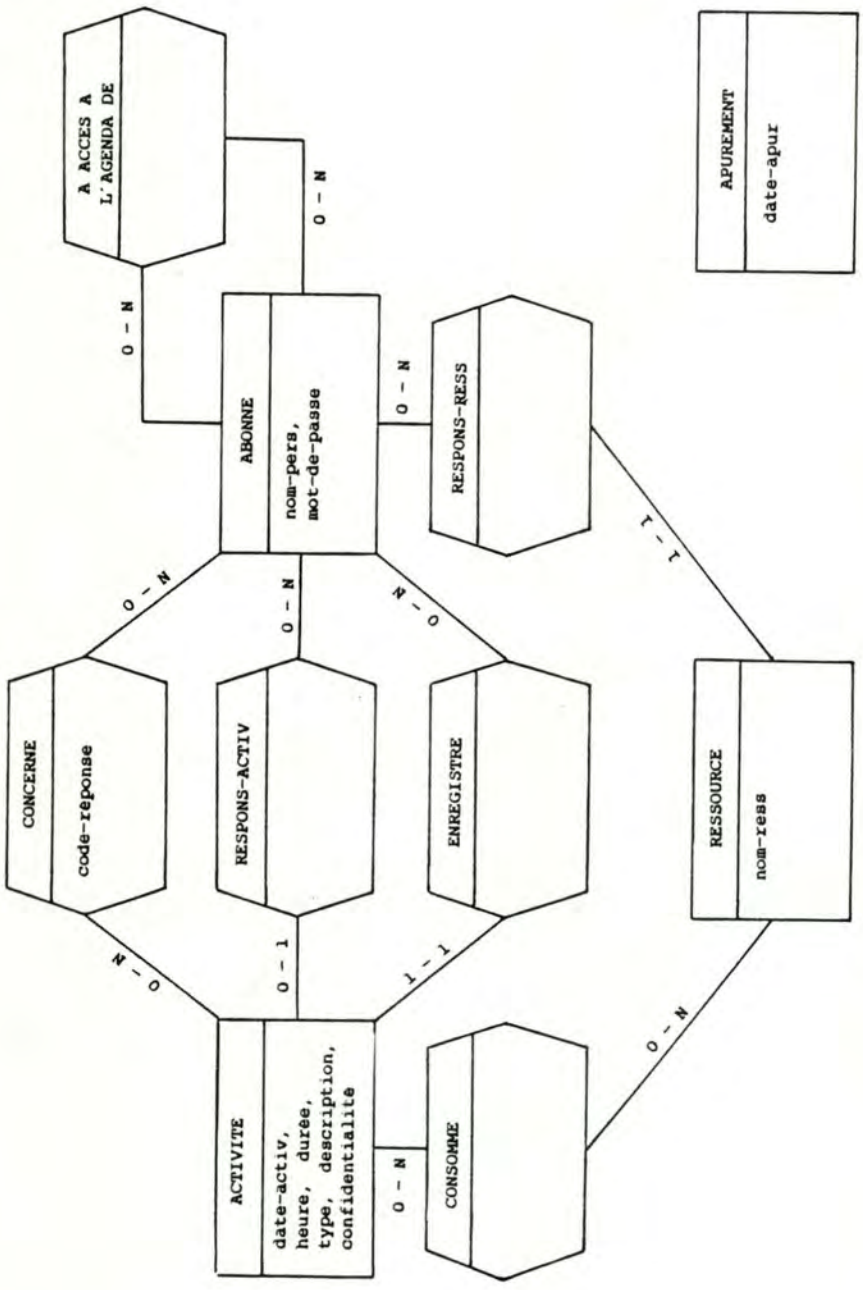
- c-min est le nombre minimal de fois que chaque occurrence de ce type d'entités peut intervenir dans une occurrence du type de relations; si la relation est forte pour ce type d'entités alors c-min vaut '1', si la relation est faible alors c-min vaut '0'.
- c-max est le nombre maximal de fois que chaque occurrence de ce type d'entités peut intervenir dans une occurrence du type de relations; c-max représente donc les contraintes de cardinalité.

L'ensemble des couples (c-min, c-max) des types d'entités participant à un type de relations expriment la connectivité du type d'associations.

Une connectivité est indiquée au niveau des arcs reliant un hexagone (relation) à un rectangle (entité).

## 2.2. Modèle des données.

Selon le formalisme du modèle "entité - association"et à partir des spécifications fonctionnelles décrites au chapitre 1, on peut donc construire le modèle des données suivant :



## 3. Dictionnaire des données.

On se propose de décrire à présent les différents concepts apparus dans notre schéma conceptuel. A cet effet, on aura recours au langage DSL développé dans le cadre du projet ISDOS [BOD80].

### 3.1. Les entités.

#### 3.1.1. L'entité "ACTIVITE".

```
DEFINE ENTITY activité;

RELATED TO abonné VIA enregistre,
           abonné VIA respons-activ,
           abonné VIA concerne,
           ressource VIA consomme;

CONSISTS OF date-activ,
            heure,
            durée,
            type,
            description,
            confidentialité;

DESCRIPTION;
```

Une occurrence de l'entité "activité" désigne une tâche concernant un ou plusieurs abonnés et/ou consommant une ou plusieurs ressources pendant un laps de temps déterminé.

#### 3.1.2. L'entité "ABONNE".

```
DEFINE ENTITY abonné;

RELATED TO activité VIA enregistre,
           activité VIA respons-activ,
           activité VIA concerne,
           ressource VIA respons-ress,
           abonné VIA a accès à l'agenda de;

CONSISTS OF nom-pers,
            mot-de-passe;

IDENTIFIED BY nom-pers;
```

DESCRIPTION;

Une occurrence de l'entité "abonné" désigne une personne qui a le droit d'utiliser le gestionnaire d'agendas et qui est reconnue par celui-ci à l'aide d'un nom et d'un mot de passe; cette personne a son agenda dans le système.

### 3.1.3. L'entité "RESSOURCE".

DEFINE ENTITY ressource;

RELATED TO activité VIA consomme,
abonné VIA respons-ress;

CONSISTS OF nom-ress;

IDENTIFIED BY nom-ress;

DESCRIPTION;

Une occurrence de l'entité "ressource" désigne un objet dont le temps d'utilisation peut être géré par le gestionnaire d'agendas.

### 3.1.4. L'entité "APUREMENT".

DEFINE ENTITY apurement;

CONSISTS OF date-apur;

DESCRIPTION;

L'occurrence de l'entité "apurement" désigne la date du dernier apurement du système d'agendas.

## 3.2. Les relations.

### 3.2.1. La relation "ENREGISTRE"

```
DEFINE RELATION enregistre;

RELATES activitè, abonnè;

CONNECTIVITY IS 1-1 FOR activitè,
                0-n FOR abonnè;

DESCRIPTION;
```

Une occurrence de la relation "enregistre" représente le fait qu'une activité est enregistrée par un abonné dans un ou plusieurs agendas.

### 3.2.2. La relation "RESPONS-ACTIV".

```
DEFINE RELATION respons-activ;

RELATES activitè, abonnè;

CONNECTIVITY IS 0-1 FOR activitè,
                0-n FOR abonnè;

DESCRIPTION;
```

Une occurrence de la relation "respons-activ" représente le fait qu'un abonné est responsable d'une activité; c'est-à-dire qu'il peut, tout comme la personne ayant enregistré l'activité, la consulter, la modifier ou la supprimer.

### 3.2.3. La relation "CONCERNE".

```
DEFINE RELATION concerne;

RELATES activitè, abonnè.

CONSISTS OF code-rèponse;

CONNECTIVITY IS 0-n FOR activitè,
                0-n FOR abonnè;
```

DESCRIPTION.

Une occurrence de la relation "concerne" représente le fait qu'une activité concerne un abonné; c'est-à-dire qu'une activité lui est proposée, et "code-réponse" représente l'accord ou le non-accord de l'abonné à participer à l'activité.

### 3.2.4. La relation "CONSOMME".

```
DEFINE RELATION consomme;

RELATES activité, ressource;

CONNECTIVITY IS 0-n FOR activité,
                0-n FOR ressource;

DESCRIPTION;
```

Une occurrence de la relation "consomme" représente le fait qu'une activité consomme une ressource; c'est-à-dire que celle-ci est indisponible pour toute autre activité pendant la durée de la première.

### 3.2.5. La relation "RESPONS-RESS".

```
DEFINE RELATION respons-ress;

RELATES ressource, abonné;

CONNECTIVITY IS 1-1 FOR ressource,
                0-n FOR abonné;

DESCRIPTION;
```

Une occurrence de la relation "respons-ress" représente le fait qu'un abonné est responsable d'une ressource.

### 3.2.6. La relation "A ACCES A L'AGENDA DE".

DEFINE RELATION a accès à l'agenda de;

RELATES abonné, abonné;

CONNECTIVITY 0-n FOR abonné;

DESCRIPTION;

Une occurrence de la relation "a accès à l'agenda de" représente le fait qu'un abonné a la permission de connaître l'emploi du temps d'un autre abonné, sauf éventuellement l'emploi du temps pendant les heures protégées.

## 3.3. Les informations élémentaires.

### 3.3.1. L'élément "date-activ".

```
DEFINE ELEMENT date-activ;

CONTAINED IN activité;

FORMAT IS 9(6);

CODE aa MEANS 80 THRU 99,
     mm MEANS 1 THRU 12,
     jj MEANS 1 THRU 31;

DESCRIPTION;
```

Date d'une activité sous la forme AAMMJJ.
Exemple : 831001 (1 octobre 1983).

### 3.3.2. L'élément "heure".

```
DEFINE ELEMENT heure;

CONTAINED IN activité;

FORMAT IS 9(4.2);

CODE hh MEANS 0 THRU 23,
     mm MEANS 0 THRU 59;

DESCRIPTION;
```

Heure de début d'une activité sous la forme HH.MM.
Exemple : 14.08 (14 h 8 min).

### 3.3.3. L'élément "durée".

```
DEFINE ELEMENT durée;

CONTAINED IN activité;

FORMAT IS 9(4.2);

CODE hh MEANS 0 THRU 23,
     mm MEANS 0 THRU 59;

DESCRIPTION;
```

Durée estimée d'une activité sous la forme HH.MM.
Exemple : 2.10 (2 h 10 min).

### 3.3.4. L'élément "type".

DESCRIPTION ELEMENT type;

CONTAINED IN activité;

FORMAT IS X(8);

DESCRIPTION;

Mot-clé déterminant le type d'une activité. L'ensemble des différents types d'activités n'est pas prédéfini, il est sous la responsabilité de chaque abonné.
Exemple : réunion.

### 3.3.5. L'élément "description".

DEFINE ELEMENT description;

CONTAINED IN activité;

FORMAT IS X(128);

DESCRIPTION;

Texte libre indiquant la nature de l'activité.

### 3.3.6. L'élément "confidentialité".

DEFINE ELEMENT confidentialité;

CONTAINED IN activité;

FORMAT IS X(1);

VALUES ARE ' ', 'X';

DESCRIPTION;

Caractère de confidentialité valant :
- ' ' si l'activité est visible par tout abonné ayant accès à l'un des agendas des abonnés concernés,
- 'X' si l'activité est uniquement visible par les abonnés concernés par l'activité.

### 3.3.7. L'élément "nom-ab".

DEFINE ELEMENT nom-ab;

CONTAINED IN abonné;

IDENTIFIES abonné;

FORMAT IS X(8);

DESCRIPTION;

Nom d'un abonné au système (patronyme ou nom donné par l'administrateur).

### 3.3.8. L'élément "mot-de-passe".

DEFINE ELEMENT mot-de-passe;

CONTAINED IN abonné;

FORMAT IS X(8);

DESCRIPTION;

Mot de passe qui permet au système d'identifier univoquement un abonné.
Ce mot de passe est choisi par l'abonné lors de son entrée dans le système et peut être modifié par celui-ci à tout moment.

### 3.3.9. L'élément "nom-ress".

DEFINE ELEMENT nom-ress;

CONTAINED IN ressource;

IDENTIFIES ressource;

FORMAT IS X(8);

DESCRIPTION;

Nom d'une ressource connue du système.
Exemple : local 03.

### 3.3.10. L'élément "date-apur".

```
DEFINE ELEMENT date-apur;

CONTAINED IN apurement;

FORMAT IS 9(6);

CODE aa MEANS 80 THRU 99,
     mm MEANS 1 THRU 12,
     jj MEANS 1 THRU 31;

DESCRIPTION;
```

Date du dernier apurement du sytème sous la forme **AAMMJJ**.

### 3.3.11. L'élément "code-réponse".

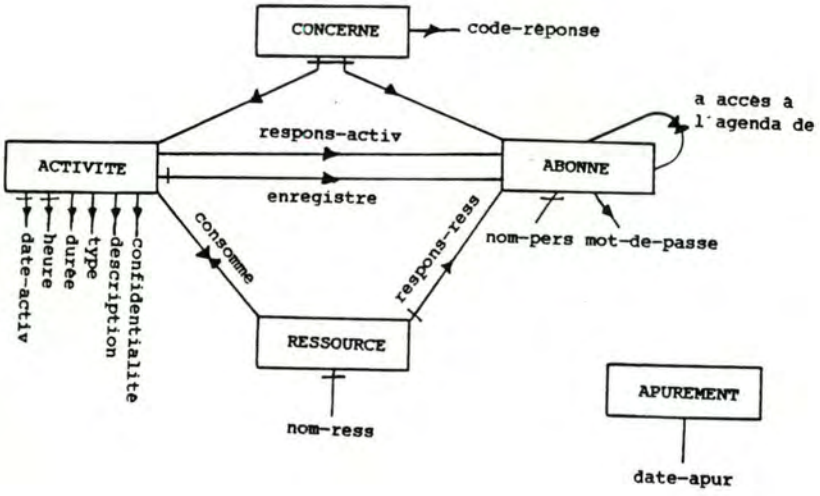
```
DEFINE ELEMENT code-réponse;

CONTAINED IN concerne;

FORMAT IS X(1);

VALUES ARE ' ', 'O', 'N';

DESCRIPTION;
```

Code réponse d'un abonné à une proposition d'activité valant :
- ' ' si l'abonné n'a pas encore répondu,
- 'O' si l'abonné a répondu affirmativement,
- 'N' si l'abonné a répondu négativement.

## 4. Implémentation du schéma conceptuel.

On reprend maintenant le schéma conceptuel décrit en 2.2. pour le convertir en un modèle conceptuel binaire, base du modèle d'accès.

Tenant compte des règles de conversion décrites dans [HAI80], le modèle conceptuel binaire est le suivant :

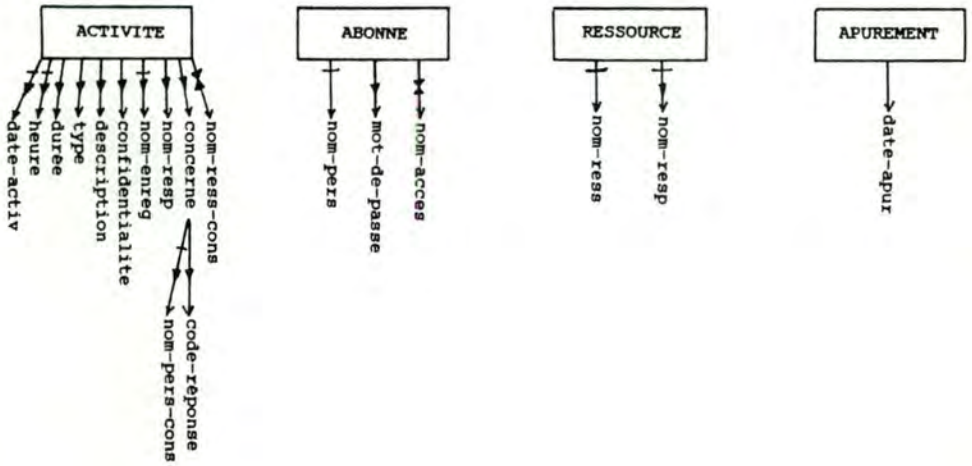


Notre application étant écrite en PASCAL, la base de données est exclusivement constituée de fichiers séquentiels.

Les caractéristiques principales d'un tel modèle d'accès sont les suivantes :

- un fichier ne peut contenir différents types d'enregistrements,
- un fichier peut être trié,
- les concepts de clé d'accès et d'identifiant ne jouent pas,
- les rubriques caractérisant un enregistrement sont :
  - obligatoires ou facultatives,
  - décomposables ou élémentaires,
  - simples ou répétitives (répétition limitée).

Ces contraintes entraînent une transformation du modèle ci-dessus pour donner le schéma d'accès suivant :



Cette nouvelle transformation met en évidence les fichiers PASCAL suivants :

- le fichier ACTIVITE :
  ```
  date-activ : real
  heure : real
  durée : real
  type : string (8)
  description : string (128)
  confidentialité : string (1)
  nom-enreg : string (8)
  nom-resp : string (8)
  nom-pers-cons : array [1..10] of string (8)
  code-réponse : array [1..10] of string (1)
  nom-ress-cons : array [1..4] of string (8)
  ```

- le fichier ABONNE :
  ```
  nom-pers : string (8)
  mot-de-passe : string (8)
  nom-accès : array [1..10] of string (8)
  ```

- le fichier RESSOURCE :
  ```
  nom-ress : string (8)
  nom-resp : string (8)
  ```

- le fichier APUREMENT :
  ```
  date-apur : real
  ```

# Chapitre 3: DESCRIPTION DES TRAITEMENTS.

## 1. Introduction.

On décrit ici la structure des traitements de notre application.

Deux architectures générales sont d'abord définies : la première concerne les traitements sur les agendas et la seconde les traitements relatifs aux abonnés et ressources.

On précise ensuite ces deux architectures en indiquant les procédures utilisées à chaque niveau. Chaque niveau devient plus précis au fur et à mesure que l'on descend dans la hiérarchie :

- le premier niveau est constitué des deux procédures coordinatrices;
- le second niveau comprend les procédures appelées par leur procédure coordinatrice respective;
- les niveaux 3 à 7 comprennent des procédures qui découlent d'une découpe de plus en plus fine;
- le huitième niveau comprend les procédures d'affichage d'erreurs;
- le neuvième niveau comprend les procédures d'accès aux fichiers.

Enfin, on définit pour chaque procédure les entrées, les sorties, les données globales, l'effet général de la procédure, les pré-conditions et les post-conditions, les procédures appelantes et les procédures appelées, l'algorithme logique et la spécification des variables internes :

- les entrées reprennent la liste des paramètres passés à la procédure ainsi que leur définition;

- les sorties reprennent la liste des paramètres renvoyés par la procédure ainsi que leur définition;

- les données globales reprennent le nom symbolique des variables utilisées par la procédure courante et définies par une procédure du premier niveau ainsi que leur définition.

- l'effet reprend de façon générale ce que fait la procédure;

- les pré-conditions sont les conditions sur les entrées qui doivent être satisfaites pour que toute exécution de la procédure soit correcte;

- les post-conditions sont les conditions qui doivent être satisfaites après toute exécution de la procédure et qui décrivent l'effet attendu de la procédure;

- les procédures appelantes sont les procédures qui utilisent la procédure courante;

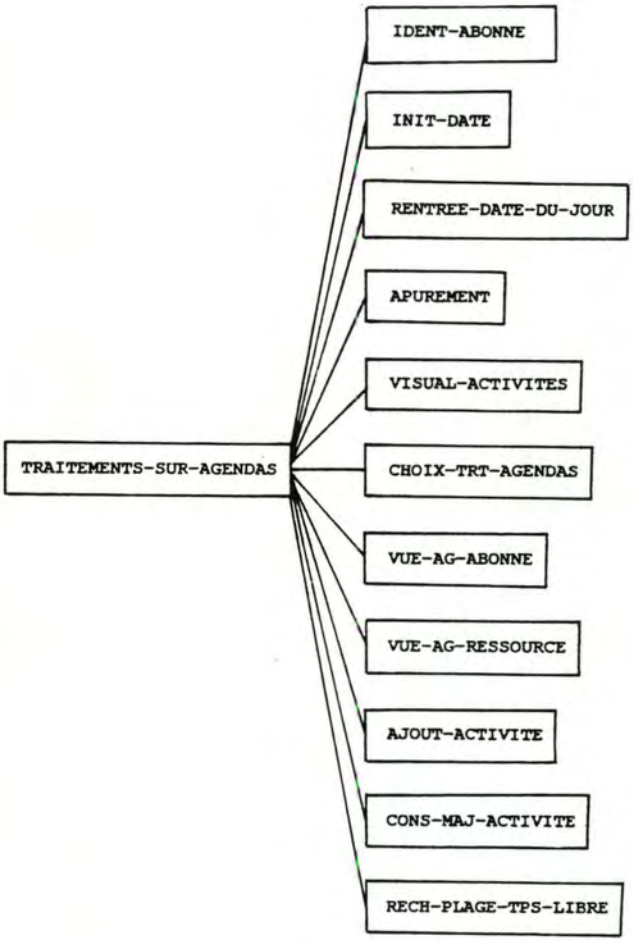
- les procédures appelées sont les procédures utilisées par la procédure courante;

- l'algorithme logique exprime la logique de la procédure;

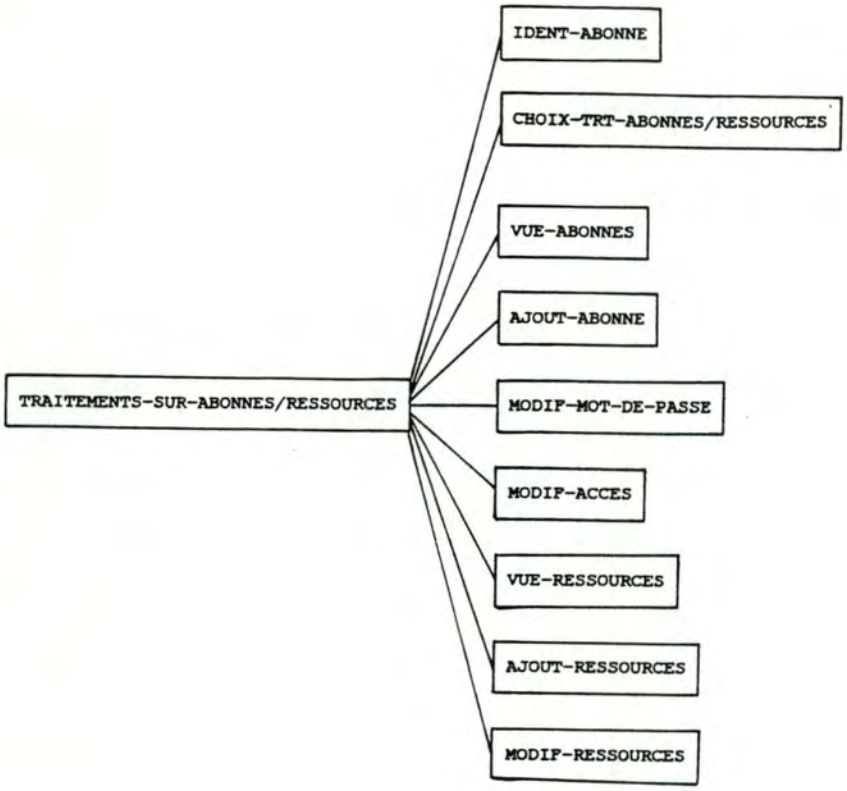
- la spécification des variables reprend pour chaque procédure le nom symbolique des variables internes utilisées dans la procédure courante ainsi que leur définition.

## 2. Architecture générale.

### 2.1. Architecture concernant les traitements sur les agendas.



## 2.2. Architecture concernant les traitements relatifs aux abonnés et ressources.



## 3. Découpe en niveau.

Cette découpe en niveau a été réalisée en appliquant une méthode d'analyse par raffinement progressif.

### 3.1. Premier niveau.

TRAITEMENTS-SUR-AGENDAS
TRAITEMENTS-SUR-ABONNES/RESSOURCES

### 3.2. Deuxième niveau.

IDENT-ABONNE
INIT-DATE
RENTREE-DATE-DU-JOUR
APUREMENT
VISUAL-ACTIVITES
CHOIX-TRT-AGENDAS
VUE-AG-ABONNE
VUE-AG-RESSOURCE
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE
RECH-PLAGE-TPS-LIBRE
CHOIX-TRT-ABONNES/RESSOURCES
VUE-ABONNES
AJOUT-ABONNE
MODIF-MOT-DE-PASSE
MODIF-ACCES
VUE-RESSOURCES
AJOUT-RESSOURCES
MODIF-RESPONSABLE

### 3.3. Troisième niveau.

ACCEDE-AGENDA
VUE-AGENDA
IDENT-ADMINISTRATEUR

## 3.4. Quatrième niveau.

REPETER-ACTIVITE
R-LST-PERSONNE
R-LST-RESSOURCE
R-T-LST-RESSOURCE
TEST-LST-RESSOURCE
SELECT-TYPE
R-INTERVALLE
SELECT-ACTIVITES
TRI-ACTIVITES
AFF-SEMAINE
AFF-MOIS
AFF-INTERVALLE

## 3.5. Cinquième niveau.

R-DATE
R-HEURE
R-NOMBRE
R-DESCRIPTION
R-PERSONNE
R-RESSOURCE
TEST-DISPONIBLE
DATE-PASSEE
AFF-JOUR

## 3.6. Sixième niveau.

VALID-DATE
TEST-DATE
TEST-HEURE
AFFICH-JOUR
PREPAR-SEMAINE
AFFICH-SEMAINE
PREPAR-MOIS
AFFICH-MOIS

## 3.7. Septième niveau.

DECOMP-DATE
TROUVE-JOUR
DECOMP-HEURE
R-REPONSE

## 3.8. Huitième niveau.

ERREUR01
ERREUR02
ERREUR03
ERREUR04
ERREUR05
ERREUR06
ERREUR07
ERREUR08
ERREUR09
ERREUR10
ERREUR11
ERREUR12
ERREUR13

## 3.9. Neuvième niveau.

OUVRIR-PERSONNE
FERMER-PERSONNE
LIRE-PERSONNE
ECRIRE-PERSONNE
REECRIRE-PERSONNE
OUVRIR-RESSOURCE
FERMER-RESSOURCE
LIRE-RESSOURCE
ECRIRE-RESSOURCE
REECRIRE-RESSOURCE
OUVRIR-ACTIVITE
FERMER-ACTIVITE
LIRE-ACTIVITE
ECRIRE-ACTIVITE
REECRIRE-ACTIVITE
SUPPRIMER-ACTIVITE
OUVRIR-APUREMENT
FERMER-APUREMENT
LIRE-APUREMENT
REECRIRE-APUREMENT

# 4. Description des procédures.

## 4.1. Premier niveau.

### 4.1.1. Procédure "TRAITEMENTS-SUR-AGENDAS"

entrées :
aucune

sorties :
aucune

effet :
cette procédure permet à un abonné d'accéder aux fonctions de consultation et/ou de modification d'un ou plusieurs agendas.

pré-conditions :
aucune

post-conditions :
aucune

procédures appelantes :
aucune

procédures appelées :
IDENT-PERSONNE
INIT-DATE
RENTREE-DATE-DU-JOUR
APUREMENT
VISUAL-ACTIVITES
CHOIX-TRT-AGENDA
VUE-AG-ABONNE
VUE-AG-RESSOURCE
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE
RECH-PLAGE-TPS-LIBRE

```
algorithme logique :
     IDENT-ABONNE (n, exact);
     si exact alors :
       INIT-DATE;
       RENTREE-DATE-DU-JOUR (date-du-jour);
       APUREMENT (date-du-jour);
       VISUAL-ACTIVITES (n);
       CHOIX-TRT-AGENDA (choix);
       tant que (choix <> '9') faire :
         selon choix :
          - '1' : VUE-AG-ABONNE (date-du-jour, n);
          - '2' : VUE-AG-RESSOURCE (date-du-jour, n);
          - '3' : AJOUT-ACTIVITE (date-du-jour, n);
          - '4' : CONS-MAJ-ACTIVITE (date-du-jour, n);
          - '5' : RECH-PLAGE-TPS-LIBRE (date-du-jour);
         CHOIX-TRT-AGENDA (choix);
```

spécification des variables internes :

- exact : booléen indiquant si l'identification de l'abonné est correcte ou non.
- n : nom de l'abonné.
- date-du-jour : date du jour sous la forme AAMMJJ.
- choix : caractère indiquant quel traitement l'abonné veut réaliser.
- début : tableau comprenant le numéro du jour de la semaine du premier jour des années d'un cycle.
- maxmois : tableau comprenant le nombre maximum de jours pour chaque mois.
- jour : tableau comprenant les jours de la semaine.
- mois : tableau comprenant les mois de l'année.
- mat-date, mat-heure, mat-duree, mat-descr : tableaux comprenant respectivement les dates, les heures, les durées et les descriptions d'un ensemble d'activités.

4.1.2. Procédure "TRAITEMENTS-SUR-ABONNES/RESSOURCES"

entrées :
aucune

sorties :
aucune

effet :
cette procédure permet à un abonné ou à l'administrateur d'accéder aux fonctions de consultation et/ou de modification de l'ensemble des abonnés et des ressources connues du système.

pré-conditions :
aucune

post-conditions :
aucune

procédures appelantes :
aucune

procédures appelées :
IDENT-ABONNE
CHOIX-TRT-ABONNES/RESSOURCES
VUE-ABONNES
AJOUT-ABONNE
MODIF-MOT-DE-PASSE
MODIF-ACCES
VUE-RESSOURCES
AJOUT-RESSOURCE
MODIF-RESPONSABLE

algorithme logique :

```
IDENT-ABONNE (n, exact);
si exact alors :
  CHOIX-TRT-ABONNES/RESSOURCES (choix);
  tant que (choix <> '9') faire :
    selon choix :
      - '1' : VUE-ABONNES;
      - '2' : AJOUT-ABONNES (n);
      - '3' : MODIF-MOT-DE-PASSE (n);
      - '4' : MODIF-ACCES (n);
      - '5' : VUE-RESSOURCES;
      - '6' : AJOUT-RESSOURCE (n);
      - '7' : MODIF-RESPONSABLE (n);
    CHOIX-TRT-ABONNES/RESSOURCES (choix);
```

spécification des variables internes :
- n : nom de l'abonné.
- exact : booléen indiquant si l'identification de l'abonné est correcte ou non.
- choix : caractère indiquant quel traitement l'abonné veut réaliser.

## 4.2. Deuxième niveau.

### 4.2.1. Procédure "IDENT-ABONNE".

entrées :
aucune

sorties :
- n : nom de l'abonné.
- exact : booléen indiquant si l'identification de l'abonné est correcte ou non.

effet :
cette procédure demande au maximum 3 fois le nom et le mot de passe de la personne désirant utiliser le gestionnaire d'agendas.
si le nom et le mot de passe correspondent à un abonné du système, alors exact = 'true'; sinon, exact = 'false'.

pré-conditions :
aucune

post-conditions :
(n est un abonné et exact) ou (pas exact)

procédures appelantes :
TRAITEMENTS-SUR-AGENDAS
TRAITEMENTS-SUR-ABONNES/RESSOURCES

procédures appelées :
OUVRIR-PERSONNE
LIRE-PERSONNE
FERMER-PERSONNE
ERREUR01

algorithme logique :

```
i := 0;
répéter :
  i := i + 1;
  rentrer n;
  rentrer m;
  exact := false ;
  OUVRIR-PERSONNE;
  LIRE-PERSONNE (p, fin);
  tant que (pas fin) et (pas exact) faire :
    si (p.nom-pers = n) et (p.mot-de-passe = m)
      alors : exact := true
      sinon : LIRE-PERSONNE (p, fin);
  FERMER-PERSONNE;
  si (pas exact) alors : ERREUR01;
jusque (exact) ou (i = 3);
```

spécification des variables internes :
- i : nombre de fois que l'utilisateur rentre un nom et un mot de passe.
- m : mot de passe rentré par l'utilisateur.
- p : enregistrement du fichier des abonnés.
- fin : booléen indiquant lorsqu'on est arrivé à la fin du fichier des abonnés.

4.2.2. Procédure "INIT-DATE".

entrées :
aucune

sorties :
aucune

données globales :
- début : tableau comprenant le numéro du jour de la semaine du premier jour des années d'un cycle.
- maxmois : tableau comprenant le nombre maximum de jour pour chaque mois.
- mois : tableau comprenant les mois de l'année.
- jour : tableau comprenant les jours de la semaine.

effet :
cette procédure initialise une série de données globales qui permettront de calculer le jour de la semaine d'une date donnée et d'afficher celle-ci.

pré-conditions :
aucune

post-conditions :
début = 5, 6, 0, 1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 0, 1, 2, 4, 5, 6, 0, 2, 3, 4, 5, 0, 1, 2, 3, et
maxmois = 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, et
mois = 'janvier', 'fevrier', 'mars', 'avril', 'mai', 'juin', 'juillet', 'aout', 'septembre', 'octobre', 'novembre', 'decembre', et
jour = 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi', 'dimanche'

procédures appelantes :
TRAITEMENT-SUR-AGENDAS

procédures appelées :
aucune

algorithme logique :
assignations;

spécifications des variables internes :
aucune

4.2.3. Procédure "RENTREE-DATE-DU-JOUR".

entrées :
aucune

sorties :
- date-du-jour : date du jour sous la forme AAMMJJ;

effet :
cette procédure demande la date du jour à l'utilisateur, vérifie si celle-ci est supérieure ou égale à la date du dernier apurement et demande confirmation.

pré-conditions :
aucune

post-conditions :
(pas de ap.date-apur) ou (date-du-jour >= ap.date-apur)

procédures appelantes :
TRAITEMENT-SUR-AGENDAS

procédures appelées :
R-DATE
OUVRIR-APUREMENT
LIRE-APUREMENT
FERMER-APUREMENT
ERREUR11
DECOMP-DATE
TROUVE-JOUR
R-REPONSE

algorithme logique :

```
répéter :
  R-DATE (date-du-jour, exact);
  si exact alors :
    OUVRIR-APUREMENT;
    LIRE-APUREMENT (ap, fin);
    FERMER-APUREMENT;
    si pas fin alors :
      si (ap.date-apur > date-du-jour) alors :
        exact := false;
        ERREUR11;
    si exact alors :
      DECOMP-DATE (date-du-jour, jo, mo, an);
      TROUVE-JOUR (jo, mo, an, jj);
      afficher la date;
      demander confirmation;
      répéter :
        R-REPONSE (réponse);
      jusque (réponse = 'o') ou (réponse = 'n');
      si (réponse = 'n') alors : exact := false;
jusque exact;
```

spécification des variables internes :
- exact : booléen indiquant si la date est correcte ou non.
- ap : enregistrement du fichier apurement.
- fin : booléen indiquant si un apurement a déjà été réalisé ou non.
- jo : jour de la date du jour.
- mo : mois de la date du jour.
- an : année de la date du jour.
- jj : jour de la semaine de la date du jour.
- réponse : caractère de confirmation.

4.2.4. Procédure "APUREMENT".

entrées :
- date-du-jour : date du jour sous la forme AAMMJJ.

sorties :
aucune

effet :
cette procédure supprime dans les agendas toutes les activités vieilles de plus de 3 mois.

pré-conditions :
(pas de ap.date-apur) ou (date-du-jour >= ap.date-apur)

post-conditions :
(date-du-jour = ap.date-apur)

procédures appelantes :
TRAITEMENTS-SUR-AGENDAS

procédures appelées :
OUVRIR-APUREMENT
LIRE-APUREMENT
REECRIRE-APUREMENT
FERMER-APUREMENT
OUVRIR-ACTIVITE
LIRE-ACTIVITE
SUPPRIMER-ACTIVITE
FERMER-ACTIVITE

```
algorithme logique :
     OUVRIR-APUREMENT;
     LIRE-APUREMENT (ap, fin);
     si fin
       alors :
         ap.date-apur := date-du-jour;
         REECRIRE-APUREMENT (ap);
       sinon :
         si (date-du-jour - ap.date-apur > 0) alors :
           ap.date-apur := date-du-jour;
           REECRIRE-APUREMENT (ap);
           date := date-du-jour - 300;
           OUVRIR-ACTIVITE;
           LIRE-ACTIVITE (a, fin);
           tant que (pas fin) faire :
             si (a.date-activ < date) alors :
               SUPPRIMER-ACTIVITE;
             LIRE-ACTIVITE (a, fin);
           FERMER-ACTIVITE;
     FERMER-APUREMENT;
```

spécification des variables internes :

- ap : enregistrement du fichier apurement.
- a : enregistrement du fichier des activités.
- fin : booléen indiquant d'abord si un apurement a déjà été réalisé ou non, et ensuite si l'on est à la fin du fichier des activités ou non.
- date : date avant laquelle toutes les activités doivent être supprimées.

### 4.2.5. Procédure "VISUAL-ACTIVITES".

entrées :

- n : nom de l'abonné.

sorties :

aucune

effet :

cette procédure affiche successivement toutes les activités concernant l'abonné et pour lesquelles il n'a pas encore donné de réponse, et lui demande une réponse.

pré-conditions :

(n est un abonné)

post-conditions :

aucune

procédures appelantes :

TRAITEMENTS-SUR-AGENDAS

```
procédures appelées :
     OUVRIR-ACTIVITE
     LIRE-ACTIVITE
     REECRIRE-ACTIVITE
     FERMER-ACTIVITE
     R-REPONSE
     DECOMP-DATE
     TROUVE-JOUR
     DECOMP-HEURE

algorithme logique :
     OUVRIR-ACTIVITE;
     LIRE-ACTIVITE (a, fin);
     tant que (pas fin) faire :
       i := 0;
       concerne := false;
       tant que (pas concerne) et (i < 10) faire :
         i := i + 1;
         si (a.nom-pers-cons [i] = n) alors :
           concerne := true;
       si (concerne) et (a.code-reponse [i] = ' ') alors :
         afficher a.nom-enreg;
         DECOMP-DATE (a.date-activ, jo, mo, an);
         TROUVE-JOUR (jo, mo, an, jj);
         afficher la date de l'activité;
         DECOMP-HEURE (a.heure, he, min);
         afficher l'heure;
         DECOMP-HEURE (a.durée, he, min);
         afficher la durée;
         afficher a.type;
         afficher a.descr;
         afficher a.nom-resp;
         répéter :
           R-REPONSE (réponse);
         jusque (réponse = ' ') or (réponse = 'o') or
                (réponse = 'n');
         si (réponse <> ' ') alors :
           a.code-réponse [i] := réponse;
           REECRIRE-ACTIVITE (a);
       LIRE-ACTIVITE (a, fin);
     FERMER-ACTIVITE;
```

spécification des variables internes :
- a : enregistrement du fichier des activités.
- fin : booléen indiquant si l'on est à la fin du fichier des activités ou non.
- i : indice portant sur la liste des personnes concernées.
- concerne : booléen indiquant si l'abonné est concerné ou non par l'activité courante.
- jo : jour de la date de l'activité courante.
- mo : mois de la date de l'activité courante.
- an : année de la date de l'activité courante.
- jj : jour de la semaine de la date de l'activité courante.
- he : partie entière de l'heure de début et de la durée de l'activité.
- min : minutes de l'heure de début et de la durée de l'activité.

### 4.2.6. Procédure "CHOIX-TRT-AGENDAS"

entrées :
aucune

sorties :
- choix : caractère indiquant quel traitement l'abonné veut réaliser.

effet :
cette procédure propose à l'utilisateur une série de traitements possibles et lui demande un choix :
1. vue de l'agenda d'un abonné,
2. vue de l'agenda d'une ressource,
3. ajout d'une activité ou d'une série d'activités,
4. consultation et/ou mise à jour d'une activité,
5. recherche de plages de temps libre.
9. fin de travail

pré-conditions :
aucune

post-conditions :
(choix > '0' et choix < '6') ou (choix = '9')

procédures appelantes :
TRAITEMENTS-SUR-AGENDAS

procédures appelées :
R-REPONSE

algorithme logique :
```
répéter :
  afficher des traitements possibles;
  demander un choix;
  R-REPONSE (choix);
jusque (choix > '0' et choix < '6') ou (choix = '9');
```

spécification des variables internes:
aucune

4.2.7. Procédure "VUE-AG-ABONNE".

entrées :
- date-du-jour : date du jour sous la forme AAMMJJ.
- n : nom de l'abonné désirant consulter l'agenda d'un abonné.

sorties :
aucune

effet :
cette procédure permet à un abonné de consulter l'agenda d'un abonné s'il a accès à celui-ci.

pré-conditions :
(date-du-jour) et (n est un abonné)

post-conditions :
rien

procedures appelantes :
TRAITEMENTS-SUR-AGENDAS

procédures appelées :
ACCEDE-AGENDA
VUE-AGENDA

algorithme logique :
ACCEDE-AGENDA (n, nom, exact);
si exact alors :
  VUE-AGENDA (date-du-jour, n, 0, nom);

spécifications des variables internes :
- nom : nom de la personne à qui appartient l'agenda que veut consulter l'abonné 'n'.
- exact : booléen indiquant si l'abonné 'n' a accès à l'agenda de l'abonné 'nom'.

4.2.8. Procédure "VUE-AG-RESSOURCE"

entrées :
- date-du-jour : date du jour sous la forme AAMMJJ.
- n : nom de l'abonné désirant consulter l'agenda d'une ressource.

sorties :
aucune

effet :
cette procédure permet à un abonné de consulter l'agenda d'une ressource connue du système.

pré-conditions :
(date-du-jour) et (n est un abonné)

post-conditions :
aucune

procédures appelantes :
TRAITEMENTS-SUR-AGENDAS

procédures appelées :
R-RESSOURCE
VUE-AGENDA
ERREUR04

algorithme logique :
```
i := 0;
répéter :
  i := i + 1;
  R-RESSOURCE (nom, exact);
  si (pas exact) alors : ERREUR04;
jusque (exact) ou (i = 3);
si exact alors :
  VUE-AGENDA (date-du-jour, n, 1, nom);
```

spécification des variables internes :
- i : nombre de fois que l'abonné rentre un nom de ressource.
- nom : nom de ressource.
- exact : booléen indiquant si la ressource est connue du système ou non.

### 4.2.9. Procédure "AJOUT-ACTIVITE".

entrées :
- date-du-jour : date du jour sous la forme AAMMJJ.
- n : nom de l'abonné désirant ajouter une activité ou une série d'activités.

sorties :
aucune

effet :
cette procédure permet à un abonné d'ajouter une activité ou une série d'activités dans un ou plusieurs agendas.

pré-conditions :
(date-du-jour) et (n est un abonné)

post-conditions :
aucune

procédures appelantes :
TRAITEMENTS-SUR-AGENDAS

```
procédures appelées :
     R-DATE
     ERREUR09
     R-HEURE
     R-DESCRIPTION
     R-REPONSE
     R-PERSONNE
     R-LST-PERSONNE
     R-T-LST-RESSOURCE
     OUVRIR-ACTIVITE
     ECRIRE-ACTIVITE
     REPETER-ACTIVITE
     VALID-DATE
     TEST-LST-RESSOURCE
     DECOMP-DATE
     TROUVE-JOUR
     FERMER-ACTIVITE

algorithme logique :
     i := 0;
     répeter :
       i := i + 1;
       R-DATE (a.date-activ, exact);
       si exact alors :
         si (a.date-activ < date-du-jour) alors :
           exact := false;
           ERREUR09;
     jusque (exact) ou (i = 3);
     si exact alors :
       répéter :
         R-HEURE (a.heure, exact);
       jusque exact;
       répéter :
         R-HEURE (a.durée, exact);
       jusque exact;
       rentrer a.type;
       R-DESCRIPTION (a.descr);
       répéter :
         R-REPONSE (a.confid);
       jusque (a.confid = 'o') ou (a.confid = 'n');
       R-LST-PERSONNE (a.nom-pers-cons);
       pour i := 1 à 10 faire : a.code-réponse [i] = ' ';
       R-T-LST-RESSOURCE (a.date-activ, a.heure, a.durée,
                          a.nom-ress-cons);
       OUVRIR-ACTIVITE;
       ECRIRE-ACTIVITE (a);
```

```
REPETER-ACTIVITE (c, nbre, inter);
si (c = 'o') alors :
  pour i := 1 à nbre faire :
    a.date-activ := a.date-activ + inter;
    VALID-DATE (a.date-activ);
    TEST-LST-RESSOURCE (a.date-activ, a.heure,
                        a.durée, a.nom-ress-cons,
                        exact);
    si exact
      alors :
        ECRIRE-ACTIVITE (a);
      sinon :
        DECOMP-DATE (a.date-activ, jo, mo, an);
        TROUVE-JOUR (jo, mo, an, jj);
        afficher la date de l'activité ne pouvant
        avoir lieu;
FERMER-ACTIVITE;
```

spécification des variables internes :

- i : indique d'abord le nombre de fois que l'abonné rentre la date de l'activité; ensuite sert d'indice à la liste des codes réponses; et pour terminer, compte les activités lors de l'ajout d'une série d'activités.
- a : enregistrement du fichier des activités.
- exact : booléen indiquant successivement si la date de l'activité est correcte, si l'heure est correcte, si la durée est correcte, si le nom du responsable est un abonné ou non, et si les ressources consommées par l'activité sont disponible durant toute la durée de celle-ci.
- c : caractère indiquant s'il s'agit d'une série d'activités ou non.
- nbre : nombre d'activités de la série - 1.
- inter : nombre de jours séparant deux activités de la série.
- jo : jour de la date de l'activité.
- mo : mois de la date de l'activité.
- an : année de la date de l'activité.
- jj : jour de la semaine de la date de l'activité.

4.2.10. Procédure "CONS-MAJ-ACTIVITE".

entrées :
- date-du-jour : date du jour sous la forme AAMMJJ.
- n : nom de l'abonné désirant consulter et/ou modifier une activité.

sorties :
aucune

effet :
cette procédure permet à un abonné de consulter les renseignements concernant une activité qu'il a lui-même enregistrée ou de laquelle il est responsable.
au terme de cette consultation, l'abonné peut modifier les renseignements de l'activité, réecrire celle-ci, l'ajouter ou la supprimer.

pré-conditions :
(date-du-jour) et (n est un abonné)

post-conditions :
aucune

procédures appelantes :
TRAITEMENTS-SUR-AGENDAS

procédures appelées :
R-DATE
R-HEURE
OUVRIR-ACTIVITE
LIRE-ACTIVITE
DECOMP-DATE
TROUVE-JOUR
DECOMP-HEURE
SUPPRIMER-ACTIVITE
R-REPONSE
ERREUR09
R-DESCRIPTION
R-PERSONNE
ERREUR03
R-LST-PERSONNE
R-T-LST-RESSOURCE
ECRIRE-ACTIVITE
TEST-LST-RESSOURCE
FERMER-ACTIVITE

```
algorithme logique ;
     répéter :
       R-DATE (date, exact);
     jusque exact;
     répéter :
       R-HEURE (heure, exact);
     jusque exact;
     OUVRIR-ACTIVITE;
     LIRE-ACTIVITE (a, fin);
     exact := false;
     tant que (pas fin) et (pas exact) faire :
       si (date = a.date-activ) et (heure = a.heure)
         alors :
           si (n = a.nom-resp) ou (n = a.nom-enreg)
             alors :
               AFFICHAGE-ACTIVITE (a);
               demander confirmation;
               répéter :
                 R-REPONSE (réponse);
               jusque (réponse = 'o') or (réponse = 'n');
               si (réponse = 'o')
                 alors : exact := true;
                 sinon : LIRE-ACTIVITE (a, fin);
             sinon :
               LIRE-ACTIVITE (a, fin);
         sinon :
           LIRE-ACTIVITE (a, fin);
     si exact
       alors :
         SUPPRIMER-ACTIVITE;
         a.sauv := a;
         sw1 := false;
         sw2 := false;
         répéter :
           AFFICHAGE-ACTIVITE (a);
           affichage des opérations possibles;
           R-REPONSE (réponse);
           si (réponse >= '1') et (réponse <= '9') alors :
             sw1 := true;
             selon réponse :
              - '1' : répéter :
                        R-DATE (a.date-activ, exact);
                        si (a.date-activ < date-du-jour)
                          alors :
                            erreur09;
                            exact := false;
                      jusque exact;
                      sw2 := true;
              - '2' : répéter :
                        R-HEURE (a.heure, exact);
                      jusque exact;
                      sw2 := true;
              - '3' : répéter :
                        R-HEURE (a.début, exact);
                      jusque exact;
                      sw2 := true;
              - '4' : rentrer a.sorte;
```

```
                    - '5' : rentrer a.description;
                    - '6' : répéter :
                              R-REPONSE (a.confid);
                            jusque (a.confid = 'o') ou
                                   (a.confid = 'n');
                    - '7' : répéter :
                              R-PERSONNE (a.nom-resp, exact);
                              if (not exact) and
                                 (a.nom-resp <> '          ')
                                alors : ERREUR03;
                            jusque (exact) ou
                                   (a.nom-resp = '          ');
                    - '8' : R-LST-PERSONNE (a,nom-pers-cons);
                    - '9' : R-T-LST-RESSOURCE (a.date-activ,
                                               a.heure, a.durée,
                                               a.nom-ress-cons);
                          sw2 := false;
                jusque (réponse = 'r') ou (réponse = 'a') ou
                       (réponse = 's');
                si (réponse <> 's') alors :
                  si sw1 alors :
                    a.nom-enreg := n;
                    pour i := 1 à 10 faire :
                      a.code-réponse [i] := ' ';
                  si (réponse = 'a') alors :
                    ECRIRE-ACTIVITE (a.sauv);
                  si (réponse = 'a') ou (sw2) alors :
                    TEST-LST-RESSOURCE (a.date-activ, a.heure,
                                        a.durée, a.nom-resscons,
                                        exact);
                  ECRIRE-ACTIVITE (a);
                FERMER-ACTIVITE;
              sinon :
                signaler qu'il n'y a pas ou plus d'activités
                correspondant aux date et heure rentrées;
```

spécification des variables internes :

- date : date de l'activité recherchée sous la forme AAMMJJ.
- heure : heure de l'activité recherchée.
- exact : booléen indiquant si les données rentrées sont exactes ou non.
- a : enregistrement du fichier des activités.
- fin : booléen indiquant si l'on est à la fin du fichier des activités ou non.
- a-sauv : sauvetage de l'enregistrement avant modification.
- i : indice portant sur la liste des codes réponses.
- réponse : caractère indiquant d'abord si l'enregistrement sélectionné est celui désiré ou non, et ensuite l'opération sélectionnée par l'abonné.

4.2.11. Procédure "RECH-PLAGE-TPS-LIBRE".

entrées :
- date-du-jour : date du jour sous la forme AAMMJJ.

sorties :
aucune

effet :
cette procédure permet à un abonné de visualiser les plages de temps libre d'un ensemble d'abonnés et de ressources suivant une date donnée.

pré-conditions :
date-du-jour

post-conditions :
aucune

procédures appelantes :
TRAITEMENTS-SUR-AGENDAS

procédures appelées :
R-DATE
ERREUR09
VALID-DATE
R-LST-PERSONNE
R-LST-RESSOURCE
OUVRIR-ACTIVITE
LIRE-ACTIVITE
FERMER-ACTIVITE
DECOMP-DATE
TROUVE-JOUR
DECOMP-HEURE

algorithme logique :

```
i := 0;
répéter :
  R-DATE (date-début, exact);
  si exact alors :
    si (date-début < date-du-jour) alors :
      exact := false;
      erreur09;
jusque (exact) ou (i = 3);
si exact alors :
  date-fin := date-début + 6;
  VALID-DATE (date-fin);
  R-LST-PERSONNE (lst-personne);
  R-LST-RESSOURCE (lst-ressource);
  initialisation de la table;
```

```
OUVRIR-ACTIVITE;
LIRE-ACTIVITE (a, fin);
tant que (pas fin) faire :
  si (date-début <= a.date-activ) et
     (date-fin >= a.date-activ) alors :
    exact := false;
    i := 0;
    tant que (i < 10) et (pas exact) faire :
      i := i + 1;
      si (a.nom-enreg = lst-personne [i]) alors :
        exact := true;
      si (a.nom-resp = lst-personne [i]) alors :
        exact := true;
      j := 0;
      tant que (j < 10) et (pas exact) faire :
        j := j + 1;
        si (a.nom-pers-cons [j] = lst-personne [i])
           et (a.code-réponse [j] = 'o') alors :
          exact := true;
    i := 0
    tant que (i < 4) et (pas exact) faire :
      i := i + 1;
      j := 0;
      tant que (j < 4) et (pas exact) faire :
        j := j + 1;
        si (a.nom_ress-cons[j] = lst-ressource [i])
          alors :
            exact := true;
    si exact alors :
      DECOMP-DATE (a.date-activ, jo, mo, an);
      TROUVE-JOUR (jo, mo, an, jj);
      DECOMP-HEURE (a.heure, he, min);
      DECOMP-HEURE (a.durée, he, min);
      mise à jour de la table;
  LIRE-ACTIVITE (a, fin);
FERMER-ACTIVITE;
afficher lst-personne;
afficher lst-ressource;
DECOMP-DATE (date-début, jo, mo, an);
TROUVE-JOUR (jo, mo, an, jj);
afficher date de début;
DECOMP-DATE (date-fin, jo, mo, an);
afficher date de fin;
afficher la table;
```

spécification des variables internes :

- i : indique d'abord le nombre de fois que l'abonné rentre la date de début de l'intervalle et sert ensuite d'indice à la liste des abonnés et à la liste des ressources.
- date-début : date de début de l'intervalle.
- exact : booléen indiquant d'abord si la date de début est correcte ou non et ensuite si l'activité est à prendre en compte ou non.
- date-fin : date de fin de l'intervalle;
- lst-personne : liste des personnes pour lesquelles on recherche les plages de temps libre.
- lst-ressource : liste des ressources pour lesquelles on recherche les plages de temps libre.
- table : table permettant de visualiser les plages de temps libre.
- a : enregistrement du fichier des activités.
- fin : booléen indiquant si l'on est à la fin du fichier des activités ou non.
- j : indice la liste des personnes concernées et la liste des ressources consommées par l'activité courante.
- jo : jour de la date.
- mo : mois de la date.
- an : annnée de la date.
- jj : jour de la semaine de la date.

### 4.2.12. Procédure "CHOIX-TRT-ABONNES/RESSOURCES".

entrées :

aucune

sorties :

- choix : caractère indiquant quel traitement l'abonné ou l'administrateur veut réaliser.

effet :

cette procédure propose à l'abonné ou à l'administrateur une série de traitements possibles et lui demande un choix :

1. liste des abonnés.
2. ajout d'un abonné.
3. modification du mot de passe d'un abonné.
4. modification de la liste des accès à l'agenda d'un abonné.
5. liste des ressources.
6. ajout d'une ressource.
7. modification du responsable d'une ressource.
9. fin de traitement.

pré-conditions :

aucune

post-conditions :

(choix > '0' et choix < '8') ou (choix = '9')

procédures appelantes :
TRAITEMENTS-SUR-ABONNES/RESSOURCES

procédures appelées :
R-REPONSE

algorithme logique :
répéter :
affichage des traitements possibles;
R-REPONSE (choix);
jusque (choix > '0' et choix < '8') or (choix = '9');

spécification des variables internes :
aucune

4.2.13. Procédure "VUE-ABONNES".

entrées :
aucune

sorties :
aucune

effet :
cette procédure permet à un abonné de visualiser la liste des abonnés ainsi que l'ensemble des personnes ayant accès à leur agenda respectif.

pré-conditions :
aucune

post-conditions :
aucune

procédures appelantes :
TRAITEMENTS-SUR-ABONNES/RESSOURCES

procédures appelées :
OUVRIR-PERSONNE
LIRE-PERSONNE
FERMER-PERSONNE

algorithme logique :
OUVRIR-PERSONNE;
LIRE-PERSONNE (p, fin);
tant que (pas fin) faire :
afficher p.nom-pers;
afficher p.nom-accès;
LIRE-PERSONNE (p, fin);
FERMER-PERSONNE;

spécification des variables internes :
- p : enregistrement du fichier des abonnés.
- fin : booléen indiquant si l'on est à la fin du fichier des abonnés ou non.

4.2.14. Procédure "AJOUT-ABONNE".

entrées :
- n : nom de l'abonné désirant ajouter un abonné dans le système.

sorties :
aucune

effet :
cette procédure permet à l'administrateur du système d'ajouter un abonné à l'ensemble des abonnés au système.

pré-conditions :
n est un abonné

post-conditions :
aucune

procédures appelantes :
TRAITEMENT-SUR-ABONNES/RESSOURCES

procédures appelées :
IDENT-ADMINISTRATEUR
R-PERSONNE
ERREUR05
R-LST-PERSONNE
OUVRIR-PERSONNE
ECRIRE-PERSONNE
FERMER-PERSONNE

algorithme logique :

```
IDENT-ADMINISTRATEUR (n, exact);
si exact alors :
  répéter :
    R-PERSONNE (p.nom-pers, exact);
    si exact
      alors :
        i := i + 1;
        ERREUR05
      sinon :
        rentrer p.mot-de-passe;
        R-LST-PERSONNE (p.nom-acces);
        OUVRIR-PERSONNE;
        ECRIRE-PERSONNE;
        FERMER-PERSONNE;
  jusque (not exact) ou (i = 3);
```

spécification des variables internes :
- i : nombre de fois que l'administrateur rentre un nom d'abonné déjà existant.
- exact : booléen indiquant d'abord si l'abonné est l'administrateur, et ensuite si le nom rentré existe déjà ou non.
- p : enregistrement du fichier des abonnés.

4.2.15. Procédure "MODIF-MOT-DE-PASSE".

entrées :
- n : nom de l'abonné désirant modifier son mot de passe.

sorties :
aucune

effet :
cette procédure permet à un abonné de modifier son mot de passe.

pré-conditions :
n est un abonné.

post-conditions :
aucune

procédures appelantes :
TRAITEMENTS-SUR-ABONNES/RESSOURCES

procédures appelées :
OUVRIR-PERSONNE
LIRE-PERSONNE
REECRIRE-PERSONNE
FERMER-PERSONNE

algorithme logique :
```
OUVRIR-PERSONNE;
répéter :
  LIRE-PERSONNE (p, fin);
jusque p.nom-pers = n;
rentrer p.mot-de-passe;
REECRIRE-PERSONNE (p);
FERMER-PERSONNE;
```

spécification des variables internes :
- p : enregistrement du fichier des abonnés.
- fin : booléen indiquant si l'on est à la fin du fichier des abonnés ou non.

4.2.16. Procédure "MODIF-ACCES".

entrées :
- n : nom de l'abonné désirant modifier la liste des personnes ayant accès à son agenda.

sorties :
aucune

effet :
cette procédure permet à un abonné de modifier la liste des personnes ayant accès à son agenda.

pré-conditions :
n est un abonné

post-conditions :
aucune

procédures appelantes :
TRAITEMENTS-SUR-ABONNES/RESSOURCES

procédures appelées :
OUVRIR-PERSONNE
LIRE-PERSONNE
R-LST-PERSONNE
REECRIRE-PERSONNE
FERMER-PERSONNE

algorithme logique :
```
OUVRIR-PERSONNE;
répéter :
  LIRE-PERSONNE (p, fin);
jusque (p.nom-pers = n);
afficher p.nom-accès;
R-LST-PERSONNE (p.nom-accès);
REECRIRE-PERSONNE (p);
FERMER-PERSONNE;
```

spécification des variables internes :
- p : enregistrement du fichier des abonnés.
- fin : booléen indiquant si l'on est à la fin du fichier des abonnés ou non.

4.2.17. Procédure "VUE-RESSOURCES".

entrées :
aucune

sorties :
aucune

effet :
cette procédure permet à un abonné de visualiser la liste des ressources ainsi que le nom de leur responsable respectif.

pré-conditions :
aucune

post-conditions :
aucune

procédures appelantes :
TRAITEMENTS-SUR-ABONNES/RESSOURCES

procédures appelées :
OUVRIR-RESSOURCE
LIRE-RESSOURCE
FERMER-RESSOURCE

algorithme logique :

```
OUVRIR-RESSOURCE;
LIRE-RESSOURCE (r, fin);
tant que (pas fin) faire :
  afficher r.nom-ress;
  afficher r.nom-resp;
  LIRE-RESSOURCE (r, fin);
FERMER-RESSOURCE;
```

spécification des variables internes :

- r : enregistrement du fichier des ressources.
- fin : booléen indiquant si l'on est à la fin du fichier des ressources ou non.

4.2.18. Procédure "AJOUT-RESSOURCE".

entrées :

- n : nom de l'abonné désirant ajouter une ressource dans le système.

sorties :

aucune

effet :

cette procédure permet à l'administrateur du système d'ajouter une ressource à l'ensemble des ressources connues du système.

pré-conditions :

n est un abonné

post-conditions :

aucune

procédures appelantes :

TRAITEMENTS-SUR-ABONNES/RESSOURCES

procédures appelées :

IDENT-ADMINISTRATEUR
R-RESSOURCE
ERREUR06
R-PERSONNE
ERREUR03
OUVRIR-RESSOURCE
ECRIRE-RESSOURCE
FERMER-RESSOURCE

```
algorithme logique :
     IDENT-ADMINISTRATEUR (n, exact);
     if exact alors :
       i := 0;
       répéter :
         R-RESSOURCE (r.nom-ress, exact);
         si exact
           alors :
             i := i + 1;
             ERREUR06;
           sinon :
             j := 0;
             répéter :
               R-PERSONNE (r.nom-resp, exact);
               si (pas exact)
                 alors :
                   j := j + 1;
                   ERREUR03;
                 sinon :
                   OUVRIR-RESSOURCE;
                   ECRIRE-RESSOURCE (r);
                   FERMER-RESSOURCE;
             jusque (exact) ou (j = 3);
       jusque (exact) ou (i = 3) ou (j = 3);
```

spécification des variables internes :

- exact : booléen indiquant d'abord si l'abonné est l'administrateur et ensuite si le nom de ressource et le nom d'abonné existe déjà ou non.
- i : nombre de fois que l'administrateur rentre un nom de ressource déjà existant.
- j : nombre de fois que l'administrateur rentre un nom d'abonné incorrect.
- r : enregistrement du fichier des ressources.

### 4.2.19. Procédure "MODIF-RESPONSABLE".

entrées :

- n : nom de l'abonné désirant modifier le responsable d'une ressource.

sorties :

aucune

effet :

cette procédure permet à l'administrateur du système de modifier le responsable d'une ressource.

pré-conditions :

n est un abonné

post-conditions :

aucune

procédures appelantes :

TRAITEMENTS-SUR-ABONNES/RESSOURCES

procédures appelées :
IDENT-ADMINISTRATEUR
ERREUR02
R-PERSONNE
ERREUR03
OUVRIR-RESSOURCE
LIRE-RESSOURCE
REECRIRE-RESSOURCE
FERMER-RESSOURCE

```
algorithme logique :
     IDENT-ADMINISTRATEUR (n, exact);
     si exact alors :
       i := 0;
       répéter :
         i := i + 1;
         rentrer nom;
         exact := false;
         OUVRIR-RESSOURCE;
         LIRE-RESSOURCE (r, fin);
         tant que (pas fin) et (pas exact) faire :
           si (r.nom-ress = nom)
             alors : exact := true;
             sinon : LIRE-RESSOURCE (r, fin);
         si (pas exact) alors :
           ERREUR02;
           FERMER-RESSOURCE;
       jusque (exact) ou (i = 3);
       si exact alors :
         i := 0;
         répéter :
           R-PERSONNE (r.nom-resp, exact);
           si (pas exact)
             alors :
               i := i + 1;
               ERREUR03;
             sinon :
               REECRIRE-RESSOURCE (r);
         jusque (exact) ou (i = 3);
         FERMER-RESSOURCE;
```

spécification des variables internes :

- exact : booléen indiquant d'abord si l'abonné est l'administrateur ou non, ensuite si la ressource rentrée est connue du système, et pour terminer si le nom du nouveau responsable rentré est un abonné ou non.
- i : indique d'abord le nombre de fois que l'administrateur rentre un nom de ressource et ensuite le nombre de fois qu'il rentre un nom d'abonné.
- nom : nom de la ressource pour laquelle l'administrateur veut changer le responsable.
- r : enregistrement du fichier des ressources.
- fin : booléen indiquant si l'on est à la fin du fichier des ressources ou non.

## 4.3. Troisième niveau.

### 4.3.1. Procédure "ACCEDE-AGENDA".

entrées :
- n : nom de l'abonné désirant consulter l'agenda d'un abonné.

sorties :
- nom : nom de la personne à qui appartient l'agenda que veut consulter l'abonné.
- exact : booléen indiquant si la personne rentrée a un agenda dans le système et si l'abonné a accès à celui-ci.

effet :
cette procédure demande à l'abonné de rentrer le nom de la personne dont il veut consulter l'agenda, vérifie si celle-ci est un abonné et si son agenda est accessible à l'abonné.

pré-conditions :
n est un abonné

post-conditions :
((nom est un abonné) et
(n a accès à l'agenda de nom) et
(exact)) ou
(pas exact)

procédures appelantes :
VUE-AG-ABONNE

procédures appelées :
OUVRIR-PERSONNE
LIRE-PERSONNE
FERMER-PERSONNE
ERREUR12
ERREUR03

```
algorithme logique :
     i := 0;
     exact := false;
     répéter :
       i := i + 1;
       rentrer nom;
       OUVRIR-PERSONNE;
       LIRE-PERSONNE (p, fin);
       tant que (pas fin) et (pas exact) faire :
         si (p.nom-pers = nom)
           alors : exact := true;
           sinon : LIRE-PERSONNE (p, fin);
       FERMER-PERSONNE;
       si exact
         alors :
           j := 1;
           si (nom <> n) alors :
             exact := false;
           tant que (pas exact) et (j < 11) faire :
             si (n = p.nom-accès [j])
               alors : exact := true;
               sinon : j := j + 1;
           si (pas exact) alors : ERREUR12;
         sinon :
           ERREUR03;
     jusque (exact) ou (i = 3);
```

spécification des variables internes :

- i : nombre de fois que l'abonné rentre un nom d'abonné.
- p : enregistrement du fichier des abonnés.
- fin : booléen indiquant si l'on est à la fin du fichier des abonnés ou non.
- j : indice portant sur la liste des personnes ayant accès à l'agenda d'un abonné.

### 4.3.2. Procédure "VUE-AGENDA".

entrées :

- date-du-jour : date du jour sous la forme AAMMJJ.
- n : nom de l'abonné désirant consulter un agenda.
- no : indique si l'abonné veut consulter l'agenda d'un abonné (0) ou d'une ressource (1).
- nom : nom de l'abonné ou de la ressource dont l'abonné veut consulter l'agenda.

sorties :

aucune

effet :

cette procédure permet à l'abonné de visualiser des parties (un jour, une semaine, un mois ou un intervalle quelconque) de l'agenda d'un abonné ou d'une ressource en sélectionnant s'il le désire un type d'activités.

```
pré-conditions :
     (date-du-jour) et (n est un abonné) et
     (((no = 0) et
       (nom est un abonné) et
       (n a accès à l'agenda de nom)) ou
     ((no = 1) et (nom est une ressource)))

post-conditions :
     aucune

procédures appelantes :
     VUE-AG-ABONNE
     VUE-AG-RESSOURCE

procédures appelées :
     SELECT-TYPE
     R-REPONSE
     R-INTERVALLE
     SELECT-ACTIVITES
     TRI-ACTIVITES
     AFF-JOUR
     AFF-SEMAINE
     AFF-MOIS
     AFF-INTERVALLE

algorithme logique :
     SELECT-TYPE (type);
     répéter :
       affichage des options possibles;
       R-REPONSE (numéro);
       si (numéro >= '1') et (numéro <= '4') alors :
         R-INTERVALLE (date-du-jour, numéro,
                       date-début, date-fin);
         SELECT-ACTIVITES (n, no, nom, type,
                           date-début, date-fin, i);
         TRI-ACTIVITES (i);
         selon numéro :
          - '1' : AFF-JOUR (nom, type, date-début, i);
          - '2' : AFF-SEMAINE (nom, type,
                               date-début, date-fin, i);
          - '3' : AFF-MOIS (nom, type,
                            date-début, date-fin, i);
          - '4' : AFF-INTER (nom, type,
                             date-début, date-fin, i);
     jusque (numéro = '9');

spécification des variables internes :
     - type : type des activités sélectionnées
              (= '       ' si pas de sélection).
     - numéro : caractère indiquant l'option choisie par
                l'abonné :
                = '1' si vue d'un jour,
                = '2' si vue d'une semaine,
                = '3' si vue d'un mois,
                = '4' si vue d'un intervalle quelconque,
                = '9' si fin de visualisation.
```

- date-début : date de début de l'intervalle à visualiser.
- date-fin : date de fin de l'intervalle à visualiser.
- i : nombre d'activités sélectionnées.

4.3.3. Procédure "IDENT-ADMINISTRATEUR".

entrées :
- n : nom de l'abonné.

sorties :
- exact : booléen indiquant si l'abonné est l'administrateur ou non.

effet :
cette procédure vérifie si l'abonné est l'administrateur du système ou non.

pré-conditions :
n est un abonné

post-conditions :
((n est l'administrateur) et (exact)) ou
(pas exact)

procédures appelantes :
AJOUT-ABONNE
AJOUT-RESSOURCE
MODIF-RESPONSABLE

procédures appelées :
OUVRIR-PERSONNE
LIRE-PERSONNE
FERMER-PERSONNE

algorithme logique :
```
OUVRIR-PERSONNE;
LIRE-PERSONNE (p, fin);
si (p.nom-pers = n)
  alors : exact := true;
  sinon : exact := false;
FERMER-PERSONNE;
```

spécification des variables internes :
- p : enregistrement du fichier des abonnés.
- fin : booléen indiquant si l'on est à la fin du fichier des abonnés ou non.

## 4.4. Quatrième niveau.

### 4.4.1. Procédure "REPETER-ACTIVITE".

entrées :
aucune

sorties :
- réponse : caractère indiquant si l'abonné reporte l'activité sur d'autres journées ('o') ou non ('n').
- nbre : nombre de fois que l'abonné reporte l'activité.
- inter : nombre de jours séparant deux de ces activités.

effet :
cette procédure demande à l'abonné s'il veut reporter l'activité sur d'autres journées et si oui, lui demande le nombre de fois et le nombre de jours séparant deux de celles-ci.

pré-conditions :
aucune

post-conditions :
(réponse = 'n') ou
((réponse = 'o') et (nbre <> 0) et
(inter > 0) et (inter < 29))

procédures appelantes :
AJOUT-ACTIVITE

procédures appelées :
R-REPONSE
R-NOMBRE

algorithme logique :
```
répéter :
  R-REPONSE (réponse);
jusque (réponse = 'o') ou (réponse = 'n');
si (réponse = 'o') alors :
  répéter :
    R-NOMBRE (nbre);
  jusque (nbre <> 0);
  répéter :
    R-NOMBRE (inter);
  jusque (inter > 0) et (inter < 29);
```

spécification des variables internes :
aucune

4.4.2. Procédure "R-LST-PERSONNE".

entrées :
aucune

sorties :
- a : liste d'abonnés.

effet :
cette procédure demande une liste de personnes et teste si celles-ci sont bien des abonnés.

pré-conditions :
aucune

post-conditions :
a sont des abonnés

procédures appelantes :
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE
RECH-PLAGE-TPS-LIBRE
MODIF-ACCES

procédures appelées :
R-PERSONNE
ERREUR03

algorithme logique :

```
i := 0;
répéter :
  R-PERSONNE (n, exact);
  si exact
    alors :
      i := i + 1;
      a [i] := n;
    sinon :
      si (n <> '          ') alors :
        ERREUR03;
jusque (n = '          ') ou (i = 10);
tant que (i < 10) faire :
  i := i + 1;
  a [i] := '          ';
```

spécification des variables :
- i : indice portant sur la liste des personnes.
- n : nom d'une personne.
- exact : booléen indiquant si le nom rentré est celui d'un abonné ou non.

4.4.3. Procédure "R-LST-RESSOURCE".

entrées ;
aucune

sorties :
- l : liste de ressources.

effet :
cette procédure demande une liste de ressources et teste si celles-ci sont connues du système.

pré-conditions :
aucune

post-conditions :
l sont des ressources connues du sytème

procédures appelantes :
RECH-PLAGE-TPS-LIBRE
AJOUT-ABONNE
MODIF-ACCES

procédures appelées :
R-RESSOURCE
ERREUR04

algorithme logique :

```
i := 0;
répéter :
  R-RESSOURCE (n, exact);
  si exact
    alors :
      i := i + 1;
      l [i] := n;
    sinon :
      si (n <>  '          ') alors :
        ERREUR04;
jusque (n =  '          ') ou (i = 4);
tant que (i < 4) faire :
  i := i + 1;
  l [i] :=  '          ';
```

spécification des variables internes :
- i : indice portant sur la liste des ressources.
- n : nom de ressource.
- exact : booléen indiquant si le nom rentré est celui d'une ressource connue du système ou non.

4.4.4. Procédure "R-T-LST-RESSOURCE".

entrées :
- date : date de l'activité.
- heure : heure de l'activité.
- durée : durée estimée de l'activité.

sorties :
- l : liste de ressources connues du système.

effet :
cette procédure demande la liste des ressources consommées par une activité, vérifie si celles-ci sont connues du système et disponibles pendant toute la durée de l'activité.

pré-conditions :
(date) et (heure) et (durée)

post-conditions :
l sont des ressources connues du sytème et disponibles pendant toute la durée de l'activité.

procédures appelantes :
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE

procédures appelées :
R-RESSOURCE
TEST-DISPONIBLE
ERREUR07
ERREUR04

algorithme logique :

```
i := 0;
répéter :
  R-RESSOURCE (n, exact);
  si exact
    alors :
      TEST-DISPONIBLE (date, heure, durée, n, occupé);
      si occupé
        alors :
          ERREUR07;
        sinon :
          i := i + 1;
          l [i] := n;
    sinon :
      si (n <> '          ') alors :
        ERREUR04;
jusque (n = '          ') ou (i = 4);
tant que (i < 4) faire :
  i := i + 1;
  l [i] := '          ';
```

spécification des variables internes :
- i : indice portant sur la liste des ressources.
- n : nom de ressource.
- exact : booléen indiquant si la ressource est connue du système ou non.
- occupé : booléen indiquant si la ressource est déjà occupée pendant la durée de l'activité courante.

### 4.4.5. Procédure "TEST-LST-RESSOURCE".

entrées :
- date : date de l'activité.
- heure : heure de début de l'activité.
- durée : durée de l'activité.
- l : liste de ressources connues du système.

sorties :
- exact : booléen indiquant si la liste des ressources est disponible pendant toute la durée de l'activité.

effet :
cette procédure teste si la liste des ressources est disponible pendant toute la durée de l'activité.

pré-conditions :
(date) et (heure) et (durée) et
(l sont des ressources connues du système)

post-conditions :
((exact) et
(toutes les ressources sont disponibles)) ou
((pas exact) et
(au moins une des ressources n'est plus disponible))

procédures appelantes :
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE

procédures appelées :
TEST-DISPONIBLE
ERREUR08

algorithme logique :

```
exact := true;
pour i := 1 à 4 faire :
  si (l [i] <> '        ') alors :
    TEST-DISPONIBLE (date, heure, durée,
                     l [i], occupé);
    si occupé alors :
      exact := false;
      ERREUR08 (l [i]);
```

spécification des variables internes :
- i : indice portant sur la liste des ressources.
- occupé : booléen indiquant si la ressource est occupée pendant la durée de l'activité courante.

4.4.6. Procédure "SELECT-TYPE".

entrées :
aucune

sorties :
- type : type d'activité.

effet :
cette procédure demande si l'abonné veut sélectionner un type d'activités et si oui lequel.

pré-conditions :
aucune

post-conditions :
(type = ' ' si pas de sélection) ou
(type <> ' ' sinon)

procédures appelantes :
VUE-AGENDA

procédures appelées :
R-REPONSE

algorithme logique :
type := ' ';
répéter :
R-REPONSE (réponse);
jusque (réponse = 'o') ou (réponse = 'n');
si (réponse = 'o') alors : rentrer type;

spécification des variables internes :
- réponse : caractère indiquant si l'abonné veut sélectionner un type ('o') ou non ('n').

4.4.7. Procédure "R-INTERVALLE".

entrées :
- date-du-jour : date du jour sous la forme AAMMJJ.
- numéro : caractère indiquant l'option choisie par l'abonné :
  = '1' si vue d'un jour,
  = '2' si vue d'une semaine,
  = '3' si vue d'un mois,
  = '4' si vue d'un intervalle quelconque.

sorties :
- date-début : date du début de l'intervalle à visualiser.
- date-fin : date de fin de l'intervalle à visualiser.

données globales :
- maxmois : tableau comprenant le nombre maximum de jours pour chaque mois.

effet :
cette procédure demande à l'abonné les dates de début et/ou de fin d'intervalle selon l'option choisie.

pré-conditions :
(date-du-jour) et (numéro > 0) et (numéro < 5)

post-conditions :
(date-début >= date-du-jour - 300) et
(date-fin >= date-début)

procédures appelantes :
VUE-AGENDA

procédures appelées :
R-DATE
DATE-PASSEE
VALID-DATE
DECOMP-DATE
ERREUR10

```
algorithme logique :
     selon numéro :
      - '1' : répéter :
                R-DATE (date-début, exact);
                si exact alors :
                  DATE-PASSEE (date-début, date-du-jour,
                               exact);
              jusque exact;
              date-fin := date-début;
      - '2' : répéter :
                R-DATE (date-début, exact);
                si exact alors :
                  DATE-PASSEE (date-début, date-du-jour,
                               exact);
              jusque exact;
              date-fin := date-début + 6;
              VALID-DATE (date-fin);
      - '3' : répéter :
                R-DATE (date-début, exact);
                si exact alors :
                  DATE-PASSEE (date-début, date-du-jour,
                               exact);
              jusque exact;
              DECOMP-DATE (date-début, jo, mo, an);
              date-début := date-début - jo + 1;
              date-fin := date-début - 1 + maxmois [mo];
      - '4' : répéter :
                R-DATE (date-début, exact);
                si exact alors :
                  DATE-PASSEE (date-début, date-du-jour,
                               exact);
              jusque exact;
              répéter :
                R-DATE (date-fin, exact);
                si exact alors :
                  si (date-fin < date-début) alors :
                    exact := false;
                    ERREUR10;
              jusque exact;

spécification des variables internes :
     - exact : booléen indiquant si la date rentrée est
               correcte ou non.
     - jo : jour de la date courante.
     - mo : mois de la date courante.
     - an : année de la date courante.
```

4.4.8. Procédure "SELECT-ACTIVITES".

entrées :
- n : nom de l'abonné.
- no : indique si l'abonné consulte l'agenda d'un abonné (0) ou d'une ressource (1).
- nom : nom de l'abonné ou de la ressource dont l'abonné veut visualiser l'agenda.
- type : type d'activité à sélectionner (= ' ' si pas de sélection).
- date-début : date du début de l'intervalle.
- date-fin : date de fin de l'intervalle.

sorties :
- i : nombre d'activités sélectionnées.

données globales :
- mat-date, mat-heure, mat-duree, mat-descr : tableaux comprenant respectivement les dates, les heures, les durées et les descriptions d'un ensemble d'activités.

effet :
cette activité sélectionne les activités concernant l'abonné ou la ressource, appartenant à un certain type si celui-ci a été spécifié et à l'intervalle donné.

pré-conditions :
(n est un abonné) et
(((no = 0) et (nom est un abonné) et
(n a accès à l'agenda de nom)) ou
((no = 1) et
(nom est une ressource connue du système))) et
((type = ' ' si pas de sélection de type) ou
(type <> ' ' si sélection d'un type donné)) et
(date-début) et (date-fin)

post-conditions :
(i >= 0)

procédures appelantes :
VUE-AGENDA

procédures appelées ;
OUVRIR-ACTIVITE
LIRE-ACTIVITE
FERMER-ACTIVITE

```
algorithme logique :
     i := 0;
     OUVRIR-ACTIVITE;
     LIRE-ACTIVITE (a, fin);
     tant que (pas fin) faire :
       si ((type = '          ') ou (type = a.type)) et
          (date-début <= a.date-activ) et
          (date-fin >= a.date-activ) alors :
         exact := false;
         si no = 0
           alors :
             si (a.nom-enreg = nom) alors :
               exact := true;
             si (a.nom-resp = nom) alors :
               exact := true;
             j := 0;
             tant que (pas exact) et (j < 10) faire :
               j := j + 1;
               si (a.nom-pers-cons [j] = nom) et
                  (a.code-réponse [j] = 'o') alors :
                 exact := true;
             si (a.confid = 'o') et (nom <> n) alors :
               a.descr := 'xxxxxx ...  xxxx';
           sinon :
             pour j := 1 à 4 faire :
               si (a.nom-ress-cons [j] = nom) alors :
                 exact := true;
         si exact alors :
           i := i + 1;
           mat-date [i] := a.date-activ;
           mat-heure [i] := a.heure;
           mat-durée [i] := a.durée;
           mat-descr [i] := a.descr;
       LIRE-ACTIVITE (a, fin);
     FERMER-ACTIVITE;
```

spécification des variables internes :

- a : enregistrement du fichier des activités.
- fin : booléen indiquant si l'on est à la fin du fichier des activités ou non.
- exact : booléen indiquant si l'activité courante doit être sélectionnée ou non.
- j : indice portant sur la liste des personnes concernées et sur la liste des ressources consommées.

### 4.4.9. Procédure "TRI-ACTIVITES".

entrées :

- i : nombre d'activités sélectionnées.

sorties :

aucune

données globales :
- mat-date, mat-heure, mat-durée, mat-descr : tableaux comprenant respectivement les dates, les heures, les durées et les descriptions d'un ensemble d'activités.

effet :
cette procédure trie les i activités sélectionnées par ordre croissant de date, heure, durée.

pré-conditions :
(i >= 0)

post-conditions :
aucune

procédures appelantes :
VUE-AGENDA

procédures appelées :
aucune

algorithme logique :

```
j := 1;
tant que (j < i) faire :
  l := j;
  k := j + 1;
  tant que (k <= i) faire :
    si (mat-date [l] > mat-date [k]) ou
       ((mat-date [l] = mat-date [k]) et
        ((mat-heure [l] > mat-heure [k]) ou
         ((mat-heure [l] = mat-heure [k]) et
          (mat-durée [j] > mat-durée [k]))))
      alors : l := k;
    k := k + 1;
  si (l <> j) alors :
    permuter mat-date [j] et mat-date [l];
    permuter mat-heure [j] et mat-heure [l];
    permuter mat-durée [j] et mat-durée [l];
    permuter mat-descr [j] et mat-descr [l];
  j := j + 1;
```

spécification des variables internes :
- j, k, l : indices portant sur mat-date, mat-heure, mat-durée et mat-descr.

### 4.4.10. Procédure "AFF-SEMAINE".

entrées :
- nom : nom de l'abonné ou de la ressource dont on va afficher une partie de l'agenda.
- type : type d'activité sélectionnée (= ' ' si pas de sélection).
- date-début : date de début de l'intervalle.
- date-fin : date de fin de l'intervalle.
- i : nombre d'activités sélectionnées.

```
sorties :
     aucune

effet ;
     cette procédure affiche les moments d'une semaine
     pendant lesquels l'abonné ou la ressource connue du
     système est occupée à un type d'activité donné si
     celui-ci est spécifié et permet de visualiser de façon
     détaillée une journée de cette semaine.

pré-conditions :
     (nom est un abonné ou une ressource connue du système)
     et
     ((type = '          ' si pas de type sélectionné) ou
      (type <> '          ' si un type d'activité donné)) et
     (date-début) et (date-fin)

post-conditions :
     aucune

procédures appelantes :
     VUE-AGENDA

procédures appelées :
     PREPAR-SEMAINE
     AFFICH-SEMAINE
     R-NOMBRE
     AFF-JOUR

algorithme logique :
     PREPAR-SEMAINE (date-début, date-fin, i, table);
     répéter :
       AFFICH-SEMAINE (nom, type, date-début, date-fin,
                       table, j1, j2);
       afficher les options possibles (visualiser une
       journée de la semaine ou retourner au menu
       précédent);
       R-NOMBRE (choix);
       si (choix >= j1) et (choix <= j2)
         alors :
           date := date-début - j1 + choix;
           AFF-JOUR (nom, type, date, j);
         sinon :
           si (j2 < j1) alors :
             DECOMP-DATE (date-début, j1, mo, an);
             si (choix >= j1) et (choix <= maxmois [mo])
               alors :
                 date := date-début - j1 + choix;
                 AFF-JOUR (nom, type, date, i);
               sinon :
                 si (choix <= j2) et (choix > 0) alors :
                   date := date-fin -j2 + choix;
                   AFF-JOUR (nom, type, date, i);
     jusque (choix = 99);
```

spécification des variables internes :
- j1 : jour de la date de début d'intervalle.
- mo : mois de la date de début d'intervalle.
- an : année de la date de début d'intervalle.
- j2 : jour de la date de fin d'intervalle.
- choix : option choisie.
- date : date de la journée à visualiser en détail.
- table : table permettant de visualiser une semaine d'un agenda.

4.4.11. Procédure "AFF-MOIS".

entrées :
- nom : nom de l'abonné ou de la ressource dont on va afficher une partie de l'agenda.
- type : type d'activité sélectionnée (= ' ' si pas de sélection).
- date-début : date de début de l'intervalle.
- date-fin : date de fin de l'intervalle.
- i : nombre d'activités sélectionnées.

sorties :
aucune

effet :
cette procédure affiche les jours d'un mois pendant lesquels l'abonné ou la ressource connue du sytème est occupée à au moins une activité d'un type donné si celui-ci a été spécifié et permet de visualiser en détail une journée de ce mois.

pré-conditions ;
(nom est un aboneé ou une ressource connue du système)
et
((type = ' ' si pas de type sélectionné) ou
(type <> ' ' si un type sélectionné)) et
(date-début) et (date-fin)

post-conditions :
aucune

procédures appelantes :
VUE-AGENDA

procédures appelées :
PREPAR-MOIS
AFFICH-MOIS
R-REPONSE
AFF-JOUR

algorithme logique :
PREPARER-MOIS (date-début, date-fin, i, table);
répéter :
AFFICH-MOIS (nom, type, date-début, date-fin, table, j1, j2);
afficher les options possibles (visualiser une journée du mois ou retourner au menu précédent);
R-NOMBRE (choix);
si (choix >= j1) et (choix <= j2) alors :
date := date-début - 1 + choix;
AFF-JOUR (nom, sorte, date, i);
jusque (choix = 99);

spécification des variables internes :
- j1 : jour de la date de début d'intervalle.
- j2 : jour de la date de fin d'intervalle.
- choix : option choisie.
- date : date de la journée à visualiser en détail.
- table : table permettant de visualiser un mois d'un agenda.

4.4.12. Procédure "AFF-INTERVALLE".

entrées :
- nom : nom de l'abonné ou de la ressource connue du système dont on va afficher une partie de l'agenda.
- type : type d'activité sélectionnée (= ' ' si pas de sélection).
- date-début : date de début de l'intervalle.
- date-fin : date de fin de l'intervalle.

sorties :
aucune

effet :
cette procédure affiche successivement et de façon détaillée les journées d'un intervalle donné.

pré-conditions :
(nom est un abonné ou une ressource connue du système)
et
((type = ' ' si pas de type sélectionné) ou
(type <> ' ' si un type sélectioné)) et
(date-début) et (date-fin)

post-conditions :
aucune

procédures appelantes :
VUE-AGENDA

procédures appelées :
AFF-JOUR
VALID-DATE

algorithme logique :
```
date := date-début;
tant que (date <= date-fin) faire :
  AFF-JOUR (nom, type, date, i);
  date := date + 1;
  VALID-DATE (date);
```

spécification des variables internes :
- date : date de la journée à visualiser en détail.

## 4.5. Cinquième niveau.

### 4.5.1. Procédure "R-DATE".

entrées :
aucune

sorties :
- date : date sous la forme AAMMJJ.
- exact : boolèen indiquant si la date est correcte ou non.

effet :
cette procédure demande de rentrer une date sous la forme JJ/MM/AA et teste si celle-ci est correcte.

pré-conditions :
aucune

post-conditions :
((date correcte) et (exact)) ou (pas exact)

procédures appelantes :
RENTREE-DATE-DU-JOUR
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE
RECH-PLAGE-TPS-LIBRE
R-INTERVALLE

procédures appelées :
TEST-DATE

algorithme logique :
exact := false;
rentrer d;
vérifier d est sous la forme JJ/MM/AA et transférer dans date;
si exact alors : TEST-DATE (date, exact);

spécification des variables internes :
- d : date sous la forme JJ/MM/AA.

### 4.5.2. Procédure "R-HEURE".

entrées :
aucune

sorties :
- heure : heure sous la forme HH.MM.
- exact : boolèen indiquant si la date est correcte ou non.

effet :
cette procédure demande de rentrer une heure sous la forme HH.MM et teste si celle-ci est correcte.

pré-conditions :
aucune

post-conditions :
((heure correcte) et (exact)) ou (pas exact)
procédures appelantes :
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE

procédures appelées :
TEST-HEURE

algorithme logique :
exact := false;
rentrer h;
vérifier si h est sous la forme HH.MM et transférer dans heure;
si exact alors : TEST-HEURE (heure, exact);

spécification des variables internes :
h : heure sous la forme HH.MM.

### 4.5.3. Procédure "R-NOMBRE".

entrées :
aucune

sorties :
- nombre : nombre de deux chiffres maximum.

effet :
cette procédure demande de rentrer un nombre de deux chiffres maximum.

pré-conditions :
aucune

post-conditions :
(nombre >= 0) et (nombre <= 99)

procédures appelantes :
REPETER-ACTIVITE
AFF-SEMAINE
AFF-MOIS

procédures appelées :
aucune

algorithme logique :
nombre := 0;
rentrer n;
vérifier si n comporte 1 ou 2 chiffres et transférer dans nombre;

spécification des variables internes :
- n : nombre de deux caractères.

4.5.4. Procédure "R-DESCRIPTION".

entrées :
aucune

sorties :
- description : texte libre de maximum deux fois 64 caractères.

effet :
cette procédure demande un texte libre de maximum deux fois 64 caractères.

pré-conditions :
aucune

post-conditions :
description

procédures appelantes :
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE

procédures appelées :
aucune

algorithme logique :
rentrer description;

spécification des variables internes :
aucune

4.5.5. Procédure "R-PERSONNE".

entrées :
aucune

sorties :
- nom : nom de personne.
- exact : booléen indiquant si la personne est un abonné ou non.

effet :
cette procédure demande le nom d'une personne et teste si celle-ci est un abonné.

pré-conditions :
aucune

post-conditions :
((n est un abonné) et (exact)) ou (pas exact)

procédures appelantes :
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE
AJOUT-ABONNE
AJOUT-RESSOURCE
MODIF-RESPONSABLE
R-LST-PERSONNE

procédures appelées :
OUVRIR-PERSONNE
LIRE-PERSONNE
FERMER-PERSONNE

algorithme logique :
rentrer n;
exact := false;
OUVRIR-PERSONNE;
LIRE-PERSONNE (p, fin);
tant que (pas fin) et (pas exact) faire :
si (p.nom-pers = n)
alors : exact := true;
sinon : LIRE-PERSONNE (p, fin);
FERMER-PERSONNE;

spécification des variables internes :
- p : enregistrement du fichier des abonnés.
- fin : booléen indiquant si l'on est a la fin du fichier des abonnés ou non.

4.5.6. Procédure "R-RESSOURCE".

entrées :
aucune

sorties :
- n : nom de ressource.
- exact : booléen indiquant si la ressource est connue du système ou non.

effet :
cette procédure demande de rentrer un nom de ressource et teste si celle-ci est connue du système.

pré-conditions :
aucune

post-conditions :
((n est une ressource connue du système) et (exact)) ou (pas exact)

procédures appelantes :
VUE-AG-RESSOURCE
AJOUT-RESSOURCE
R-LST-RESSOURCE
R-T-LST-RESSOURCE

procédures appelées :
OUVRIR-RESSOURCE
LIRE-RESSOURCE
FERMER-RESSOURCE

algorithme logique :
rentrer n;
exact := false;
OUVRIR-RESSOURCE;
LIRE-RESSOURCE (r, fin);
tant que (pas fin) et (pas exact) faire :
si (r.nom-ress = n)
alors : exact := true;
sinon : LIRE-RESSOURCE (r, fin);
FERMER-RESSOURCE;

spécification des variables internes :
- r : enregistrement du fichier des ressources.
- fin : booléen indiquant si l'on est à la fin du fichier des abonnés ou non.

4.5.7. Procédure "TEST-DISPONIBLE".

entrées :
- date : date de l'activité.
- heure : heure de l'activité.
- durée : durée de l'activité.
- n : nom de ressource.

sorties :
- occupé : booléen indiquant si cette ressource est disponible pendant toute la durée de l'activité ou non.

effet :
cette procédure teste si la ressource est disponible pendant toute la durée de l'activité.

pré-conditions :
(date) et (heure) et (durée) et (n)

post-conditions :
((n est disponible pendant toute la durée de l'activité) et (pas occupé)) ou
((n n'est pas disponible pendant toute la durée de l'activité) et (occupé))

procédures appelantes :
CONS-MAJ-ACTIVITE
R-T-LST-RESSOURCE
TEST-LST-RESSOURCE

procédures appelées :
OUVRIR-ACTIVITE
LIRE-ACTIVITE
FERMER-ACTIVITE

```
algorithme logique :
     occupé := false;
     OUVRIR-ACTIVITE;
     LIRE-ACTIVITE (a, fin);
     tant que (pas fin) et (pas occupé) faire :
       pour j := 1 à 4 faire :
         si (n = a.nom-pers-cons [j]) et
            (date = a.date-activ) alors :
           si ((heure >= a.heure) et
               (heure < a.heure + a.durée)) ou
              ((a.heure >= heure) et
               (a.heure < heure + durée)) alors :
             occupé := true;
       LIRE-ACTIVITE (a, fin);
```

spécification des variables internes :

- a : enregistrement du fichier des activités.
- fin : booléen indiquant si l'on est à la fin du fichier des activités ou non.

## 4.5.8. Procédure "DATE-PASSEE".

entrées :

- date : date sous la forme **AAMMJJ**.
- date-du-jour : date du jour sous la forme **AAMMJJ**.

sorties :

- exact : booléen indiquant si la date est vieille de plus de 3 mois ou non.

effet :

cette procédure teste si la date est vieille de plus de 3 mois.

pré-conditions :

(date) et (date-du-jour)

post-conditions :

((date < date-du-jour - 300) et (pas exact)) ou
((date >= date-du-jour - 300) et (exact))

procédures appelantes :

R-INTERVALLE

procédures appelées :

ERREUR13

```
algorithme logique :
     si (date < date-du-jour - 300)
       alors :
         exact := false;
         ERREUR13;
       sinon :
         exact := true;
```

spécification des variables internes :
aucune

4.5.9. Procedure "AFF-JOUR".

entrées :
- nom : nom de l'abonné ou de la ressource connue du système.
- type : type d'activité sélectionnée (= ' ' si pas de sélection).
- date : date.
- i : nombre d'activités sélectionnées.

sorties :
aucune

effet :
cette procédure affiche de façon détaillée les activités de l'abonné ou de la ressource connue du système correspondant au type donné si celui-ci a été spécifié et à la date donnée.

pré-conditions :
(nom est un abonné ou une ressource connue du système)
et
((type = ' ' si pas de type sélectionnés) ou
(type <> ' ' si un type sélectionné)) et
(date) et (i >= 0)

post-conditions :
aucune

procédures appelantes :
VUE-AGENDA
AFF-SEMAINE
AFF-MOIS
AFF-INTERVALLE

procédures appelées :
DECOMP-DATE
TROUVE-JOUR
AFFICH-JOUR

algorithme logique :
```
afficher nom;
afficher type;
DECOMP-DATE (date, jo, mo, an);
TROUVE-JOUR (jo, mo, an, jj);
affiche la date;
j := 0
tant que (j < i) faire :
  j := j + 1;
  si (date = mat-date [j]) alors :
    AFFICH-JOUR (mat-heure [j], mat-durée [j],
                 mat-descr [j]);
```

spécification des variables internes :

- jo : jour de la date.
- mo : mois de la date.
- an : annèe de la date.
- jj : jour de la semaine de la date.

## 4.6. Sixième niveau.

### 4.6.1. Procédure "VALID-DATE".

entrées :
- date : date sous la forme AAMMJJ.

sorties :
aucune

données globales :
- maxmois : tableau comprenant le nombre maximum de jours pour chaque mois.

effet :
cette procédure corrige la date en passant au mois suivant si cela est nécessaire.

pré-conditions :
date

post-conditions :
date correcte

procédures appelantes :
AJOUT-ACTIVITE
RECH-PLAGE-TPS-LIBRE
R-INTERVALLE
AFFICH-INTERVALLE

procédures appelées :
DECOMP-DATE

algorithme logique :

```
DECOMP-DATE (date, jo, mo, an);
si (jo > maxmois [mo]) alors :
  date := date + 100 - maxmois [mo];
  si (mo > 11) alors :
    date := date + 8800;
```

spécification des variables internes :
- jo : jour de la date.
- mo : mois de la date.
- an : année de la date.

### 4.6.2. Procédure "TEST-DATE".

entrées :
- date : date.

sorties :
- exact : booléen indiquant si la date est correcte ou non.

données globales :
- maxmois : tableau comprenant le nombre maximum de jours pour chaque mois.

effet :
cette procédure teste si la date est correcte.

pré-conditions :
date

post-conditions :
((date est correcte) et (exact)) ou
((date n'est pas correcte) et (pas exact))

procédures appelantes :
R-DATE

procédures appelées :
DECOMP-DATE

algorithme logique :

```
DECOMP-DATE (date, jo, mo, an);
si (mo <= 12)
  alors :
    si (jo <= maxmois [mo]) et (jo > 0)
      alors : exact := true;
      sinon : exact := false;
  sinon :
    exact := false;
```

spécification des variables internes :
- jo : jour de la date.
- mo : mois de la date.
- an : année de la date.

### 4.6.3. Procédure "TEST-HEURE".

entrées :
- heure : heure.

sorties :
- exact : booléen indiquant si l'heure est correcte ou non.

effet :
cette procédure teste si la date est correcte.

pré-conditions :
heure

post-conditions :
((heure est correcte) et (exact)) ou
((heure n'est pas correcte) et (pas exact))

procédures appelantes :
R-HEURE

procédures appelées :
aucune

algorithme logique :
si (heure >= 0) et (heure < 24)
alors :
si (heure - trunc (heure) < 0.60)
alors : exact := true;
sinon : exact := false;
sinon :
exact := false;

spécification des variables internes :
aucune

4.6.4. Procédure "AFFICH-JOUR".

entrées :
- heure : heure de début d'activité.
- durée : durée estimée d'activité.
- descr : description d'activité.

sorties ;
aucune

effet :
cette procédure affiche l'heure, la durée et la description d'un activité.

pré-conditions :
(heure) et (durée) et (descr)

post-conditions :
aucune

procédures appelantes :
AFF-JOUR

procédures appelées :
DECOMP-HEURE

algorithme logique :
DECOMP-HEURE (heure, he, min);
afficher l'heure;
DECOMP-HEURE (durée, he, min);
afficher la durée;
afficher descr;

spécification des variables internes :
- he : partie entière de l'heure.
- min : minutes de l'heure.

4.6.5. Procédure "PREPAR-SEMAINE".

entrées :
- date-début : date de début d'intervalle.
- date-fin : date de fin d'intervalle.
- i : nombre d'activités sélectionnées.

sorties :
- table : table permettant de visualiser une semaine d'un agenda.

effet :
cette procédure contruit la table à visualiser en fonction des activités sélectionnées.

pré-conditions :
(date-début) et (date-fin) et (i)

post-conditions :
table

procédures appelantes :
AFF-SEMAINE

procédures appelées :
DECOMP-DATE
TROUVE-JOUR
DECOMP-HEURE

algorithme logique :

```
j := 0;
tant que (j < i) faire :
  j := j + 1;
  si (date-début <= mat-date [j]) et
     (date-fin >= mat-date [j]) alors :
    DECOMP-DATE (mat-date [j], jo, mo, an);
    TROUVE-JOUR (jo, mo, an, jj);
    DECOMP-HEURE (mat-heure [j], he, min);
    DECOMP-HEURE (mat-durée [j], he, min);
    mettre à jour la table;
```

spécification des variables internes :
- j : indice portant sur mat-date, mat-heure, mat-durée.
- jo : jour de la date.
- mo : mois de la date.
- an : année de la date.
- he : partie entière de l'heure.
- min : minutes de l'heure.

4.6.6. Procédures "AFFICH-SEMAINE".

entrées :
- nom : nom de l'abonné ou de la ressource dont on va afficher une partie de l'agenda.
- type : type d'activités sélectionnées (= ' ' si pas de sélection).
- date-début : date de début d'intervalle.
- date-fin : date de fin d'intervalle.
- table : table permettant de visualiser une semaine de l'agenda.

sorties :
- j1 : jour de la date de début d'intervalle.
- j2 : jour de la date de fin d'intervalle.

effet :
cette procédure affiche les moments d'une semaine pendant lesquels l'abonné ou la ressource connue du système est occupée à un type d'activités si celui-ci est spécifié.

pré-conditions :
(nom est un abonné ou une ressource connue du sytème) et
((type = ' ' si pas de type selectionné) ou
(type <> ' ' si un type sélectionné)) et
(date-début) et (date-fin) et (table)

post-conditions :
(j1) et (j2)

procédures appelantes :
AFF-SEMAINE

procédures appelées :
DECOMP-DATE
TROUVE-JOUR

algorithme logique :
```
afficher nom;
afficher type;
DECOMP-DATE (date-début, j1, mo, an);
TROUVE-JOUR (j1, mo, an, jj);
afficher la date de début d'intervalle;
DECOMP-DATE (date-fin, j2, mo, an);
afficher la date de fin d'intervalle;
afficher la table;
```

spécification des variables internes :
- mo : mois de la date de début et de fin d'intervalle.
- an : année de la date de début et de fin d'intervalle.

4.6.7. Procédure "PREPAR-MOIS".

entrées :
- date-début : date de début d'intervalle.
- date-fin : date de fin d'intervalle.
- i : nombre d'activités sélectionnées.

sorties :
- table : table permettant de visualiser un mois d'un agenda.

effet :
cette procédure construit la table à visualiser en fonction des activités sélectionnées.

pré-conditions :
(date-début) et (date-fin) et (i)

post-conditions :
table

procédures appelantes :
AFF-MOIS

procédures appelées :
DECOMP-DATE

algorithme logique :
```
j := 0;
tant que (j < i) faire :
  j := j + 1;
  si (date-début <= mat-date [j]) et
     (date-fin >= mat-date [j]) alors :
    DECOMP-DATE (mat-date [j], jo, mo, an);
    mise à jour de la table;
```

spécification des variables internes :
- j : indice portant sur mat-date.
- jo : jour de la date.
- mo : mois de la date.
- an : année de la date.

4.6.8. Procédure "AFFICH-MOIS".

entrées :
- nom : nom de l'abonné ou de la personne dont on va afficher une partie de l'agenda.
- type : type d'activités sélectionnées (= ' ' si pas de sélection.
- date-début : date de début d'intervalle.
- date-fin : date de fin d'intervalle.
- table : table permettant de visualiser un mois de l'agenda.

sorties :
- j1 : jour de la date de début d'intervalle.
- j2 : jour de la date de fin d'intervalle.

effet :
cette procédure affiche les jours d'un mois pendant lesquels l'abonné ou la ressource connue du système est occupée à au moins une activité d'un type donné si celui-ci a été spécifié.

pré-conditions :
(nom est un abonné ou une ressource connue du système)
et
((type = ' ' si pas de type sélectionné) ou
(type <> ' ' si un type sélectionné)) et
(date-début) et (date-fin) et (table)

post-conditions :
(j1) et (j2)

procédures appelantes :
AFF-MOIS

procédures appelées :
DECOMP-DATE
TROUVE-JOUR

algorithme logique :
```
afficher nom;
afficher type;
DECOMP-DATE (date-début, j1, mo, an);
DECOMP-DATE (date-fin, j2, mo, an);
afficher la date;
TROUVE-JOUR (j1, mo, an, jj);
afficher la table;
```

spécification des variables internes :
mo : mois de la date.
an : année de la date.
jj : jour de la semaine de la date.

## 4.7. Septième niveau.

### 4.7.1. Procédure "DECOMP-DATE".

entrées :
- date : date sous la forme AAMMJJ.

sorties :
- jo : jour de la date.
- mo : mois de la date.
- an : année de la date.

données globales :
- maxmois : tableau comprenant le nombre maximum de jours pour chaque mois.

effet :
cette procédure décompose une date de la forme AAMMJJ en 3 parties : jour, mois et année; et modifie le nombre maximum de jours au mois de février si cela est nécessaire.

pré-conditions :
date

post-conditions :
(jo) et (mo) et (an) et
((maxmois [2] = 29 si an mod 4 = 0) ou
(maxmois [2] = 28 si an mod 4 <> 0))

procédures appelantes :
RENTREE-DATE-DU-JOUR
VISUAL-ACTIVITES
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE
RECH-PLAGE-TPS-LIBRE
R-INTERVALLE
VALID-DATE
TEST-DATE
AFF-JOUR
PREPAR-SEMAINE
AFFICH-SEMAINE
PREPAR-MOIS
AFFICH-MOIS

procédures appelées :
aucune

```
algorithme logique :
     an := 1900 + trunc (date / 10000);
     date := date - an * 10000;
     mo := trunc (date / 100);
     date := date - mo * 100;
     jo := trunc (date);
     si (an mod 4 = 0)
       alors : maxmois [2] := 29;
       sinon : maxmois [2] := 28;

spécification des variables internes :
     aucune
```

4.7.2. Procédure "TROUVE-JOUR".

entrées :
- jo : jour d'une date.
- mo : mois d'une date.
- an : année d'une date.

sorties :
- jj : jour de la semaine de la date.

données globales :
- maxmois : tableau comprenant le nombre maximum de jours pour chaque mois.
- début : tableau comprenant le numéro du jour de la semaine du premier jour des années d'un cycle.

effet :
cette procédure calcule le jour de la semaine d'une date donnée sous la forme : jour, mois et année.

pré-conditions :
(jo) et (mo) et (an >= 1601)

post-conditions :
(jj >= 1) et (jj <= 7)

procédures appelantes :
- RENTREE-DATE-DU-JOUR
- VISUAL-ACTIVITES
- AJOUT-ACTIVITE
- CONS-MAJ-ACTIVITE
- RECH-PLAGE-TPS-LIBRE
- AFF-JOUR
- PREPAR-SEMAINE
- AFFICH-SEMAINE
- AFFICH-MOIS

procédures appelées ;
aucune

```
algorithme logique :
     jj := début((an - 1601) mod 28 + 1) + jo - 1;
     i := 1
     tant que (i < mo) faire :
       jj := jj + maxmois [i];
       i := i + 1;
     jj := (jj - 1) mod 7 + 1;

spécification des variables internes :
     - i : indice portant sur maxmois.
```

### 4.7.3. Procédure "DECOMP-HEURE".

```
entrées :
     - heure : heure sous la forme HH.MM.

sorties :
     - he : partie entière de l'heure.
     - min : minutes de l'heure.

effet :
     cette procédure décompose une heure de la  forme  HH.MM
     en deux parties : heures, minutes.

pré-conditions :
     heure

post-conditions :
     (he) et (min)

procédures appelantes :
     VISUAL-ACTIVITES
     CONS-MAJ-ACTIV
     RECH-PLAGE-TPS-LIBRE
     AFFICH-JOUR
     PREPAR-SEMAINE

procédures appelées :
     aucune

algorithme logique :
     he := trunc (heure);
     min := trunc ((heure - he) * 100);

spécification des variables internes :
     aucune
```

4.7.4. Procédure "R-REPONSE".

entrées :
aucune

sorties :
- réponse : caractère.

effet :
cette procédure demande de rentrer un caractère; si plusieurs caractères sont rentrés, seul le premier est pris en compte.

pré-conditions :
aucune

post-conditions :
réponse

procédures appelantes :
RENTREE-DATE-DU-JOUR
VISUAL-ACTIVITES
AJOUT-ACTIVITE
CONS-MAJ-ACTIV
REPETER-ACTIVITE
SELECT-TYPE

procédures appelées :
aucune

algorithme logique :
réponse := ' ';
rentrer réponse;

spécification des variables :
aucune

## 4.8. Huitième niveau.

### 4.8.1. Procédure "ERREUR01".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : nom et/ou mot de passe incorrect.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
IDENT-ABONNE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

### 4.8.2. Procédure "ERREUR02".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : nom de ressource incorrect.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
MODIF-RESPONSABLE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.3. Procédure "ERREUR03".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : cette personne n'est pas connue du système.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
CONS-MAJ-ACTIVITE
AJOUT-RESSOURCE
ACCEDE-AGENDA
R-LST-PERSONNE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.4. Procédure "ERREUR04".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : cette ressource n'est pas connue du système.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
VUE-AG-RESSOURCE
R-LST-RESSOURCE
R-T-LST-RESSOURCE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.5. Procédure "ERREUR05".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : ce nom de personne existe déjà.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
AJOUT-ABONNE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.6. Procédure "ERREUR06".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : ce nom de ressource existe déjà.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
AJOUT-RESSOURCE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.7. Procédure "ERREUR07".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : cette ressource n'est pas disponible à ce moment précis.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
R-T-LST-RESSOURCE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.8. Procédure "ERREUR08".

entrées :
- nom : nom de ressource connue du système.

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : la ressource 'nom' n'est plus disponible.

pré-conditions :
nom est une ressource connue du système

post-conditions ;
aucune

procédures appelantes :
TEST-LST-RESSOURCE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.9. Procédure "ERREUR09".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : cette date est déjà passée.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
AJOUT-ACTIVITE
CONS-MAJ-ACTIVITE
RECH-PLAGE-TPS-LIBRE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.10. Procédure "ERREUR10".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : cette date est inférieure à la date du début d'intervalle.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
R-INTERVALLE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.11. Procédure "ERREUR11".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : cette date est inférieure à une date déjà rentrée.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
RENTREE-DATE-DU-JOUR

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.12. Procédure "ERREUR12".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : vous n'avez pas accès à l'agenda de cette personne.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
ACCEDE-AGENDA

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

4.8.13. Procédure "ERREUR13".

entrées :
aucune

sorties :
aucune

effet :
cette procédure affiche le message d'erreur : cette date ne fait plus partie de l'agenda.

pré-conditions :
aucune

post-conditions ;
aucune

procédures appelantes :
DATE-PASSEE

procédures appelées :
aucune

algorithme logique :
afficher le message d'erreur;

spécification des variables :
aucune

# Chapitre 4: MANUEL UTILISATEUR.

## 1. Introduction.

On décrit ici la marche à suivre pour une bonne utilisation du gestionnaire d'agendas.

Le gestionnaire d'agendas se compose de deux programmes principaux : le premier regroupant les traitements sur les agendas (AGENDA) et le second les traitements relatifs aux abonnés et ressources (GESTION).

On trouvera en annexe le texte des différents programmes.

## 2. Le programme "AGENDA".

Ce programme regroupe les differents traitements possibles sur les agendas.

### 2.1. Chargement du programme.

Une fois connecté au système PDP 11, le caractère '%' apparaît à l'écran et l'utilisateur introduit par le clavier la commande 'EM1 AGENDA' qu'il termine en appuyant sur la touche <RETURN>.

L'exécution du programme AGENDA est ainsi lancée (2.2).

### 2.2. Identification.

Après un petit temps d'attente, un nom et un mot de passe sont demandés... L'utilisateur les introduit par le clavier, chacun d'entre eux étant terminé en appuyant sur la touche <RETURN> (seuls les huit premiers caractères du nom et du mot de passe sont pris en considération).

Si le nom et le mot de passe introduits sont reconnus par le programme, il sera alors demandé d'introduire la date du jour (2.3); sinon un nom et un mot de passe sont à nouveau demandés.

A la troisième erreur, l'exécution du programme AGENDA est arrêtée.

### 2.3. Introduction de la date du jour.

La date du jour est demandée... L'utilisateur l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche <RETURN>. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

La date introduite est alors vérifiée par le programme.

Si celle-ci est correcte, une confirmation est demandée... L'utilisateur répond par 'OUI' ou 'NON', et appuie ensuite sur la touche <RETURN> (seul le premier caractère est nécessaire).

Si l'utilisateur confirme la date introduite, le programme affichera la visualisation des activités inscrites dans son agenda depuis la dernière connexion ainsi que de celles n'ayant pas encore été confirmées (2.4).

Si la date introduite est incorrecte ou si l'utilisateur ne la confirme pas, celle-ci est à nouveau demandée.

## 2.4. Visualisation de nouvelles activités.

Le programme affiche les activités qui ont été proposées à l'utilisateur par d'autres abonnés depuis sa dernière connexion ainsi que celles n'ayant pas encore été confirmées sous la forme décrite page 1.7, et demande une réponse à leur propos... L'utilisateur répond par 'OUI', 'NON' ou rien, et appuie ensuite sur la touche ‹RETURN› (seul le premier caractère est nécessaire).

Si l'utilisateur répond 'OUI', cela signifie que l'activité sera définitivement ajoutée dans son agenda et cela sans tenir compte des autres activités déjà présentes dans son agenda (c'est donc l'utilisateur qui aura à charge de vérifier préalablement s'il est libre durant toute la durée de l'activité proposée).

S'il répond 'NON', cela signifie que l'activité ne sera pas ajoutée dans son agenda.

S'il ne répond rien, cela signifie qu'il ajourne sa réponse et que l'activité lui sera donc à nouveau proposée la prochaine fois qu'il se connectera au système.

S'il répond quoi que ce soit d'autre, la question est reposée jusqu'à introduction d'une réponse correcte.

De toute facon, au terme de cette visualisation, le programme lui proposera de choisir un des traitements qu'il peut effectuer (2.5).

## 2.5. Choix d'un traitement.

Le programme propose à l'utilisateur un des traitements suivants :

1. vue de l'agenda d'un abonné,
2. vue de l'agenda d'une ressource,
3. ajout d'une activité ou d'une série d'activités,
4. consultation et/ou mise à jour d'une activité,
5. recherche de plages de temps libre,
9. fin de travail.

Un choix est demandé... L'utilisateur introduit normalement par le clavier un des chiffres suivants : '1', '2', '3', '4', '5' ou '9', et appuie ensuite sur la touche ‹RETURN›.

Selon que l'utilisateur introduit '1', '2', '3', '4' ou '5', le programme passera respectivement à la vue de l'agenda d'un abonné (2.6), à la vue de l'agenda d'une ressource (2.7), à l'ajout d'une activité ou d'une série d'activités (2.8), à la consultation et/ou mise à jour d'une activité (2.9) ou à la recherche de plages de temps libre (2.10).

S'il introduit '9', il met ainsi fin à l'exécution du programme AGENDA.

S'il introduit n'importe quel autre caractère, un nouveau choix est demandé jusqu'à l'introduction d'un chiffre correct.

Si par mégarde, l'utilisateur a introduit plusieurs caractères, seul le premier est pris en compte.

## 2.6. Vue de l'agenda d'un abonné.

Un nom d'abonné est demandé... L'utilisateur l'introduit par le clavier et appuie ensuite sur la touche <RETURN> (seuls les 8 premiers caractères du nom sont pris en considération).

Le nom de l'abonné est ensuite analysé par le programme.

Si le nom de l'abonné est reconnu par le programme, celui-ci vérifie si l'utilisateur connecté a bien accès à l'agenda de l'abonné. Si oui, l'utilisateur a la permission de consulter l'agenda et aura la possibilité de sélectionner un type d'activités (2.6.1).

Si le nom de l'abonné n'est pas reconnu par le programme ou si l'utilisateur n'a pas accès à l'agenda de l'abonné, un autre nom d'abonné est demandé.

A la troisième erreur, le programme retournera au choix d'un traitement (2.5).

### 2.6.1. Sélection d'un type.

Le programme demande si l'utilisateur désire sélectionner un type d'activités... Il répond par 'OUI' ou 'NON', et appuie ensuite sur la touche <RETURN> (seul le premier caractère est nécessaire).

S'il répond 'OUI', il devra introduire ce type et appuyer ensuite sur la touche <RETURN> (seuls les 8 premiers caractères du type seront pris en considération).

Exemple : si l'utilisateur désire sélectionner un type d'activité et introduit le type 'COURS', cela signifie que, par la suite, l'utilisateur ne visualisera dans l'agenda que des activités appartenant au type 'COURS'.

De toute façon, le programme passera alors à la sélection d'un intervalle (2.6.2).

## 2.6.2. Sélection d'un intervalle.

Le programme propose à l'utilisateur une des possibilités suivantes :

1. vue d'un jour,
2. vue d' une semaine,
3. vue d'un mois,
4. vue d'un intervalle quelconque,
9. fin de visualisation.

Un choix est demandé... L'utilisateur introduit normalement par le clavier un des chiffres suivants : '1', '2', '3', '4' ou '9', et appuie ensuite sur la touche ‹RETURN›.

Selon que l'utilisateur introduit '1', '2', '3' ou '4', le programme passera respectivement à la vue d'un jour (2.6.3), à la vue d'une semaine (2.6.4), à la vue d'un mois (2.6.5) ou à la vue d'un intervalle quelconque (2.6.6).

S'il introduit '9', le programme retournera au choix d'un traitement (2.5).

S'il introduit n'importe quel autre caractère, un nouveau choix est demandé jusqu'à l'introduction d'un chiffre correct.

Si par mégarde, l'utilisateur introduit plusieurs caractères, seul le premier est pris en considération.

## 2.6.3. Vue d'un jour.

### 2.6.3.1. Introduction du jour.

La date du jour dont l'utilisateur veut voir les activités est demandée... L'utilisateur l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche ‹RETURN›. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

Si la date introduite est incorrecte (jour inconnu), elle est à nouveau demandée; sinon le programme passera à l'affichage du jour (2.6.3.2).

### 2.6.3.2. Affichage du jour.

Les activités de l'abonné durant la journée sont affichées sous la forme décrite page 1.5.

Pour retourner à la sélection d'un intervalle (2.6.2), l'utilisateur appuie sur la touche ‹RETURN›.

## 2.6.4. Vue d'une semaine.

### 2.6.4.1. Introduction du premier jour de la semaine.

La date du premier jour de la semaine dont l'utilisateur veut voir les moments d'occupation de l'abonnè est demandèe... L'utilisateur l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche ‹RETURN›. Les formes simplifièes suivantes sont aussi acceptèes : J/MM/AA, JJ/M/AA et J/M/AA.

Si la date introduite est incorrecte (jour inconnu), elle est à nouveau demandèe; sinon, le programme passera à l'affichage de la semaine (2.6.4.2).

### 2.6.4.2. Affichage de la semaine.

Un tableau contenant les zones de temps durant lesquelles l'abonnè est occupè est affichè sous la forme dècrite page 1.6.

Si l'utilisateur dèsire visualiser plus en dètail un jour de cette semaine, il introduit par le clavier ce jour et appuie ensuite sur la touche ‹RETURN›, et le programme passera à l'affichage de ce jour (2.6.4.3); sinon il introduit par le clavier '99' et appuie ensuite sur la touche ‹RETURN›, et le programme retournera à la sèlection d'un intervalle (2.6.2).

Si par mègarde, l'utilisateur introduit plus de 2 caractères, seuls les 2 premiers sont pris en considèration.

Si le nombre introduit n'est ni un jour de la semaine, ni '99', un nouveau choix est demandè jusqu'à l'introduction d'un nombre correct.

### 2.6.4.3. Affichage d'un jour de la semaine.

Les activitès de l'abonnè durant la journèe sont affichèes sous la forme dècrite page 1.5.

Pour retourner à l'affichage de la semaine (2.6.4.2), l'utilisateur appuie sur la touche ‹RETURN›.

## 2.6.5. Vue d'un mois.

### 2.6.5.1. Introduction d'un jour du mois.

La date d'un jour du mois durant lequel l'utilisateur veut voir les moments d'occupation de l'abonnè est demandèe... L'utilisateur l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche ‹RETURN›. Les formes simplifièes suivantes sont aussi acceptèes : J/MM/AA, JJ/M/AA et J/M/AA.

Si la date introduite est incorrecte (jour inconnu), elle est à nouveau demandèe; sinon, le programme passera à l'affichage du mois (2.6.5.2).

### 2.6.5.2. Affichage du mois.

Un tableau contenant les jours durant lesquels l'abonnè est occupè est affichè sous la forme dècrite page 1.6.

Si l'utilisateur dèsire visualiser plus en dètail un jour de ce mois, il introduit par le clavier ce jour et appuie ensuite sur la touche ‹RETURN›, et le programme passera à l'affichage de ce jour (2.6.5.3); sinon il introduit par le clavier '99' et appuie ensuite sur la touche ‹RETURN›, et le programme retournera à la sèlection d'un intervalle (2.6.2).

Si par mègarde, l'utilisateur introduit plus de 2 caractères, seuls les 2 premiers sont pris en considèration.

Si le nombre introduit n'est ni un jour du mois, ni '99', un nouveau choix est demandè jusqu'à l'introduction d'un nombre correct.

### 2.6.5.3. Affichage d'un jour du mois.

Les activitès de l'abonnè durant la journèe sont affichèes sous la forme dècrite page 1.5.

Pour retourner à l'affichage du mois (2.6.5.2), l'utilisateur appuie sur la touche ‹RETURN›.

### 2.6.6. Vue d'un intervalle quelconque.

#### 2.6.6.1. Introduction du début et de fin de l'intervalle.

Les dates de début et de fin de l'intervalle sont demandées... L'utilisateur les introduit par le clavier sous la forme JJ/MM/AA, chacune d'entre elles étant terminée en appuyant sur la touche <RETURN>. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

Si l'une de ces dates est incorrecte (jour inconnu), elle est à nouveau demandée; sinon le programme passera à l'affichage des jours demandés (2.6.6.2).

#### 2.6.6.2. Affichage des jours.

Les activités de l'abonné sont affichées successivement pour chaque jour sous la forme décrite page 1.5.

A la fin de chaque jour, l'utilisateur appuie sur la touche <RETURN>.

A la fin de l'intervalle, le programme retournera à la sélection d'un intervalle (2.6.2).

## 2.7. Vue de l'agenda d'une ressource.

Un nom de ressource est demandé... L'utilisateur l'introduit par le clavier et appuie ensuite sur la touche <RETURN> (seuls les 8 premiers caractères du nom sont pris en considération).

Le nom de la ressource est ensuite analysé par le programme.

Si le nom de la ressource est reconnu par le programme, l'utilisateur aura la possibilité de sélectionner un type d'activités (2.7.1).

Si le nom de la ressource n'est pas reconnu par le programme, un autre nom de ressource est demandé.

A la troisième erreur, le programme retournera au choix d'un traitement (2.5).

### 2.7.1. Sélection d'un type.

Le programme demande si l'utilisateur désire sélectionner un type d'activités... Il répond par 'OUI' ou 'NON', et appuie ensuite sur la touche <RETURN> (seul le premier caractère est nécessaire).

S'il répond 'OUI', il devra introduire par le clavier ce type et appuyer ensuite sur la touche <RETURN> (seuls les 8 premiers caractères du type seront pris en considération).

Exemple : l'utilisateur désire sélectionner un type d'activités et introduit le type 'COURS', cela signifie que, par la suite, l'utilisateur ne visualisera dans l'agenda que des activités appartenant au type "COURS".

De toute façon, le programme passera à la sélection d'un intervalle (2.6.2).

### 2.7.2. Sélection d'un intervalle.

Le programme propose à l'utilisateur une des possibilités suivantes :

1. vue d'un jour,
2. vue d'une semaine,
3. vue d'un mois,
4. vue d'un intervalle quelconque,
9. fin de visualisation.

Un choix est demandé... L'utilisateur introduit normalement par le clavier un des chiffres suivants : '1', '2', '3', '4' ou '9', et appuie ensuite sur la touche <RETURN>.

Selon que l'utilisateur introduit '1', '2', '3' ou '4', le programme passera respectivement à la vue d'un jour (2.7.3), à la vue d'une semaine (2.7.4), à la vue d'un mois (2.7.5) ou à la vue d'un intervalle quelconque (2.6.6).

S'il introduit '9', le programme retournera au choix d'un traitement (2.5).

S'il introduit n'importe quel autre caractère, un nouveau choix est demandé jusqu'à l'introduction d'un chiffre correct.

Si par mégarde, l'utilisateur introduit plusieurs caractères, seul le premier est pris en considération.

## 2.7.3. Vue d'un jour.

### 2.7.3.1. Introduction du jour.

La date du jour dont l'utilisateur veut voir les activités est demandée... L'utilisateur l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche <RETURN>. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

Si la date introduite est incorrecte (jour inconnu), elle est à nouveau demandée; sinon le programme passera à l'affichage du jour (2.7.3.2).

### 2.7.3.2. Affichage du jour.

Les activités consommant la ressource durant la journée sont affichées sous la forme décrite page 1.5.

Pour retourner à la sélection d'un intervalle (2.7.2), l'utilisateur appuie sur la touche <RETURN>.

## 2.7.4. Vue d'une semaine.

### 2.7.4.1. Introduction du premier jour de la semaine.

La date du premier jour de la semaine dont l'utilisateur veut voir les moments d'occupation de la ressource est demandée... L'utilisateur l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche <RETURN>. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

Si la date introduite est incorrecte (jour inconnu), elle est à nouveau demandée; sinon, le programme passera à l'affichage de la semaine (2.7.4.2).

### 2.7.4.2. Affichage de la semaine.

Un tableau contenant les zones de temps durant lesquelles la ressource est consommée est affiché sous la forme décrite page 1.6.

Si l'utilisateur désire visualiser plus en détail un jour de cette semaine, il introduit par le clavier ce jour et appuie ensuite sur la touche <RETURN>, et le programme passera à l'affichage de ce jour (2.7.4.3); sinon il introduit par le clavier '99' et appuie ensuite sur la touche <RETURN>, et le programme retournera à la sélection d'un intervalle (2.7.2).

Si par mégarde, l'utilisateur introduit plus de 2 caractères, seuls les 2 premiers sont pris en compte.

Si le nombre introduit n'est ni un jour de la semaine, ni '99', un nouveau choix est demandé jusqu'à l'introduction d'un nombre correct.

2.7.4.3. Affichage d'un jour de la semaine.

Les activités consommant la ressource durant la journée sont affichées sous la forme décrite page 1.5.

Pour retourner à l'affichage de la semaine (2.7.4.2), l'utilisateur appuie sur la touche <RETURN>.

2.7.5. Vue d'un mois.

2.7.5.1. Introduction d'un jour du mois.

La date d'un jour du mois durant lequel l'utilisateur veut voir les moments d'occupation de la ressource est demandée... L'utilisateur l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche <RETURN>. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

Si la date introduite est incorrecte (jour inconnu), elle est à nouveau demandée; sinon, le programme passera à l'affichage du mois (2.7.5.2).

2.7.5.2. Affichage du mois.

Un tableau contenant les jours durant lesquels la ressource est consommée est affiché sous la forme décrite page 1.6.

Si l'utilisateur désire visualiser plus en détail un jour de ce mois, il introduit par le clavier ce jour et appuie ensuite sur la touche <RETURN>, et le programme passera à l'affichage de ce jour (2.7.5.3); sinon il introduit par le clavier '99' et appuie ensuite sur la touche <RETURN>, et le programme retournera à la sélection d'un intervalle (2.7.2).

Si par mégarde, l'utilisateur introduit plus de 2 caractères, seuls les 2 premiers sont pris en compte.

Si le nombre introduit n'est ni un jour du mois, ni '99', un nouveau choix est demandé jusqu'à l'introduction d'un nombre correct.

2.7.5.3. Affichage d'un jour du mois.

Les activités consommant la ressource durant la journée sont affichées sous la forme décrite page 1.5.

Pour retourner à l'affichage du mois (2.7.5.2), l'utilisateur appuie sur la touche <RETURN>.

2.7.6. Vue d'un intervalle quelconque.

2.7.6.1. Introduction du début et de fin de l'intervalle.

Les dates de début et de fin de l'intervalle sont demandées... L'utilisateur les introduit par le clavier sous la forme JJ/MM/AA, chacune d'entre elles étant terminée en appuyant sur la touche <RETURN>. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

Si l'une de celles-ci est incorrecte (jour inconnu), elle est à nouveau demandée; sinon le programme passera à l'affichage des jours demandés (2.7.6.2).

2.7.6.2. Affichage des jours.

Les activités consommant la ressource sont affichées successivement pour chaque jour sous la forme décrite page 1.5.

A la fin de chaque jour, l'utilisateur appuie sur la touche <RETURN>.

A la fin de l'intervalle, le programme retournera à la sélection d'un intervalle (2.7.2).

## 2.8. Ajout d'une activité ou d'une série d'activités.

L'ajout d'une activité ou d'une série d'activités (c'est-à-dire de plusieurs activités séparées par un nombre de jours constant) dans un ou plusieurs agendas se fait en répondant aux demandes suivantes :

1. La date.

   La date de l'activité est demandée... L'utilisateur l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche ‹RETURN›. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

   Si la date introduite est incorrecte (jour inconnu) ou est inférieure à la date du jour (date introduite en 2.3), elle est à nouveau demandée.

   S'il s'agit de l'ajout d'une série d'activités, la date introduite correspond à la première activité de la série.

2. L'heure.

   L'heure de début de l'activité est demandée... L'utilisateur l'introduit par le clavier sous la forme HH.MM et appuie ensuite sur la touche ‹RETURN›. Les formes simplifiées suivantes sont aussi acceptées : H.MM, HH.M, H.M, HH et H.

   Si l'heure introduite est incorrecte, elle est à nouveau demandée.

3. La durée.

   La durée estimée de l'activité est demandée... L'utilisateur l'introduit par le clavier sous la forme HH.MM et appuie ensuite sur la touche ‹RETURN›. Les formes simplifiées suivantes sont aussi acceptées : H.MM, HH.N, H.M, HH et H.

   Si la durée introduite est incorrecte, elle est à nouveau demandée.

4. Le type.

   Le type de l'activité est demandé... Si l'utilisateur désire le spécifier, il l'introduit par le clavier et appuie ensuite sur la touche ‹RETURN› (seuls les 8 premiers caractères du type sont pris en considération); sinon, il appuie directement sur la touche ‹RETURN›.

5. La description.

   La description de l'activité est demandée... L'utilisateur l'introduit par le clavier au moyen de 2 lignes de 64 caractères maximum, chaque ligne étant terminée en appuyant sur la touche ‹RETURN›.

6. L'état de confidentialité.

L'état de confidentialité de l'activité est demandé... L'utilisateur répond par 'OUI' ou 'NON', et appuie ensuite sur la touche ‹RETURN› (seul le premier caractère est nécessaire).

S'il répond 'OUI', cela signifie que la description de l'activité est confidentielle.

S'il répond 'NON', cela signifie que la description de l'activité n'est pas confidentielle.

S'il répond n'importe quoi d'autre, la question est à nouveau posée jusqu'à l'introduction d'une réponse correcte.

7. Le nom du responsable.

Le nom du responsable de l'activité est demandé... S'il existe, l'utilisateur l'introduit par le clavier et appuie ensuite sur la touche ‹RETURN› (seuls les 8 premiers caractères du nom sont pris en considération); sinon, il appuie directement sur la touche ‹RETURN›.

Si le nom introduit n'est pas le nom d'un abonné, un autre nom de responsable est demandé.

8. La liste des personnes concernées.

Une liste de personnes concernées par l'activité est demandée... L'utilisateur introduit par le clavier le nom de ces personnes, chacun d'entre eux étant terminé en appuyant sur la touche ‹RETURN› (seuls les 8 premiers caractères de chaque nom sont pris en considération).

Si un des noms introduits n'est pas le nom d'un abonné, il n'est pas pris en considération.

Pour terminer l'introduction de la liste, l'utilisateur appuie directement sur la touche ‹RETURN›.

9. La liste res ressources consommées.

Une liste de ressources consommées par l'activité est demandée... L'utilisateur introduit par le clavier le nom de ces ressources, chacun d'entre eux étant terminé en appuyant sur la touche ‹RETURN› (seuls les 8 premiers caractères de chaque nom sont pris en considération).

Si un des noms introduits n'est pas le nom d'une ressource connue du système, il n'est pas pris en considération.

Pour terminer l'introduction de la liste, l'utilisateur appuie directement sur la touche ‹RETURN›.

10. L'ajout d'une activité ou d'une série d'activités.

La question de savoir s'il s'agit de l'ajout d'une activité ou d'une série d'activités est posée... L'utilisateur répond par 'OUI' ou 'NON', et appuie ensuite sur la touche <RETURN> (seul le premier caractère est nécessaire).

S'il répond 'OUI', cela signifie qu'il s'agit d'une série d'activités. Le nombre d'activités et le nombre de jours séparant 2 activités sont alors demandés... L'utilisateur les introduit par le clavier, chacun d'entre eux étant terminé en appuyant sur la touche <RETURN>.

S'il répond 'NON', cela signifie qu'il s'agit d'une seule activité.

S'il répond n'importe quoi d'autre, la question est reposée jusqu'à introduction d'une réponse correcte.

De toute façon, au terme de cet ajout, le programme retournera au choix d'un traitement (2.5).

REMARQUE : l'activité est ajoutée dans l'agenda des abonnés concernés sans tenir compte des autres activités déjà présentes dans leur agenda, les abonnés auront à charge de vérifier s'ils sont libres durant toute la durée de l'activité ajoutée.

## 2.9. Consultation et/ou mise à jour d'une activité.

### 2.9.1. Introduction de la date et de l'heure.

La date de l'activité recherchée est demandée... L'utilisateur l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche <RETURN>. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

Si la date introduite est incorrecte (jour inconnu), elle est à nouveau demandée.

L'heure de début de l'activité recherchée est demandée... L'utilisateur l'introduit par le clavier sous la forme HH.MM et appuie ensuite sur la touche <RETURN>. Les formes simplifiées suivantes sont aussi acceptées : H.MM, H.M, HH et H.

Si l'heure introduite est incorrecte, elle est à nouveau demandée.

### 2.9.2. Recherche de l'activité désirée.

Le programme cherche une activité que l'utilisateur a lui-même enregistrée ou de laquelle il est responsable, et correspondant aux date et heure introduites.

S'il en trouve une, il l'affiche et demande s'il s'agit bien de l'activité souhaitée... L'utilisateur répond par 'OUI' ou 'NON', et appuie ensuite sur la touche <RETURN> (seul le premier caractère est nécessaire).

Si l'utilisateur répond 'OUI', cela signifie qu'il souhaite traiter l'activité affichée et le programme passera au traitement de l'activité (2.9.3).

S'il répond 'NON', cela signifie qu'il ne souhaite pas traiter l'activité affichée et le programme en cherchera une autre correspondant toujours aux date et heure introduites.

Si le programme ne trouve pas l'activité que l'utilisateur souhaite traiter, il le signale... L'utilisateur doit alors appuyer sur la touche <RETURN> pour retourner au choix d'un traitement (2.5).

### 2.9.3. Traitement de l'activité.

L'activité est à nouveau affichée et diverses possibilités sont proposées :

1. modification de la date,
2. modification de l'heure,
3. modification de la durée,
4. modification du type,
5. modification de la description,
6. modification de l'état de confidentialité,
7. modification du nom de responsable,
8. modification de la liste des personnes concernées,
9. modification de la liste des ressources consommées,

R. remplacement de l'activité,
A. ajout de l'activité,
S. suppression de l'activité.

Un choix est demandé... L'utilisateur introduit normalement un des caractères suivants : '1', '2', '3', '4', '5', '6', '7', '8', '9', 'R', 'A' ou 'S', et appuie ensuite sur la touche <RETURN>.

Selon qu'il introduit '1', '2', '3', '4', '5', '6', '7', '8', '9', 'R', 'A' ou 'S', le programme passera respectivement à la modification de la date (2.9.4), à la modification de l'heure (2.9.5), à la modification de la durée (2.9.6), à la modification du type (2.9.7), à la modification de la description (2.9.8), à la modification de l'état de confidentialité (2.9.9), à la modification du nom du responsable (2.9.10), à la modification de la liste des personnes concernées (2.9.11), à la modification de la liste des ressources consommées (2.9.12), au remplacement de l'activité (2.9.13), à l'ajout de l'activité (2.9.14) ou à la suppression de l'activité (2.9.15).

S'il introduit n'importe quel autre caractère, un nouveau choix est demandé jusqu'à l'introduction d'un caractère correct.

Si par mégarde, l'utilisateur a introduit plusieurs caractères, seul le premier est pris en considération.

2.9.4. Modification de la date.

La nouvelle date de l'activité est demandée... L'utilisateur l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche <RETURN>. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

Si la date introduite est incorrecte (jour inconnu) ou est inférieure à la date du jour (date introduite en 2.3), elle est à nouveau demandée.

Au terme de cette modification, le programme retournera au traitement sur l'activité (2.9.3).

2.9.5. Modification de l'heure.

La nouvelle heure de début de l'activité est demandée... L'utilisateur l'introduit par le clavier sous la forme HH.MM et appuie ensuite sur la touche <RETURN>. Les formes simplifiées suivantes sont aussi acceptées : H.MM, HH.M, H.M, HH et H.

Si l'heure introduite est incorrecte, elle est à nouveau demandée.

Au terme de cette modification, le programme retournera au traitement sur l'activité (2.9.3).

2.9.6. Modification de la durée.

La nouvelle durée estimée de l'activité est demandée... L'utilisateur l'introduit par le clavier sous la forme HH.MM et appuie ensuite sur la touche ‹RETURN›. Les formes simplifiées suivantes sont aussi acceptées : H.MM, HH.M, H.M, HH et H.

Si la durée introduite est incorrecte, elle est à nouveau demandée.

Au terme de cette modification, le programme retournera au traitement sur l'activité (2.9.3)

2.9.7. Modification du type.

Le nouveau type de l'activité est demandé... Si l'utilisateur désire le spécifier, il l'introduit par le clavier et appuie ensuite sur la touche ‹RETURN› (seuls les 8 premiers caractères du type sont pris en considération); sinon, il appuie directement sur la touche ‹RETURN›.

Au terme de cette modification, le programme retournera au traitement sur l'activité (2.9.3).

2.9.8. Modification de la description.

La nouvelle description de l'activité est demandée... L'utilisateur l'introduit par le clavier au moyen de 2 lignes de 64 caractères maximum, chaque ligne étant terminée en appuyant sur la touche ‹RETURN›.

Au terme de cette modification, le programme retournera au traitement sur l'activité (2.9.3).

2.9.9. Modification de l'état de confidentialité.

Le nouvel état de confidentialité de l'activité est demandé... L'utilisateur répond par 'OUI' ou 'NON', et appuie ensuite sur la touche ‹RETURN› (seul le premier caractère est nécessaire).

S'il répond 'OUI', cela signifie que la description de l'activité devient confidentielle.

S'il répond 'NON', cela signifie que la description de l'activité n'est plus confidentielle.

S'il répond n'importe quoi d'autre, la question est à nouveau posée jusqu'à l'introduction d'une réponse correcte.

Au terme de cette modification, le programme retournera au traitement sur l'activité (2.9.3).

2.9.10. Modification du responsable.

Le nouveau nom du responsable de l'activité est demandé... S'il existe, l'utilisateur l'introduit par le clavier et appuie ensuite sur la touche <RETURN> (seuls les 8 premiers caractères du nom sont pris en considération); sinon, il appuie directement sur la touche <RETURN>.

Si le nom introduit n'est pas le nom d'un abonné, un autre nom de responsable est demandé.

Au terme de cette modification, le programme retournera au traitement sur l'activité (2.9.3).

2.9.11. Modification de la liste des personnes concernées.

Une nouvelle liste de personnes concernées par l'activité est demandée... L'utilisateur introduit par le clavier le nom de ces personnes, chacun d'entre eux étant terminé en appuyant sur la touche <RETURN> (seuls les 8 premiers caractères de chaque nom sont pris en considération).

Si un des noms introduits n'est pas le nom d'un abonné, il n'est pas pris en considération.

Pour terminer l'introduction de la liste, l'utilisateur appuie directement sur la touche <RETURN>.

Au terme de cette modification, le programme retournera au traitement sur l'activité (2.9.3).

2.9.12. Modification de la liste des ressources consommées.

Une nouvelle liste de ressources consommées par l'activité est demandée... L'utilisateur introduit par le clavier le nom de ces ressources, chacun d'entre eux étant terminé en appuyant sur la touche <RETURN> (seuls les 8 premiers caractères de chaque nom sont pris en considération).

Si un des noms introduits n'est pas le nom d'une ressource connue du système, il n'est pas pris en considération.

Pour terminer l'introduction de la liste, l'utilisateur appuie directement sur la touche <RETURN>.

Au terme de cette modification, le programme retournera au traitement sur l'activité (2.9.3).

2.9.13. Remplacement de l'activitè.

L'activitè initiale est remplacée dans le ou les agendas concernès par l'activitè courante.

Si une des ressources consommées n'est plus disponible pendant toute la durée de l'activitè, une nouvelle liste de ressources consommées est demandée... L'utilisateur introduit par le clavier le nom de ces ressources, chacun d'entre eux étant terminè en appuyant sur la touche <RETURN> (seuls les 8 premiers caractères de chaque nom sont pris en considération).

Si un des noms introduits n'est pas le nom d'une ressource connue du système, il n'est pas pris en considération.

Pour terminer l'introduction de la nouvelle liste, l'utilisateur appuie directement sur le touche <RETURN>.

Au terme de ce remplacement, le programme retournera au choix d'un traitement (2.5).

REMARQUE : l'activitè est modifiée dans l'agenda des abonnès concernès sans tenir compte des autres activitès dèjà présentes dans leur agenda, les abonnès auront à charge de vèrifier s'ils sont libres durant toute la durée de l'activitè modifiée.

2.9.14. Ajout de l'activitè.

L'activitè courante est ajoutée dans le ou les agendas concernès.

Si une des ressources consommées n'est plus disponible pendant toute la durée de l'activitè, une nouvelle liste de ressources consommées est demandée... L'utilisateur introduit par le clavier le nom de ces ressources, chacun d'entre eux étant terminè en appuyant sur la touche <RETURN> (seuls les 8 premiers caractères de chaque nom sont pris en considération).

Si un des noms introduits n'est pas le nom d'une ressource connue du sytème, il n'est pas pris en considération.

Pour terminer l'introduction de la liste, l'utilisateur appuie directement sur la touche <RETURN>.

Au terme de cet ajout, le programme retournera au choix d'un traitement (2.5).

REMARQUE : l'activité est ajoutée dans l'agenda des abonnés concernés sans tenir compte des autres activités déjà présentes dans leur agenda, les abonnés auront à charge de vérifier s'il sont libres durant toute la durée de l'activité ajoutée.

2.9.15. Suppression de l'activité.

L'activité initiale est supprimée du ou des agendas concernés.

Au terme de cette suppression, le programme retournera au choix d'un traitement (2.5).

2.10. Recherche de plages de temps libre.

La date du jour à partir duquel l'utilisateur veut visualiser les plages de temps libre est demandée... Il l'introduit par le clavier sous la forme JJ/MM/AA et appuie ensuite sur la touche ‹RETURN›. Les formes simplifiées suivantes sont aussi acceptées : J/MM/AA, JJ/M/AA et J/M/AA.

Une liste des abonnés et des ressources pour lesquels l'utilisateur veut visualiser les plages de temps libre communes est ensuite demandée... L'utilisateur introduit par le clavier le nom de ces personnes et de ces ressources, chacun d'entre eux étant terminé en appuyant sur la touche ‹RETURN› (seuls les 8 premiers caractères de chaque nom sont pris en considération).

Si un des noms de personne introduits n'est pas le nom d'un abonné, il n'est pas pris en considération.

Si un des noms de ressource introduits n'est pas le nom d'une ressource connue du système, il n'est pas pris en considération.

Pour terminer l'introduction de chacune de ces deux listes, l'utilisateur appuie directement sur la touche ‹RETURN›.

Les plages de temps libre communes aux abonnés et aux ressources connues du système introduites sont affichées sous la forme décrite page 1.8.

Pour retourner au choix d'un traitement (2.5), l'utilisateur appuie sur la touche ‹RETURN›.

## 3. Le programme "GESTION".

Ce programme regroupe les différents traitements "utilitaires" possibles relatifs aux abonnés et ressources.

### 3.1. Chargement du programme.

Une fois connecté au système PDP 11, le caractère '%' apparait à l'écran et l'utilisateur introduit par le clavier la commande 'EM1 GESTION' qu'il termine en appuyant sur la touche <RETURN>.

L'exécution du programme GESTION est ainsi lancée (3.2).

### 3.2. Identification.

Après un petit temps d'attente, un nom et un mot de passe sont demandés... L'utilisateur les introduit par le clavier, chacun d'entre eux étant terminé en appuyant sur la touche <RETURN> (seuls les huit premiers caractères du nom et du mot de passe sont pris en considération).

Si le nom et le mot de passe introduits sont reconnus par le programme, celui-ci proposera à l'utilisateur de choisir un des traitements qu'il peut effectuer (3.3); sinon un nom et un mot de passe sont à nouveau demandés.

A la troisième erreur, l'exécution du programme GESTION est arrêtée.

### 3.3. Choix d'un traitement.

Le programme propose à l'utilisateur un des traitements suivants :

1. liste des abonnés.
2. ajout d'un abonné.
3. modification du mot de passe d'un abonné.
4. modification de la liste des accès à l'agenda d'un abonné.
5. liste des ressources.
6. ajout d'une ressource.
7. modification du responsable d'un ressource.
9. fin de traitement.

Un choix est demandé... L'utilisateur introduit normalement par le clavier un des chiffres suivants : '1', '2', '3', '4', '5', '6', '7' ou '9', et appuie ensuite sur la touche <RETURN>.

Selon que l'utilisateur introduit '1', '2', '3', '4', '5', '6' ou '7', le programme passera respectivement à la liste des abonnés (3.4), à l'ajout d'un abonné (3.5), à la modification du mot de passe d'un abonné (3.6), à la modification de la liste des accès à l'agenda d'un abonné (3.7), à la liste des ressources (3.8), à l'ajout d'une ressource (3.9) ou à la modification du responsable d'une ressource (3.10).

S'il introduit '9', il met ainsi fin à l'exécution du programme GESTION.

S'il introduit n'importe quel autre caractère, un nouveau choix est demandé jusqu'à introduction d'un chiffre correct.

Si par mégarde, l'utilisateur introduit plusieurs caractères, seul le premier est pris en considération.

## 3.4. Liste des abonnés.

La liste des abonnés est affichée ainsi que l'ensemble des personnes ayant accès à leur agenda respectif sous la forme décrite page 1.11.

Pour visualiser la suite de la liste ou pour retourner au choix d'un traitement (3.3), l'utilisateur appuie sur la touche ‹RETURN›.

## 3.5. Ajout d'un abonné.

Cette fonction est réservée à l'administrateur du système.

Si l'utilisateur n'est pas l'administrateur du système, le programme retournera directement au choix d'un traitement (3.3).

Sinon, un nom d'abonné lui est demandé... Il l'introduit par le clavier et appuie ensuite sur la touche ‹RETURN› (seuls les 8 premiers caractères du nom sont pris en considération).

Le nom d'abonné est ensuite analysé par le programme.

Si le nom du nouvel abonné est déjà celui d'un abonné existant, un autre nom d'abonné est demandé. A la troisième erreur, le programme retournera au choix d'un traitement (3.3).

Sinon, un mot de passe est demandé... L'utilisateur l'introduit par le clavier et appuie ensuite sur la touche ‹RETURN› (seuls les 8 premiers caractères du mot de passe sont pris en considération).

Ensuite, une liste des personnes ayant accès à l'agenda du nouvel abonné est demandée... L'utilisateur introduit par le clavier le nom de ces personnes, chacun d'entre eux étant suivi de la touche <RETURN> (seuls les 8 premiers caractères de chaque nom sont pris en considération).

Si un des noms introduits n'est pas le nom d'un abonné, il n'est pas pris en considération.

Pour terminer l'introduction de la liste, l'utilisateur appuie directement sur la touche <RETURN>.

Au terme de cet ajout, le programme retournera aux choix d'un traitement (3.3).

## 3.6. Modification du mot de passe d'un abonné.

L'utilisateur a ici la possibilité de changer son mot de passe... Il introduit donc son nouveau mot de passe et appuie ensuite sur la touche <RETURN> (seuls les 8 premiers caractères du mot de passe sont pris en considération).

Au terme de cette modification, le programme retournera aux choix d'un traitement (3.3).

## 3.7. Modification de la liste des accès à l'agenda d'un abonné.

L'utilisateur a ici la possibilité de changer la liste des personnes ayant accès à son agenda... Il introduit donc par le clavier le nom de ces personnes, chacun d'entre eux suivi de la touche <RETURN> (seuls les 8 premiers caractères de chaque nom sont pris en considération).

Si un des noms introduits n'est pas le nom d'un abonné, il n'est pas pris en considération.

Pour terminer l'introduction de la liste, l'utilisateur appuie directement sur la touche <RETURN>.

Au terme de cette modification, le programme retournera au choix d'un traitement (3.3).

## 3.8. Liste des ressources.

La liste des ressource est affichée ainsi que le nom de leur responsable respectif sous la forme décrite page 1.11.

Pour visualiser la suite ce cette liste ou pour retourner au choix d'un traitement (3.3), l'utilisateur appuie sur la touche <RETURN>.

## 3.9. Ajout d'une ressource.

Cette fonction est réservée à l'administrateur du système.

Si l'utilisateur n'est pas l'administrateur du système, le programme retournera directement au choix d'un traitement (3.3).

Sinon, un nom de ressource est demandé... L'utilisateur l'introduit par le clavier et appuie ensuite sur la touche ‹RETURN› (seuls les 8 premiers caractères du nom sont pris en considération).

Si le nom introduit est déjà celui d'un ressource connue du système, un autre nom de ressource est demandé. A la troisième erreur, le programme retournera au choix d'un traitement (3.3).

Sinon, un nom de responsable est demandé... L'utilisateur l'introduit par le clavier et appuie ensuite sur la touche ‹RETURN› (seuls les 8 premiers caractères du nom sont pris en considération).

Si le nom introduit n'est pas le nom d'un abonné, un autre nom de responsable est demandé.

Au terme de cet ajout, le programme retournera au choix d'un traitement.

## 3.10. Modification du responsable d'une ressource.

Cette fonction est réservée à l'administrateur du système.

Si l'utilisateur n'est pas l'administrateur du système, le programme retournera directement au choix d'un traitement (3.3).

Sinon, un nom de ressource est demandé... L'utilisateur l'introduit par le clavier et appuie ensuite sur la touche ‹RETURN› (seuls les 8 premiers caractères sont pris en considération).

Si le nom introduit n'est pas le nom d'une ressource connue du système, un autre nom de ressource est demandé. A la troisième erreur, le programme retournera au choix d'un traitement (3.3).

Si le nom introduit est le nom d'une ressource connue du système, le nom de l'ancien responsable est affiché et un nouveau nom de responsable est demandé... L'utilisateur l'introduit par le clavier et appuie ensuite sur la touche ‹RETURN› (seuls les 8 premiers caractères du nom sont pris en considération).

Si le nom de responsable introduit n'est pas le nom d'un abonné, un autre nom de responsable est demandé.

Au terme de cette modification, le programme retournera au choix d'un traitement (3.3).

# Chapitre 5: CONCLUSIONS.

Le but essentiel de ce travail était de concevoir et de réaliser un gestionnaire d'agendas

La méthode de travail suivie se rapproche beaucoup de celle enseignée à l'Institut d'Informatique durant ces trois dernières années.

La première étape a consisté en une analyse des besoins fonctionnels.

Ensuite, une structure de données et une structure de traitements ont été développées en fonction des spécifications fonctionnelles tout en prenant en considération certaines contraintes matérielles. Et de là, découlent deux programmes écrits en langage "PASCAL" (le premier regroupant les traitements sur les agendas et le second les traitements relatifs aux abonnés et ressources); ceux-ci constituant le gestionnaire d'agendas.

Enfin, un manuel utilisateur a été rédigé pour ce gestionnaire d'agendas.

Il resterait maintenant à étudier le comportement de ces deux programmes, tels qu'ils fonctionnent actuellement auprès d'un ensemble d'utilisateurs. Il semble probable que, après une brève période de familiarisation, ces programmes soient utilisables de façon efficace par des personnes non expérimentées en informatique; tel était le but que l'on s'était assigné dès le départ.

# BIBLIOGRAPHIE.

[BLA82] J-P. de Blasis
La bureautique : outils et applications
Paris, Editions d'Organisation, 1982

[BOD80] F. Bodart
Dynamic problem specification language ISDOS project
Institut d'Informatique, novembre 1980

[BOD81] F. Bodart
Elèments de conception et d'analyse des sytèmes d'information
Institut d'Informatique, juillet 1981

[ENG79] G. H. Engel et autres
An office communication system
IBM Systems Journal 18 no. 1, 1979

[HAI80] J-L. Hainaut
Dérivation d'une première structure d'accès à partir d'un schèma conceptuel entité/association
Institut d'Informatique, décembre 1980

[HAU82] F. d'Haucourt-Carette
Conception d'un système d'agenda électronique
Congrès FAIB, 1982

[LAM81] A. van Lamsweerde
Quelques principes pour le développement d'une architecture logicielle
Institut d'Informatique, juillet 1981

[ROS80] R. M. Rosembaum
Office technology systems : conceptual functional requirements from a user's perspective
Standard Oil cy (Office Technology Department), juillet 1980

# Annexes:

On trouve ici le texte des différents programmes constituant le gestionnaire d'agendas :

1. Le programme AGENDA dans lequel sont regroupés les différents traitements possibles sur les agendas (pages A.2 à A.43).

2. Le programme GESTION dans lequel sont regroupés les différents traitements "utilitaires" possibles relatifs aux abonnés et ressources (pages A.44 à A.55).

3. Le fichier ACCES.CONST dans lequel se trouve la définition des constantes utilisées par les procédures d'accès (page A.56).

4. Le fichier ACCES.TYPE dans lequel se trouve la définition des types utilisés par les procédures d'accès (pages A.57).

5. Le fichier ACCES.PROCEDURE dans lequel se trouve l'ensemble des procédures d'accès aux fichiers (pages A.58 à A.67).

```
#
{$c+}

program agenda (input, output);

const long_mat = 90;

#include "acces.const"

type string8 = packed array [1..8] of char;
     string128 = packed array [1..128] of char;
     liste_string = array [1..10] of string8;
     liste_char = array [1..10] of char;
     liste_4 = array [1..4] of string8;
     mat_real = array [1..long_mat] of real;
     mat_string = array [1..long_mat] of string128;
     tbl_semaine = array [1..7] of packed array [0..95] of char;
     tbl_mois = array [1..31] of char;
     pers = record
               nom_pers : string8;
               mot_de_passe : string8;
               nom_acces : liste_string;
            end;
     ress = record
               nom_ress : string8;
               nom_resp : string8;
            end;
     activ = record
                date_activ : real;
                heure : real;
                duree : real;
                sorte : string8;
                descr : string128;
                confid : char;
                nom_enreg : string8;
                nom_resp : string8;
                nom_pers_cons : liste_string;
                code_reponse : liste_char;
                nom_ress_cons : liste_4;
             end;
     apur = record
               date_apur : real;
            end;

#include "acces.type"

var exact : boolean;
    n : string8;
    choix : char;
    date_du_jour : real;
    debut : array [1..28] of integer;
    maxmois : array [1..12] of integer;
    jour : array [1..7] of string8;
    mois : array [1..12] of packed array [1..9] of char;
    mat_date, mat_heure, mat_duree : mat_real;
    mat_descr : mat_string;
```

```
#include "acces.procedure"

{*****************************************************************************}

procedure lire_string (var n : string8);

   var i : integer;
       c : char;

   begin
      n := '        ';
      i := 0;
      repeat
         read (c);
         if (i > 0) and (i < 9) then n [i] := c;
         i := i + 1;
      until eoln;
   end;

{*****************************************************************************}

procedure erreur01;

   begin
      writeln ('nom et/ou mot de passe incorrect !!!');
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure erreur03;

   begin
      writeln ('cette personne n''est pas connue du systeme !!!');
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure erreur04;

   begin
      writeln ('cette ressource n''est pas connue du systeme !!!');
      writeln;
      writeln;
   end;
```

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

procedure erreur07;

   begin
      write ('cette ressource n''est pas disponible a ce moment precis !!!');
      writeln;
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure erreur08 (nom : string8);

   begin
      writeln ('la ressource "', nom, '" n''est plus disponible !!!');
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure erreur09;

   begin
      writeln ('cette date est deja passee !!!');
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure erreur10;

   begin
      write ('cette date est inferieure a la date du debut ');
      writeln ('d''intervalle !!!');
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure erreur11;

   begin
      writeln ('cette date est inferieure a une date deja rentree !!!');
```

```
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure erreur12;

   begin
      writeln ('vous n''avez pas acces a l''agenda de cet abonne !!!');
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure ident_personne (var n : string8; var exact : boolean);

   var fin : boolean;
       i : integer;
       p : pers;
       m : string8;
       num : integer;

   begin
      for i := 1 to 10 do writeln;
      writeln ('                    ************************************');
      writeln ('                    *                                  *');
      writeln ('                    *                                  *');
      writeln ('                    *     G E S T I O N N A I R E      *');
      writeln ('                    *                                  *');
      writeln ('                    *        D ' ' A G E N D A S       *');
      writeln ('                    *                                  *');
      writeln ('                    *                                  *');
      writeln ('                    ************************************');
      for i := 1 to 6 do writeln;
      i := 0;
      repeat
         i := i + 1;
         write ('nom : ');
         lire_string (n);
         writeln;
         write ('mot de passe : ');
         lire_string (m);
         writeln;
         exact := false;
         ouvrir_personne (num);
         lire_personne (num, p, fin);
         while (not fin) and (not exact) do
            begin
               if (p.nom_pers = n) and (p.mot_de_passe = m)
                  then exact := true
                  else lire_personne (num, p, fin);
```

```
            end;
         fermer_personne (num);
         if not exact then erreur01;
      until (exact) or (i = 3);
   end;

{*****************************************************************************}

procedure init_date;

   begin
      debut [1] := 5;                    debut [15] := 1;
      debut [2] := 6;                    debut [16] := 2;
      debut [3] := 0;                    debut [17] := 4;
      debut [4] := 1;                    debut [18] := 5;
      debut [5] := 3;                    debut [19] := 6;
      debut [6] := 4;                    debut [20] := 0;
      debut [7] := 5;                    debut [21] := 2;
      debut [8] := 6;                    debut [22] := 3;
      debut [9] := 1;                    debut [23] := 4;
      debut [10] := 2;                   debut [24] := 5;
      debut [11] := 3;                   debut [25] := 0;
      debut [12] := 4;                   debut [26] := 1;
      debut [13] := 6;                   debut [27] := 2;
      debut [14] := 0;                   debut [28] := 3;
      maxmois [1] := 31;                 mois [1] := 'janvier  ';
      maxmois [2] := 28;                 mois [2] := 'fevrier  ';
      maxmois [3] := 31;                 mois [3] := 'mars     ';
      maxmois [4] := 30;                 mois [4] := 'avril    ';
      maxmois [5] := 31;                 mois [5] := 'mai      ';
      maxmois [6] := 30;                 mois [6] := 'juin     ';
      maxmois [7] := 31;                 mois [7] := 'juillet  ';
      maxmois [8] := 31;                 mois [8] := 'aout     ';
      maxmois [9] := 30;                 mois [9] := 'septembre';
      maxmois [10] := 31;                mois [10] := 'octobre  ';
      maxmois [11] := 30;                mois [11] := 'novembre ';
      maxmois [12] := 31;                mois [12] := 'decembre ';
      jour [1] := 'lundi    ';           jour [5] := 'vendredi';
      jour [2] := 'mardi    ';           jour [6] := 'samedi  ';
      jour [3] := 'mercredi';            jour [7] := 'dimanche';
      jour [4] := 'jeudi    ';
   end;

{*****************************************************************************}

procedure decomp_date (date : real;
                       var jo : integer; var mo : integer; var an : integer);

   begin
      an := trunc (date / 10000);
      date := date - an * 10000.0;
      an := an + 1900;
      mo := trunc (date / 100);
```

```
      date := date - mo * 100;
      jo := trunc (date);
      if an mod 4 = 0
         then maxmois [2] := 29
         else maxmois [2] := 28;
   end;

{*****************************************************************************}

procedure valid_date (var date : real);

   var jo, mo, an : integer;

   begin
      decomp_date (date, jo, mo, an);
      if jo > maxmois [mo] then
         begin
            date := date + 100 - maxmois [mo];
            if mo > 11 then date := date + 8800;
         end;
   end;

{*****************************************************************************}

procedure trouve_jour (jo : integer; mo : integer; an : integer;
                       var jj : integer);

   var i : integer;

   begin
      jj := debut [(an - 1601) mod 28 + 1] + jo - 1;
      i := 1;
      while i < mo do
         begin
            jj := jj + maxmois [i];
            i := i + 1;
         end;
      jj := (jj - 1) mod 7 + 1;
   end;

{*****************************************************************************}

procedure decomp_heure (heure : real; var he : integer; var min : integer);

   begin
      he := trunc (heure);
      min := trunc ((heure - he) * 100);
   end;

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

```
procedure r_reponse (var reponse : char);

   var i : integer;
       c : char;

   begin
      reponse := ' ';
      i := 0;
      repeat
         read (c);
         if i = 1 then reponse := c;
         i := i + 1;
      until eoln;
      writeln;
   end;


{*****************************************************************************}


procedure test_date (date : real; var exact : boolean);

   var jo, mo, an : integer;

   begin
      decomp_date (date, jo, mo, an);
      if mo <= 12
         then
            begin
               if (jo <= maxmois [mo]) and (jo > 0)
                  then exact := true
                  else exact := false;
            end
         else exact := false;
   end;


{*****************************************************************************}


procedure r_date (var date : real; var exact : boolean);

   var d : string8;
       i : integer;
       nbre : array [1..8] of integer;

   begin
      exact := false;
      lire_string (d);
      writeln;
      for i := 1 to 8 do nbre [i] := ord (d [i]) - 48;
      if (nbre [1] >= 0) and (nbre [1] <= 9) then
         begin
            if (nbre [2] >= 0) and (nbre [2] <= 9)
               then
                  begin
                     date := nbre [1] * 10 + nbre [2];
```

```
            if (nbre [4] >= 0) and (nbre [4] <= 9) then
               begin
                  if (nbre [5] >= 0) and (nbre [5] <= 9)
                     then
                        begin
                           date := date + nbre [4] * 1000
                                        + nbre [5] * 100;
                           if (nbre [7] >= 0) and (nbre [7] <= 9) and
                             (nbre [8] >= 0) and (nbre [8] <= 9) then
                              begin
                                 date := date + nbre [7] * 100000.0
                                              + nbre [8] * 10000.0;
                                 test_date (date, exact);
                              end;
                        end
                     else
                        begin
                           date := date + nbre [4] * 100;
                           if (nbre [6] >= 0) and (nbre [6] <= 9) and
                             (nbre [7] >= 0) and (nbre [7] <= 9) then
                              begin
                                 date := date + nbre [6] * 100000.0
                                              + nbre [7] * 10000.0;
                                 test_date (date, exact);
                              end;
                        end;
               end;
         end
      else
         begin
            date := nbre [1];
            if (nbre [3] >= 0) and (nbre [3] <= 9) then
               begin
                  if (nbre [4] >= 0) and (nbre [4] <= 9)
                     then
                        begin
                           date := date + nbre [3] * 1000
                                        + nbre [4] * 100;
                           if (nbre [6] >= 0) and (nbre [6] <= 9) and
                             (nbre [7] >= 0) and (nbre [7] <= 9) then
                              begin
                                 date := date + nbre [6] * 100000.0
                                              + nbre [7] * 10000.0;
                                 test_date (date, exact);
                              end;
                        end
                     else
                        begin
                           date := date + nbre [3] * 100;
                           if (nbre [5] >= 0) and (nbre [5] <= 9) and
                             (nbre [6] >= 0) and (nbre [6] <= 9) then
                              begin
                                 date := date + nbre [5] * 100000.0
                                              + nbre [6] * 10000.0;
                                 test_date (date, exact);
                              end;
                        end;
```

```
                        end;
                    end;
            end;
    end;

{*****************************************************************************}

procedure test_heure (heure : real; var exact : boolean);

    begin
        if (heure >= 0) and (heure <= 24)
            then
                begin
                    if heure - trunc (heure) < 0.60
                        then exact := true
                        else exact := false;
                end
            else exact := false;
    end;

{*****************************************************************************}

procedure r_heure (var heure : real; var exact : boolean);

    var h : string8;
        nbre : array [1..5] of integer;
        i : integer;

    begin
        exact := false;
        lire_string (h);
        writeln;
        for i := 1 to 5 do nbre [i] := ord (h [i]) - 48;
        if (nbre [1] >= 0) and (nbre [1] <= 9) then
            begin
                if (nbre [2] >= 0) and (nbre [2] <= 9)
                    then
                        begin
                            heure := nbre [1] * 10 + nbre [2];
                            if (nbre [4] >= 0) and (nbre [4] <= 9)
                                then
                                    begin
                                        if (nbre [5] >= 0) and (nbre [5] <= 9)
                                            then
                                                begin
                                                    heure := heure + nbre [4] / 10
                                                                   + nbre [5] / 100;
                                                    test_heure (heure, exact);
                                                end
                                            else
                                                begin
                                                    heure := heure + nbre [4] / 100;
                                                    test_heure (heure, exact);
```

```
                                    end;
                              end
                           else test_heure (heure, exact);
                     end
                  else
                     begin
                        heure := nbre [1];
                        if (nbre [3] >= 0) and (nbre [3] <= 9)
                           then
                              begin
                                 if (nbre [4] >= 0) and (nbre [4] <= 9)
                                    then
                                       begin
                                          heure := heure + nbre [3] / 10
                                                        + nbre [4] / 100;
                                          test_heure (heure, exact);
                                       end
                                    else
                                       begin
                                          heure := heure + nbre [3] / 100;
                                          test_heure (heure, exact);
                                       end;
                              end
                           else test_heure (heure, exact);
                     end;
            end;
      end;

{*****************************************************************************}

procedure r_nombre (var nombre : integer);

   var n : string8;

   begin
      nombre := 0;
      lire_string (n);
      writeln;
      if (n [1] >= '0') and (n [1] <= '9') then
         begin
            if (n [2] >= '0') and (n [2] <= '9')
               then nombre := (ord (n [1]) - 48) * 10 + ord (n [2]) - 48
               else nombre := ord (n [1]) - 48;
         end;
   end;

{*****************************************************************************}

procedure r_description (var description : string128);

   var i, j, k : integer;
       c : char;
       fin : boolean;
```

```
   begin
      for i := 1 to 128 do description [i] := ' ';
      j := 0;
      repeat
         k := j * 64;
         i := 0;
         fin := true;
         repeat
            read (c);
            if (i > 0) and (i < 65) then
               begin
                  description [k + i] := c;
                  if c <> ' ' then fin := false;
               end;
            i := i + 1;
         until eoln;
         write ('                ');
         j := j + 1;
      until (j = 2) or (fin);
      writeln;
   end;

{*****************************************************************************}

procedure r_personne (var n : string8; var exact : boolean);

   var fin : boolean;
       p : pers;
       num : integer;

   begin
      lire_string (n);
      writeln;
      exact := false;
      ouvrir_personne (num);
      lire_personne (num, p, fin);
      while (not fin) and (not exact) do
         begin
            if p.nom_pers = n
               then exact := true
               else lire_personne(num, p, fin);
         end;
      fermer_personne (num);
   end;

{*****************************************************************************}

procedure r_ressource (var n : string8; var exact : boolean);

   var fin : boolean;
       r : ress;
       num : integer;
```

```
   begin
      lire_string (n);
      writeln;
      exact := false;
      ouvrir_ressource (num);
      lire_ressource (num, r, fin);
      while (not fin) and (not exact) do
         begin
            if r.nom_ress = n
               then exact := true
               else lire_ressource (num, r, fin);
         end;
      fermer_ressource (num);
   end;


{*************************************************************************}


procedure r_lst_personne (var a : liste_string);

   var i : integer;
       n : string8;
       exact : boolean;

   begin
      i := 0;
      repeat
         write ('nom : ');
         r_personne (n, exact);
         if exact
            then
               begin
                  i := i + 1;
                  a [i] := n;
               end
            else if n <> '        ' then erreur03;

      until (n = '        ') or (i = 10);
      while i < 10 do
         begin
            i := i + 1;
            a [i] := '        ';
         end;
   end;


{*************************************************************************}


procedure test_disponible (date : real; heure : real; duree : real;
                           n : string8; var occupe : boolean);

   var num, j : integer;
       a : activ;
       fin : boolean;
```

```
   begin
      occupe := false;
      ouvrir_activite (num);
      lire_activite (num, a, fin);
      while (not fin) and (not occupe) do
         begin
            for j := 1 to 4 do
               begin
                  if (n = a.nom_ress_cons [j]) and (date = a.date_activ) then
                     begin
                        if ((heure >= a.heure) and
                            (heure < a.heure + a.duree)) or
                           ((a.heure >= heure) and
                            (a.heure < heure + duree)) then
                           occupe := true;
                     end;
               end;
            lire_activite (num, a, fin);
         end;
   end;

{*****************************************************************************}

procedure test_lst_ressource (date : real; heure : real; duree : real;
                              l : liste_4; var exact : boolean);

   var occupe : boolean;
       i : integer;

   begin
      for i := 1 to 24 do writeln;
      exact := true;
      for i := 1 to 4 do
         begin
            if l [i] <>          then
               begin
                  test_disponible (date, heure, duree, l [i], occupe);
                  if occupe then
                     begin
                        exact := false;
                        erreur08 (l [i]);
                     end;
               end;
         end;
   end;

{*****************************************************************************}

procedure r_t_lst_ressource (date : real; heure : real; duree : real;
                             var l : liste_4);

   var i : integer;
       n : string8;
```

```
    exact, occupe : boolean;

begin
   i := 0;
   repeat
      write ('nom : ');
      r_ressource (n, exact);
      if exact
          then
            begin
               test_disponible (date, heure, duree, n, occupe);
               if occupe
                  then erreur07
                  else
                     begin
                        i := i + 1;
                        l [i] := n;
                     end;
            end
         else if n <> '        ' then erreur04;
   until (n = '        ') or (i = 4);
   while i < 4 do
      begin
         i := i + 1;
         l [i] := '        ';
      end;
end;

{*****************************************************************************}

procedure r_lst_ressource (var l : liste_4);

   var i : integer;
       n : string8;
       exact : boolean;

   begin
      i := 0;
      repeat
         write ('nom : ');
         r_ressource (n, exact);
         if exact
            then
               begin
                  i := i + 1;
                  l [i] := n;
               end
            else if n <> '        ' then erreur04;
      until (n = '        ') or (i = 4);
      while i < 4 do
         begin
            i := i + 1;
            l [i] := '        ';
         end;
   end;
```

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

procedure repeter_activite (var reponse : char;
                            var nbre : integer; var inter : integer);

   var i : integer;

   begin
      for i := 1 to 20 do writeln;
      writeln ('l''activite est enregistree...');
      writeln;
      writeln;
      writeln;
      repeat
         write ('voulez-vous la reporter sur d''autres journees ? ');
         r_reponse (reponse);
      until (reponse = 'o') or (reponse = 'n');
      if reponse = 'o' then
         begin
            repeat
               write ('combien de fois (NN) ? ');
               r_nombre (nbre);
            until nbre <> 0;
            repeat
               write ('nombre de jours separant deux de ces activites ');
               write ('(JJ) ? ');
               r_nombre (inter);
            until (inter > 0) and (inter < 29);
         end;
   end;


{*****************************************************************************}

procedure accede_agenda (n : string8; var nom : string8; var exact : boolean);

   var fin : boolean;
       p : pers;
       i, j, num : integer;

   begin
      i := 0;
      exact := false;
      repeat
         i := i + 1;
         write ('nom : ');
         lire_string (nom);
         writeln;
         ouvrir_personne (num);
         lire_personne (num, p, fin);
         if not fin then
            begin
               if n = p.nom_pers then exact := true;
            end;
         while (not fin) and (not exact) do
```

```
            begin
               if p.nom_pers = nom
                  then exact := true
                  else lire_personne (num, p, fin);
            end;
         if exact
            then
               begin
                  j := 1;
                  if nom <> n then exact := false;
                  while (not exact) and (j < 11) do
                     begin
                        if (n = p.nom_acces [j])
                           then exact := true
                           else j := j + 1;
                     end;
                  if not exact then erreur12;
               end
            else erreur03;
         fermer_personne (num);
      until (exact) or (i = 3);
   end;

{*************************************************************************}

procedure select_type (var sorte : string8);

   var reponse : char;

   begin
      sorte := '        ';
      repeat
         write ('desirez-vous selectionner un type d''activite ? ');
         r_reponse (reponse);
      until (reponse = 'o') or (reponse = 'n');
      if reponse = 'o' then
         begin
            write ('lequel ? ');
            lire_string (sorte);
            writeln;
         end;
   end;

{*************************************************************************}

procedure date_passee (date : real; date_du_jour : real;
                       var exact : boolean);

   begin
      if date < date_du_jour - 300
         then
            begin
               exact := false;
```

```
                writeln ('cette date ne fait plus partie de l''agenda !!!');
                writeln;
                writeln;
             end
          else exact := true;
   end;


{*****************************************************************************}

procedure r_intervalle (date_du_jour : real; no : char;
                        var date_debut : real; var date_fin : real);

   var exact : boolean;
       jo, mo, an : integer;

   begin
      case no of
         '1' : begin
                  repeat
                     write ('date de ce jour (JJ/MM/AA) : ');
                     r_date (date_debut, exact);
                     if exact then date_passee (date_debut, date_du_jour,
                                                exact);
                  until exact;
                  date_fin := date_debut;
               end;
         '2' : begin
                  repeat
                     write ('date du premier jour de cette semaine ');
                     write ('(JJ/MM/AA) : ');
                     r_date (date_debut, exact);
                     if exact then date_passee (date_debut, date_du_jour,
                                                exact);
                  until exact;
                  date_fin := date_debut + 6;
                  valid_date (date_fin);
               end;
         '3' : begin
                  repeat
                     write ('date d''un jour de ce mois (JJ/MM/AA) : ');
                     r_date (date_debut, exact);
                     if exact then date_passee (date_debut, date_du_jour,
                                                exact);
                  until exact;
                  decomp_date (date_debut, jo, mo, an);
                  date_debut := date_debut - jo + 1;
                  date_fin := date_debut - 1 + maxmois [mo];
               end;
         '4' : begin
                  repeat
                     write ('date du debut de l''intervalle (JJ/MM/AA) : ');
                     r_date (date_debut, exact);
                     if exact then date_passee (date_debut, date_du_jour,
                                                exact);
                  until exact;
                  repeat
```

```
                    write ('date de la fin de l''intervalle (JJ/MM/AA) : ');
                    r_date (date_fin, exact);
                    if exact then
                       begin
                          if date_fin < date_debut then
                             begin
                                exact := false;
                                erreur10;
                             end;
                       end;
                 until exact;
              end;
        end;
   end;

{*****************************************************************************}

procedure select_activites (n : string8; no : integer; nom : string8;
                            sorte : string8;
                            date_debut : real; date_fin : real;
                            var i : integer);

   var j, k, num : integer;
       a : activ;
       exact, fin : boolean;

   begin
      i := 0;
      ouvrir_activite (num);
      lire_activite (num, a, fin);
      while not fin do
         begin
            if ((sorte = '        ') or (sorte = a.sorte)) and
               (date_debut <= a.date_activ) and
               (date_fin >= a.date_activ) then
               begin
                  exact := false;
                  if no = 0
                     then
                        begin
                           if a.nom_enreg = nom then exact := true;
                           if a.nom_resp = nom then exact := true;
                           j := 0;
                           while (not exact) and (j < 10) do
                              begin
                                 j := j + 1;
                                 if (a.nom_pers_cons [j] = nom) and
                                    (a.code_reponse [j] = 'o') then
                                    exact := true;
                              end;
                           if (a.confid = 'o') and (nom <> n) then
                              begin
                                 for k := 1 to 30 do a.descr [k] := 'x';
                                 for k := 31 to 128 do a.descr [k] := ' ';
                              end;
```

```
                        end
                     else
                        begin
                           for j := 1 to 4 do
                              if a.nom_ress_cons [j] = nom then exact := true;
                        end;
                  if exact then
                     begin
                        i := i + 1;
                        mat_date [i] := a.date_activ;
                        mat_heure [i] := a.heure;
                        mat_duree [i] := a.duree;
                        mat_descr [i] := a.descr;
                     end;
               end;
            lire_activite (num, a, fin);
         end;
      fermer_activite (num);
   end;


{*****************************************************************************}


procedure tri_activites (i : integer);

   var date, heure, duree : real;
       descr : string128;
       j, k, l : integer;

   begin
      j := 1;
      while j < i do
         begin
            l := j;
            k := j + 1;
            while k <= i do
               begin
                  if (mat_date [j] > mat_date [k]) or
                    ((mat_date [j] = mat_date [k]) and
                    ((mat_heure [j] > mat_heure [k]) or
                    ((mat_heure [j] = mat_heure [k]) and
                     (mat_duree [j] > mat_duree [k])))) then l := k;
                  k := k + 1;
               end;
            if l <> j then
               begin
                  date := mat_date [j];
                  heure := mat_heure [j];
                  duree := mat_duree [j];
                  descr := mat_descr [j];
                  mat_date [j] := mat_date [l];
                  mat_heure [j] := mat_heure [l];
                  mat_duree [j] := mat_duree [l];
                  mat_descr [j] := mat_descr [l];
                  mat_date [l] := date;
                  mat_heure [l] := heure;
```

```
                mat_duree [l] := duree;
                mat_descr [l] := descr;
             end;
          j := j + 1;
       end;
   end;

{*****************************************************************************}

procedure affich_jour (heure : real; duree : real; descr : string128;
                       var cpt : integer);

   var j, k, he, min : integer;

   begin
      decomp_heure (heure, he, min);
      write (he : 3, '.');
      if min = 0
         then write ('  ')
         else write (min : 2);
      decomp_heure (duree, he, min);
      write (he : 4, '.');
      if min = 0
         then write ('     ')
         else write (min : 2, '   ');
      for j := 1 to 64 do write (descr [j]);
      writeln;
      cpt := cpt + 1;
      j := 65;
      repeat
         if descr [j] <> ' ' then
            begin
               j := 128;
               write ('                  ');
               for k := 65 to 128 do write (descr [k]);
               writeln;
               cpt := cpt + 1;
            end;
         j := j + 1;
      until j > 128;
   end;

{*****************************************************************************}

procedure aff_jour (nom : string8; sorte : string8; date : real; i : integer);

   var j, jo, mo, an, jj, cpt : integer;
       c : char;

   begin
      for j := 1 to 24 do writeln;
      write (nom, '                ', sorte, '                ');
      decomp_date (date, jo, mo, an);
```

```
      trouve_jour (jo, mo, an, jj);
      writeln (jour [jj], jo : 3, ' ', mois [mo], an : 5);
      writeln;
      write ('------------------------------------------------------------');
      writeln ('-------------------');
      writeln (' heure  duree   description');
      write ('------------------------------------------------------------');
      writeln ('-------------------');
      cpt := 5;
      j := 0;
      while j < i do
         begin
            j := j + 1;
            if date = mat_date [j] then
               affich_jour (mat_heure [j], mat_duree [j], mat_descr [j],
                            cpt);
         end;
      if cpt = 5 then
         begin
            writeln;
            writeln;
            write ('                pas d''activites ');
            if sorte <> '         ' then write ('de ce type ');
            writeln ('pour cette journee !!!');
            cpt := cpt + 3;
         end;
      while cpt < 20 do
         begin
            writeln;
            cpt := cpt + 1;
         end;
      write ('------------------------------------------------------------');
      writeln ('-------------------');
      writeln;
      write ('poussez sur <RETURN> ... ');
      repeat read (c) until eoln;
   end;

{*****************************************************************************}

procedure aff_intervalle (nom : string8; sorte : string8; date_debut : real;
                          date_fin : real; i : integer);

   var date : real;

   begin
      date := date_debut;
      while date <= date_fin do
         begin
            aff_jour (nom, sorte, date, i);
            date := date + 1;
            valid_date (date);
         end;
   end;
```

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

procedure prepar_semaine (date_debut : real; date_fin : real; i : integer;
                          var table : tbl_semaine);

   var j, k, l, m, jo, mo, an, jj, he, min : integer;

   begin
      for j := 1 to 7 do
         for k := 0 to 95 do
            table [j, k] := ' ';
      j := 0;
      while j < i do
         begin
            j := j + 1;
            if (date_debut <= mat_date [j]) and
               (date_fin >= mat_date [j]) then
               begin
                  decomp_date (mat_date [j], jo, mo, an);
                  trouve_jour (jo, mo, an, jj);
                  decomp_heure (mat_heure [j], he, min);
                  k := he * 4 + round (min / 15);
                  decomp_heure (mat_duree [j], he, min);
                  l := he * 4 + round (min / 15);
                  m := 0;
                  while m < l do
                     begin
                        m := m + 1;
                        if k > 95
                           then
                              begin
                                 if mat_date [j] < date_fin
                                    then
                                       begin
                                          k := k - 96;
                                          jj := jj + 1;
                                          if jj > 7 then jj := jj - 7;
                                          table [jj, k] := 'X';
                                          k := k + 1;
                                       end
                                    else m := l;
                              end
                           else
                              begin
                                 table [jj, k] := 'X';
                                 k := k + 1;
                              end;
                     end;
               end;
         end;
   end;

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

```
procedure affich_semaine (nom : string8; sorte : string8;
                          date_debut : real; date_fin : real;
                          table : tbl_semaine;
                          var j1 : integer; var j2 : integer);

   var j, k, l, mo, an, jj : integer;

   begin
      for j := 1 to 24 do writeln;
      write (nom, '                    ', sorte, '                    ');
      decomp_date (date_debut, j1, mo, an);
      trouve_jour (j1, mo, an, jj);
      writeln ('du ', jour [jj], j1 : 3, ' ', mois [mo], an : 5);
      decomp_date (date_fin, j2, mo, an);
      jj := jj - 1;
      if jj < 1 then jj := jj + 7;
      write ('                                                    ');
      writeln ('au ', jour [jj], j2 : 3, ' ', mois [mo], an : 5);
      write ('-------------------------------------------------');
      writeln ('----------------------------');
      write ('    heure  !');
      for j := 1 to 7 do
         begin
            jj := jj + 1;
            if jj > 7 then jj := jj - 7;
            write (jour [jj], '!');
         end;
      writeln;
      write ('-------------------------------------------------');
      writeln ('----------------------------');
      for j := 0 to 11 do
         begin
            write (j * 2 : 8, '    !');
            for k := 1 to 7 do
               begin
                  jj := jj + 1;
                  if jj > 7 then jj := jj - 7;
                  for l := 0 to 7 do write (table [jj, j * 8 + l]);
                  write ('!');
               end;
            writeln;
         end;
      writeln;
      writeln ('x = 1/4 d''heure d''occupation');
      write ('-------------------------------------------------');;
      writeln ('----------------------------');
   end;


{*****************************************************************************}

procedure aff_semaine (nom : string8; sorte : string8;
                       date_debut : real; date_fin : real; i : integer);

   var j1, j2, mo, an, choix : integer;
       date : real;
       table : tbl_semaine;
```

```
   begin
      prepar_semaine (date_debut, date_fin, i, table);
      repeat
         affich_semaine (nom, sorte, date_debut, date_fin, table, j1, j2);
         write ('les operations possibles sont : ');
         writeln ('JJ  <---  visualiser le jour JJ');
         write ('                                  ');
         writeln ('99  <---  retourner au menu precedent');
         writeln;
         write ('quel est votre choix ? ');
         r_nombre (choix);
         if ((choix >= j1) and (choix <= j2))
            then
               begin
                  date := date_debut - j1 + choix;
                  aff_jour (nom, sorte, date, i);
               end
            else
               begin
                  if j2 < j1 then
                     begin
                        decomp_date (date_debut, j1, mo, an);
                        if (choix >= j1) and (choix <= maxmois [mo])
                           then
                              begin
                                 date := date_debut - j1 + choix;
                                 aff_jour (nom, sorte, date, i);
                              end
                           else
                              begin
                                 if (choix <= j2) and (choix >= 0) then
                                    begin
                                       date := date_fin - j2 + choix;
                                       aff_jour (nom, sorte, date, i);
                                    end;
                              end;
                     end;
               end;
      until choix = 99;
   end;


{*************************************************************************}


procedure prepar_mois (date_debut : real; date_fin : real; i : integer;
                       var table : tbl_mois);

   var j, jo, mo, an : integer;

   begin
      for j := 1 to 31 do table [j] := ' ';
      j := 0;
      while j < i do
         begin
            j := j + 1;
            if (date_debut <= mat_date [j]) and
```

```
                (date_fin >= mat_date [j]) then
                begin
                   decomp_date (mat_date [j], jo, mo, an);
                   table [jo] := 'X';
                end;
          end;
   end;

{*****************************************************************************}

procedure affich_mois (nom : string8; sorte : string8;
                       date_debut : real; date_fin : real;
                       table : tbl_mois;
                       var j1 : integer; var j2 : integer);

   var j, mo, an, jj : integer;

   begin
      for j := 1 to 24 do writeln;
      write (nom, '                              ');
      write (sorte, '                                ');
      decomp_date (date_debut, j1, mo, an);
      writeln (mois [mo], an : 5);
      decomp_date (date_fin, j2, mo, an);
      write ('-------------------------------------------------');
      writeln ('----------------------------');
      writeln;
      trouve_jour (j1, mo, an, jj);
      for j := 1 to 7 do
         begin
            write ('    ', jour [jj]);
            jj := jj + 1;
            if jj > 7 then jj := jj - 7;
         end;
      writeln;
      writeln;
      writeln;
      j := 1;
      while j <= j2 do
         begin
            write (j : 6, '   ', table [j], '   ');
            if (j mod 7) = 0 then
               begin
                  writeln;
                  writeln;
                  writeln;
               end;
            j := j + 1;
         end;
      writeln;
      writeln;
      write ('-------------------------------------------------');
      writeln ('----------------------------');
   end;
```

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

procedure aff_mois (nom : string8; sorte : string8;
                    date_debut : real; date_fin : real; i : integer);

   var j1, j2, choix : integer;
       date : real;
       table : tbl_mois;

   begin
      prepar_mois (date_debut, date_fin, i, table);
      repeat
         affich_mois (nom, sorte, date_debut, date_fin, table, j1, j2);
         write ('les operations possibles sont : ');
         writeln ('JJ  <---  visualiser le jour JJ');
         write ('                                  ');
         writeln ('99  <---  retourner au menu precedent');
         write ('quel est votre choix ? ');
         r_nombre (choix);
         if (choix >= j1) and (choix <= j2) then
            begin
               date := date_debut - 1 + choix;
               aff_jour (nom, sorte, date, i);
            end;
      until choix = 99;
   end;


{*****************************************************************************}

procedure rentree_date_du_jour (var date_du_jour : real);

   var fin, exact : boolean;
       num, jo, mo, an, jj : integer;
       ap : apur;
       reponse : char;

   begin
      repeat
         write ('date du jour (JJ/MM/AA) : ');
         r_date (date_du_jour, exact);
         if exact then
            begin
               ouvrir_d_apurement (num);
               lire_d_apurement (num, ap, fin);
               fermer_d_apurement (num);
               if not fin then
                  begin
                     if ap.date_apur > date_du_jour then
                        begin
                           exact := false;
                           erreur11;
                        end;
                  end;
               if exact then
```

```
                begin
                   decomp_date (date_du_jour, jo, mo, an);
                   trouve_jour (jo, mo, an, jj);
                   repeat
                      write ('est-ce bien le ', jour [jj], jo : 3, ' ');
                      write (mois [mo], an : 5, ' ? ');
                      r_reponse (reponse);
                   until (reponse = 'o') or (reponse = 'n');
                   if reponse = 'n' then exact := false;
                end;
          end;
      until exact;
   end;


{*****************************************************************************}


procedure apurement (date_du_jour : real);

   var ap : apur;
       a : activ;
       date : real;
       num1, num2 : integer;
       fin : boolean;

   begin
      ouvrir_d_apurement (num1);
      lire_d_apurement (num1, ap, fin);
      if not fin
         then
            begin
               if (date_du_jour - ap.date_apur) >= 1 then
                  begin
                     for num2 := 1 to 10 do writeln;
                     write ('                ++++++++++++++++++++++++');
                     writeln ('++++++++++++++++++++++++++');
                     write ('                +                      ');
                     writeln ('                      +');
                     write ('                +  A P U R E M E N T   ');
                     writeln ('D E S   A G E N D A S    +');
                     write ('                +                      ');
                     writeln ('                      +');
                     write ('                ++++++++++++++++++++++++');
                     writeln ('++++++++++++++++++++++++++');
                     for num2 := 1 to 10 do writeln;
                     ap.date_apur := date_du_jour;
                     reecr_d_apurement (num1, ap);
                     date := date_du_jour - 300;
                     ouvrir_activite (num2);
                     lire_activite (num2, a, fin);
                     while not fin do
                        begin
                           if a.date_activ < date then suppr_activite (num2);
                           lire_activite (num2, a, fin);
                        end;
                     fermer_activite (num2);
```

```
                end;
            end
         else
            begin
               ap.date_apur := date_du_jour;
               reecr_d_apurement (num1, ap);
            end;
      fermer_apurement (num1);
   end;

{*****************************************************************************}

procedure visual_activites (n : string8);

   var num, i : integer;
       concerne, fin : boolean;
       a : activ;
       reponse : char;

   procedure aff_activ;
      var i, jj, jo, mo, an, he, min : integer;
      begin
         for i := 1 to 8 do writeln;
         writeln ('activite communiquee par : ', a.nom_enreg);
         writeln;
         decomp_date (a.date_activ, jo, mo, an);
         trouve_jour (jo, mo, an, jj);
         writeln ('date : ',jour [jj], jo : 3, ' ', mois [mo], an : 5);
         writeln;
         if a.heure <> 0.0 then
            begin
               decomp_heure (a.heure, he, min);
               write ('heure : ', he : 2, ' h ');
               if min = 0
                  then write ('  ')
                  else write (min : 2);
            end;
         if a.duree <> 0.0 then
            begin
               decomp_heure (a.duree, he, min);
               write ('          duree : ', he : 2, ' h ');
               if min = 0
                  then write ('  ')
                  else write (min : 2);
            end;
         writeln;
         writeln;
         writeln ('type : ', a.sorte);
         writeln;
         write ('description : ');
         for i := 1 to 64 do write (a.descr [i]);
         writeln;
         write ('                ');
         for i := 65 to 128 do write (a.descr [i]);
         writeln;
```

```
            writeln;
            if a.nom_resp <> '          ' then
               begin
                  writeln ('responsable : ', a.nom_resp);
                  writeln;
               end;
            writeln;
            writeln;
            writeln;
            writeln;
            writeln;
         end;

      begin
         for i := 1 to 10 do writeln;
         writeln ('                ++++++++++++++++++++++++++++++++++++++++++++');
         writeln ('                +                                          +');
         writeln ('                +  N O U V E L L E S   A C T I V I T E S  +');
         writeln ('                +                                          +');
         writeln ('                ++++++++++++++++++++++++++++++++++++++++++++');
         for i := 1 to 10 do writeln;
         ouvrir_activite (num);
         lire_activite (num, a, fin);
         while not fin do
            begin
               i := 0;
               concerne := false;
               while (not concerne) and (i < 10) do
                  begin
                     i := i + 1;
                     if a.nom_pers_cons [i] = n then concerne := true;
                  end;
               if (concerne) and (a.code_reponse [i] = ' ') then
                  begin
                     aff_activ;
                     repeat
                        write ('y serez-vous present ? ');
                        r_reponse (reponse);
                     until (reponse = ' ')
                        or (reponse = 'o') or (reponse = 'n');
                     if reponse <> ' ' then
                        begin
                           a.code_reponse [i] := reponse;
                           reecr_activite (num,a);
                        end;
                  end;
               lire_activite (num, a, fin);
            end;
         fermer_activite (num);
      end;


{*************************************************************************}


procedure choix_trt_agenda (var choix : char);
```

```
   begin
      repeat
         writeln;
         writeln;
         writeln ('                                    TRAITEMENTS POSSIBLES :');
         writeln ('                                    ----------------------');
         writeln;
         writeln;
         writeln;
         writeln ('1  <--  vue de l''agenda d''un abonne');
         writeln;
         writeln ('2  <--  vue de l''agenda d''une ressource');
         writeln;
         write ('3  <--  ajout d''une activite ou d''une serie');
         writeln (' d''activites');
         writeln;
         writeln ('4  <--  consultation et/ou mise a jour d''une activite');
         writeln;
         write ('5  <--  recherche de plages de temps libre');
         writeln (' suivant une date donnee');
         writeln;
         writeln;
         writeln ('9  <--  fin de travail');
         writeln;
         writeln;
         writeln;
         writeln;
         writeln;
         write ('                                       votre choix : ');
         r_reponse (choix);
         writeln;
      until ( (choix >= '1') and (choix <= '5') ) or (choix = '9');
   end;


{*************************************************************************}


procedure vue_agenda (date_du_jour : real;
                      n : string8; no : integer; nom : string8);

   var sorte : string8;
       date_debut, date_fin : real;
       numero : char;
       i : integer;

   begin
      select_type (sorte);
      repeat
         for i := 1 to 20 do writeln;
         write ('les operations possibles sont : ');
         writeln ('1  <---  visualiser un jour');
         write ('                                ');
         writeln ('2  <---  visualiser une semaine');
         write ('                                ');
         writeln ('3  <---  visualiser un mois ');
         write ('                                ');
```

```
            writeln ('4  <---  visualiser un intervalle quelconque');
            write ('                                    ');
            writeln ('9  <---  retourner au menu principal');
            writeln;
            write ('quel est votre choix ? ');
            r_reponse (numero);
            if (numero >= '1') and (numero <= '4') then
               begin
                  r_intervalle (date_du_jour, numero, date_debut, date_fin);
                  select_activites (n, no, nom, sorte, date_debut, date_fin, i);
                  tri_activites (i);
                  case numero of
                     '1' : aff_jour (nom, sorte, date_debut, i);
                     '2' : aff_semaine (nom, sorte, date_debut, date_fin, i);
                     '3' : aff_mois (nom, sorte, date_debut, date_fin, i);
                     '4' : aff_intervalle (nom, sorte, date_debut, date_fin, i);
                     end;
               end;
         until numero = '9';
      end;

{*****************************************************************************}

procedure vue_ag_personne (date_du_jour : real; n : string8);

   var i : integer;
       nom : string8;
       exact : boolean;

   begin
      for i := 1 to 20 do writeln;
      writeln ('                          VUE DE L''AGENDA D''UN ABONNE');
      writeln ('                          -------------------------');
      writeln;
      writeln;
      accede_agenda (n, nom, exact);
      if exact then vue_agenda (date_du_jour, n, 0, nom);
   end;

{*****************************************************************************}

procedure vue_ag_ressource (date_du_jour : real; n : string8);

   var i : integer;
       exact : boolean;
       nom : string8;

   begin
      for i := 1 to 20 do writeln;
      writeln ('                          VUE DE L''AGENDA D''UNE RESSOURCE');
      writeln ('                          ----------------------------');
      writeln;
      writeln;
```

```
      i := 0;
      repeat
         i := i + 1;
         write ('nom : ');
         r_ressource (nom, exact);
         if not exact then erreur04;
      until (exact) or (i = 3);
      if exact then vue_agenda (date_du_jour, n, 1, nom);
   end;


{*****************************************************************************}


procedure ajout_activite (date_du_jour : real; n : string8);

   var i, nbre, inter, jo, mo, an, jj, num : integer;
       exact : boolean;
       a : activ;
       c : char;

   begin
      for i := 1 to 20 do writeln;
      writeln ('                                    AJOUT D''UNE ACTIVITE');
      writeln ('                                    ------------------');
      writeln;
      writeln;
      i := 0;
      repeat
         i := i + 1;
         write ('date (JJ/MM/AA) : ');
         r_date (a.date_activ, exact);
         if exact then
            begin
               if a.date_activ < date_du_jour then
                  begin
                     exact := false;
                     erreur09;
                  end;
            end;
      until (exact) or (i = 3);
      if exact then
         begin
            repeat
               write ('heure (HH.MM) : ');
               r_heure (a.heure, exact);
            until exact;
            repeat
               write ('duree (HH.MM) : ');
               r_heure (a.duree, exact);
            until exact;
            write ('type : ');
            lire_string (a.sorte);
            writeln;
            write ('description : ');
            r_description (a.descr);
            repeat
```

```
            write ('est-ce confidentiel ? ');
            r_reponse (a.confid);
         until (a.confid = 'o') or (a.confid = 'n');
         a.nom_enreg := n;
         repeat
            write ('responsable : ');
            r_personne (a.nom_resp, exact);
            if (a.nom_resp <> '          ') and (not exact) then erreur03;
         until (exact) or (a.nom_resp = '          ');
         writeln ('liste des personnes concernees : ');
         writeln;
         r_lst_personne (a.nom_pers_cons);
         for i := 1 to 10 do a.code_reponse [i] := ' ';
         writeln ('liste des ressources consommees : ');
         writeln;
         r_t_lst_ressource (a.date_activ, a.heure, a.duree,
                            a.nom_ress_cons);
         ouvrir_activite (num);
         ecrire_activite (num, a);
         repeter_activiter (c, nbre, inter);
         if c = 'o' then
            begin
               for i := 1 to nbre do
                  begin
                     a.date_activ := a.date_activ + inter;
                     valid_date (a.date_activ);
                     test_lst_ressource (a.date_activ, a.heure, a.duree,
                                         a.nom_ress_cons, exact);
                     if exact
                        then ecrire_activite (num, a)
                        else
                           begin
                              decomp_date (a.date_activ, jo, mo, an);
                              trouve_jour (jo, mo, an, jj);
                              write ('===>  l''activite devant avoir ');
                              write ('lieu le ', jour [jj], jo : 3,' ');
                              writeln (mois [mo], an : 5);
                              writeln ('       n''est pas enregistree !!!');
                              writeln;
                              writeln;
                              writeln;
                              writeln;
                              write ('passer sur <RETURN> ... ');
                              repeat read (c) until eoln;
                           end;
                  end;
            end;
         fermer_activite (num);
      end;
   end;

{*****************************************************************************}

procedure cons_maj_activite (date_du_jour : real; n : string8);
```

```
var i, num : integer;
    exact, fin, sw1, sw2 : boolean;
    a, a_sauv : activ;
    date, heure : real;
    reponse : char;

procedure affichage_activite;
   var i, j, k, jj, jo, mo, an, he, min : integer;
   begin
      for i := 1 to 8 do writeln;
      write (' [1]                                          [2]' );
      writeln ('                    [3]' );
      decomp_date (a.date_activ, jo, mo, an);
      trouve_jour (jo, mo, an, jj);
      write ('date : ', jour [jj], jo : 3, ' ', mois [mo], an : 5);
      decomp_heure (a.heure, he, min);
      write ('      heure : ', he : 2, ' h ' );
      if min = 0
         then write ('  ' )
         else write (min : 2);
      decomp_heure (a.duree, he, min);
      write ('      duree : ', he : 2, ' h ' );
      if min = 0
         then writeln
         else writeln (min : 2);
      writeln (' [4]' );
      writeln ('type : ', a.sorte);
      writeln (' [5]' );
      write ('description : ' );
      for j := 1 to 64 do write (a.descr [j]);
      writeln;
      j := 65;
      repeat
         if a.descr [j] <> ' ' then
            begin
               j := 128;
               write ('                ' );
               for k := 65 to 128 do write (a.descr [k]);
               writeln;
            end;
         j := j + 1;
      until j > 128;
      writeln (' [6]                                          [7]' );
      write ('etat de confidentialite : ', a.confid);
      writeln ('          responsable : ', a.nom_resp);
      writeln (' [8]' );
      writeln ('liste des personnes concernees + leur reponse : ' );
      i := 1;
      repeat
         if a.nom_pers_cons [i] = '          '
            then i := 10
            else
               begin
                  if i = 6 then writeln;
                  write (a.nom_pers_cons [i], ' ' );
                  write (a.code_reponse [i], '      ' );
               end;
```

```
            i := i + 1;
         until i > 10;
         writeln;
         writeln ('[9]');
         writeln ('liste des ressources consommees :');
         i := 1;
         repeat
            if a.nom_ress_cons [i] = '        '
               then i := 4
               else write (a.nom_ress_cons [i], '        ');
            i := i + 1;
         until i > 4;
         writeln;
         writeln;
         write ('------------------------------------------------------');
         writeln ('----------------------------');
         writeln;
      end;

begin
   for i := 1 to 20 do writeln;
   write ('                    ');
   writeln ('CONSULTATION ET/OU MISE A JOUR D''UNE ACTIVITE');
   write ('                    ');
   writeln ('--------------------------------------------');
   writeln;
   writeln;
   repeat
      write ('date (JJ/MM/AA) : ');
      r_date (date, exact);
   until exact;
   repeat
      write ('heure (HH.MM) : ');
      r_heure (heure, exact);
   until exact;
   ouvrir_activite (num);
   lire_activite (num, a, fin);
   exact := false;
   while (not fin) and (not exact) do
      begin
         if (date = a.date_activ) and (heure = a.heure)
            then
               begin
                  if (n = a.nom_resp) or (n = a.nom_enreg)
                     then
                        begin
                           affichage_activite;
                           repeat
                              write ('est-ce bien cette activite ');
                              write ('que vous desirez ? ');
                              r_reponse (reponse);
                           until (reponse = 'o') or (reponse = 'n');
                           if reponse = 'o'
                              then exact := true
                              else lire_activite (num, a, fin);
                        end
                     else lire_activite (num, a, fin);
```

```
            end
         else lire_activite (num, a, fin);
   end;
if exact
   then
      begin
         suppr_activite (num);
         a_sauv := a;
         sw1 := false;
         sw2 := false;
         repeat
            affichage_activite;
            write ('les operations possibles sont : ');
            writeln ('i  <---  modifier la zone [i]');
            write ('                                ');
            writeln ('r  <---  reecrire l''activite');
            write ('                                ');
            writeln ('a  <---  ajouter l''activite');
            write ('                                ');
            writeln ('s  <---  supprimer l''activite');
            writeln;
            write ('quel est votre choix ? ');
            r_reponse (reponse);
            if (reponse >= '1') and (reponse <= '9') then
               begin
                  for i := 1 to 24 do writeln;
                  sw1 := true;
                  case reponse of
                     '1' : begin
                              repeat
                                 write ('date (JJ/MM/AA) : ');
                                 r_date (a.date_activ, exact);
                                 if exact then
                                    if a.date_activ < date_du_jour then
                                       begin
                                          exact := false;
                                          write ('cette date est deja');
                                          writeln ('passee !!!');
                                          writeln;
                                          writeln;
                                       end;
                              until exact;
                              sw2 := true;
                           end;
                     '2' : begin
                              repeat
                                 write ('heure (HH.MM) : ');
                                 r_heure (a.heure, exact);
                              until exact;
                              sw2 := true;
                           end;
                     '3' : begin
                              repeat
                                 write ('duree (HH.MM) : ');
                                 r_heure (a.duree, exact);
                              until exact;
                              sw2 := true;
```

```
                    end;
                '4' : begin
                        write ('type : ');
                        lire_string (a.sorte);
                        writeln;
                    end;
                '5' : begin
                        write ('description : ');
                        r_description (a.descr);
                    end;
                '6' : begin
                        repeat
                          write ('est-ce confidentiel ? ');
                          r_reponse (a.confid);
                        until (a.confid = 'o')
                          or (a.confid = 'n');
                    end;
                '7' : begin
                        repeat
                          write ('responsable : ');
                          r_personne (a.nom_resp, exact);
                          if (not exact) and
                            (a.nom_resp <> '          ') then
                            begin
                              erreur03;
                            end;
                        until exact
                          or (a.nom_resp = '          ');
                    end;
                '8' : begin
                        write ('liste des personnes ');
                        writeln ('concernees : ');
                        writeln;
                        r_lst_personne (a.nom_pers_cons);
                    end;
                '9' : begin
                        write ('liste des ressources ');
                        writeln ('consommees : ');
                        writeln;
                        r_t_lst_ressource (a.date_activ, a.heure,
                          a.duree, a.nom_ress_cons);
                        sw2 := false;
                    end;
            end;
        end;
until (reponse = 'r') or (reponse = 'a') or (reponse = 's');
if reponse <> 's' then
   begin
      if sw1 then
         begin
            a.nom_enreg := n;
            for i := 1 to 10 do a.code_reponse [i] := ' ';
         end;
      if reponse = 'a' then
         ecrire_activite (num, a_sauv);
      if (reponse = 'a') or (sw2) then
         begin
```

```
                        test_lst_ressource (a.date_activ, a.heure, a.duree,
                                            a.nom_ress_cons, exact);
                        if not exact then
                           begin
                              write ('liste des ressources consommees : ');
                              writeln;
                              writeln;
                              r_t_lst_ressource (a.date_activ, a.heure,
                                                 a.duree, a.nom_ress_cons);
                           end;
                     end;
                  ecrire_activite (num, a);
               end;
            fermer_activite (num);
         end
      else
         begin
            for i := 1 to 12 do writeln;
            write ('vous n''avez pas ou plus d''activites correspondant ');
            writeln ('a ces date et heure !!!');
            for i := 1 to 12 do writeln;
            write ('poussez sur <RETURN> ... ');
            r_reponse (reponse);
         end;
end;


{*****************************************************************************}


procedure rech_plage_tps_libre (date_du_jour : real);

   var i, j, k, num, jo, mo, an, jj, he, min : integer;
       date_debut, date_fin : real;
       fin, exact : boolean;
       a : activ;
       table : tbl_semaine;
       lst_personne : liste_string8;
       lst_ressource : liste_4;
       c : char;

   begin
      for i := 1 to 20 do writeln;
      write ('           RECHERCHE DE PLAGES DE TEMPS LIBRE SUIVANT UNE ');
      writeln ('DATE DONNEE');
      write ('           ------------------------------------------');
      writeln ('----------');
      writeln;
      writeln;
      i := 0;
      repeat
         i := i + 1;
         write ('date (JJ/MM/AA) : ');
         r_date (date_debut, exact);
         if exact then
            begin
               if date_debut < date_du_jour then
```

```
            begin
               exact := false;
               erreur09;
            end;
      end;
until (exact) or (i = 3);
if exact then
   begin
      date_fin := date_debut + 6;
      valid_date (date_fin);
      writeln ('liste des abonnes : ');
      writeln;
      r_lst_personne (lst_personne);
      writeln ('liste des ressources : ');
      writeln;
      r_lst_ressource (lst_ressource);
      for i := 1 to 7 do
         for j := 0 to 95 do
            table [i, j] := 'X';
      ouvrir_activite (num);
      lire_activite (num, a, fin);
      while not fin do
         begin
            if (date_debut <= a.date_activ) and
               (date_fin >= a.date_activ) then
               begin
                  exact := false;
                  i := 0;
                  while (i < 10) and (not exact) do
                     begin
                        i := i + 1;
                        if a.nom_enreg = lst_personne [i]
                           then exact := true;
                        if a.nom_resp = lst_personne [i]
                           then exact := true;
                        j := 0;
                        while (j < 10) and (not exact) do
                           begin
                              j := j + 1;
                              if (a.nom_pers_cons [j] =
                                  lst_personne [i]) and
                                 (a.code_reponse [j] = 'o') then
                                 exact := true;
                           end;
                     end;
                  i := 0;
                  while (i < 4) and (not exact) do
                     begin
                        i := i + 1;
                        j := 0;
                        while (j < 4) and (not exact) do
                           begin
                              j := j + 1;
                              if a.nom_ress_cons [j] =
                                 lst_ressource [i] then
                                 exact := true;
                           end;
```

```
                        end;
                     if exact then
                        begin
                           decomp_date (a.date_activ, jo, mo, an);
                           trouve_jour (jo, mo, an, jj);
                           decomp_heure (a.heure, he, min);
                           i := he * 4 + round (min / 15);
                           decomp_heure (a.duree, he, min);
                           j := he * 4 + round (min / 15);
k := 0;
                           while k < j do
                              begin
                                 k := k + 1;
                                 if i > 95
                                    then
                                       begin
                                          if a.date_activ < date_fin
                                             then
                                                begin
                                                   i := i - 96;
                                                   jj := jj + 1;
                                                   if jj > 7 then
                                                      jj := jj - 7;
                                                   table [jj, i] := ' ';
                                                   i := i + 1;
                                                end
                                             else k := j;
                                       end
                                    else
                                       begin
                                          table [jj, i] := ' ';
                                          i := i + 1;
                                       end;
                              end;
                        end;
                  end;
               lire_activite (num, a, fin);
            end;
         fermer_activite (num);
         for i := 1 to 24 do writeln;
         i := 1;
         repeat
            if lst_personne [i] = '          '
               then i := 10
               else
                  begin
                     if i = 6 then writeln;
                     write (lst_personne [i],'      ');
                  end;
            i := i + 1;
         until i > 10;
         writeln;
         i := 1;
         repeat
            if lst_ressource [i] = '          '
               then i := 4
               else write (lst_ressource [i], '      ');
            i := i + 1;
```

```
            until i > 4;
            writeln;
            writeln;
            write ('          ');
            decomp_date (date_debut, jo, mo, an);
            trouve_jour (jo, mo, an, jj);
            write ('du ', jour [jj], jo : 3, ' ', mois [mo], an : 5);
            decomp_date (date_fin, jo, mo, an);
            jj := jj -1;
            if jj < 1 then jj := jj + 7;
            writeln (' au ',jour [jj], jo : 3, ' ', mois [mo], an : 5);
            write ('-------------------------------------------------');
            writeln ('----------------------------');
            write ('    heure  !');
            for i := 1 to 7 do
               begin
                  jj := jj + 1;
                  if jj > 7 then jj := jj - 7;
                  write (jour [jj], '!');
               end;
            writeln;
            write ('-------------------------------------------------');
            writeln ('----------------------------');
            for i := 0 to 11 do
               begin
                  write (i * 2 : 8, '   !');
                  for j := 1 to 7 do
                     begin
                        jj := jj + 1;
                        if jj > 7 then jj := jj - 7;
                        for k := 0 to 7 do write (table [jj, i * 8 + k]);
                        write ('!');
                     end;
                  writeln;
               end;
            write ('x = 1/4 d''heure de temps libre pour l''ensemble ');
            writeln ('des abonnes et ressources');
            write ('-------------------------------------------------');
            writeln ('----------------------------');
            write ('poussez sur <RETURN> ... ');
            repeat read (c) until eoln;
         end;
   end;


{*****************************************************************************}


begin
   ident_personne (n, exact);
   if exact then
      begin
         init_date;
         rentree_date_du_jour (date_du_jour);
         apurement (date_du_jour);
         visual_activites (n);
         choix_trt_agenda (choix);
```

```
            while choix <> '9' do
               begin
                  case choix of
                     '1' : vue_ag_personne(date_du_jour, n);
                     '2' : vue_ag_ressource(date_du_jour, n);
                     '3' : ajout_activite (date_du_jour, n);
                     '4' : cons_maj_activite (date_du_jour, n);
                     '5' : rech_plage_tps_libre (date_du_jour);
                  end;
                  choix_trt_agenda (choix);
               end;
         end;
   end.
```

```
{$c+}

program persress (input, output);

const

#include "acces.const"

type string8 = packed array [1..8] of char;
     string128 = packed array [1..128] of char;
     liste_string = array [1..10] of string8;
     liste_char = array [1..10] of char;
     liste_4 = array [1..4] of string8;
     pers = record
               nom_pers : string8;
               mot_de_passe : string8;
               nom_acces : liste_string;
            end;
     ress = record
               nom_ress : string8;
               nom_resp : string8;
            end;
     activ = record
                date_activ : real;
                heure : real;
                duree : real;
                sorte : string8;
                descr : string128;
                confid : char;
                nom_enreg : string8;
                nom_resp : string8;
                nom_pers_cons : liste_string;
                code_reponse : liste_char;
                nom_ress_cons : liste_4;
             end;
     apur = record
               date_apur : real;
            end;

#include "acces.type"

var choix : char;
    nom : string8;
    exact : boolean;

#include "acces.procedure"


{*****************************************************************************}


procedure lire_string (var n : string8);

   var i : integer;
       c : char;

   begin
```

```
      n := '          ';
      i := 0;
      repeat
         read (c);
         if (i > 0) and (i < 9) then n [i] := c;
         i := i + 1;
      until eoln;
   end;

{*****************************************************************************}

procedure erreur01;

   begin
      writeln ('nom et/ou mot de passe incorrect !!!');
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure erreur02;

   begin
      writeln ('nom de ressource incorrect !!!');
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure erreur03;

   begin
      writeln ('cette ressource n''est pas connue du systeme !!!');
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure erreur05;

   begin
      writeln ('ce nom d''abonne existe deja !!!');
      writeln;
      writeln;
   end;
```

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

procedure erreur06;

   begin
      writeln ('ce nom de ressource existe deja !!!');
      writeln;
      writeln;
   end;

{*****************************************************************************}

procedure r_reponse (var reponse : char);

   var i : integer;
       c : char;

   begin
      reponse := ' ';
      i := 0;
      repeat
         read (c);
         if i = 1 then reponse := c;
         i := i + 1;
      until eoln;
      writeln;
   end;

{*****************************************************************************}

procedure r_personne (var n : string8; var exact : boolean);

   var fin : boolean;
       p : pers;
       num : integer;

   begin
      lire_string (n);
      writeln;
      exact := false;
      ouvrir_personne (num);
      lire_personne (num, p, fin);
      while (not fin) and (not exact) do
         begin
            if p.nom_pers = n
               then exact := true
               else lire_personne (num, p, fin);
         end;
      fermer_personne (num);
   end;
```

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

procedure r_ressource (var n : string8; var exact : boolean);

   var fin : boolean;
       r : ress;
       num : integer;

   begin
      lire_string (n);
      writeln;
      exact := false;
      ouvrir_ressource (num);
      lire_ressource (num, r, fin);
      while (not fin) and (not exact) do
         begin
            if r.nom_ress = n
               then exact := true
               else lire_ressource (num, r, fin);
         end;
      fermer_ressource (num);
   end;

{*****************************************************************************}

procedure r_lst_personne (var a : liste_string);

   var i : integer;
       n : string8;
       exact : boolean;

   begin
      i := 0;
      repeat
         write ('nom : ');
         r_personne (n, exact);
         if n <> '        ' then
            if exact
               then
                  begin
                     i := i + 1;
                     a [i] := n;
                  end
               else if n <> '        ' then erreur03;
      until (n = '        ') or (i = 10);
      while i < 10 do
         begin
            i := i + 1;
            a [i] := '        ';
         end;
   end;

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

```
procedure ident_personne (var n : string8; var exact : boolean);

   var fin : boolean;
       i : integer;
       p : pers;
       m : string8;
       num : integer;

   begin
      for i := 1 to 10 do writeln;
      writeln ('                    ************************************');
      writeln ('                    *                                  *');
      writeln ('                    *                                  *');
      writeln ('                    *     G E S T I O N N A I R E      *');
      writeln ('                    *                                  *');
      writeln ('                    *        D ' ' A G E N D A S        *');
      writeln ('                    *                                  *');
      writeln ('                    *                                  *');
      writeln ('                    ************************************');
      for i := 1 to 6 do writeln;
      i := 0;
      repeat
         i := i + 1;
         write ('nom : ');
         lire_string (n);
         writeln;
         write ('mot de passe : ');
         lire_string (m);
         writeln;
         exact := false;
         ouvrir_personne (num);
         lire_personne (num, p, fin);
         while (not fin) and (not exact) do
            begin
               if (p.nom_pers = n) and (p.mot_de_passe = m)
                  then exact := true
                  else lire_personne (num, p, fin);
            end;
         fermer_personne (num);
         if not exact then erreur01;
      until (exact) or (i = 3);
   end;


{*****************************************************************************}


procedure ident_administrateur (nom : string8; var exact : boolean);

   var p : pers;
       fin : boolean;
       num : integer;

   begin
      ouvrir_personne (num);
      lire_personne (num, p, fin);
      if p.nom_pers = nom
```

```
        then exact := true
        else exact := false;
     fermer_personne (num);
  end;

{*****************************************************************************}

procedure choix_trt (var choix : char);

  begin
     repeat
        writeln;
        writeln;
        writeln;
        writeln ('                                TRAITEMENTS POSSIBLES :');
        writeln ('                                ----------------------');
        writeln;
        writeln;
        writeln;
        writeln ('1  <--  liste des abonnes');
        writeln ('2  <--  ajout d''un abonne');
        writeln ('3  <--  modification du mot de passe d''un abonne');
        write ('4  <--  modification de la liste des acces a l''agenda ');
        writeln ('d''un abonne');
        writeln;
        writeln;
        writeln ('5  <--  liste des ressources');
        writeln ('6  <--  ajout d''une ressource');
        writeln ('7  <--  modification du responsable d''une ressource');
        writeln;
        writeln;
        writeln ('9  <--  fin de travail');
        writeln;
        writeln;
        writeln;
        writeln;
        write ('                                    votre choix : ');
        r_reponse (choix);
        writeln;
     until (choix >= '1') and (choix <= '9') and (choix <> '8');
  end;

{*****************************************************************************}

procedure vue_personnes;

  var i, j, num : integer;
      fin : boolean;
      p : pers;
      temp : char;

  begin
     for i := 1 to 20 do writeln;
```

```
      writeln ('                                    LISTE DES ABONNES');
      writeln ('                                    ----------------');
      writeln;
      writeln;
      writeln ('   nom    I  abonnes ayant acces a leur agenda');
      write ('----------I-------------------------------------------');
      writeln ('------------');
      i := 6;
      ouvrir_personne (num);
      lire_personne (num, p, fin);
      while not fin do
         begin
            while (not fin) and (i < 22) do
               begin
                  write (p.nom_pers,'  I  ');
                  for j := 1 to 5 do write (p.nom_acces [j], '  ');
                  writeln;
                  write ('          I  ');
                  for j := 6 to 10 do write (p.nom_acces [j], '  ');
                  writeln;
                  i := i+2;
                  lire_personne (num, p, fin);
               end;
            while i < 23 do
               begin
                  writeln;
                  i := i + 1;
               end;
            write ('poussez sur <RETURN> ... ');
            repeat read (temp) until eoln;
            i := 0;
         end;
      fermer_personne (num);
   end;


{*****************************************************************************}


procedure ajout_personne (nom : string8);

   var i, num : integer;
       exact : boolean;
       n : string8;
       p : pers;

   begin
      ident_administrateur (nom, exact);
      if exact then
         begin
            for i := 1 to 20 do writeln;
            writeln ('                                    AJOUT D''UN ABONNE');
            writeln ('                                    ------------------');
            writeln;
            writeln;
            i := 0;
            repeat
```

```
                write ('nom : ');
                r_personne (n, exact);
                if exact
                   then
                      begin
                         i := i + 1;
                         erreur05;
                      end
                   else
                      begin
                         p.nom_pers := n;
                         write ('mot de passe : ');
                         lire_string (p.mot_de_passe);
                         writeln;
                         writeln;
                         writeln ('personnes ayant acces a cet agenda :');
                         writeln;
                         r_lst_personne (p.nom_acces);
                         ouvrir_personne (num);
                         ecrire_personne (num, p);
                         fermer_personne (num);
                      end;
             until (not exact) or (i = 3);
          end;
   end;

{*****************************************************************************}

procedure modif_mot_de_passe (nom : string8);


   var p : pers;
       fin : boolean;
       i, num : integer;

   begin
      for i := 1 to 20 do writeln;
      writeln ('                              MODIFICATION D''UN MOT DE PASSE');
      writeln ('                              ------------------------------');
      writeln;
      writeln;
      ouvrir_personne (num);
      repeat
         lire_personne (num, p, fin)
      until (nom = p.nom_pers);
      write ('nouveau mot de passe : ');
      lire_string (p.mot_de_passe);
      reecr_personne (num, p);
      fermer_personne (num);
   end;

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

```
procedure modif_acces (nom : string8);

   var i, j, num : integer;
       fin : boolean;
       p : pers;

   begin
      for i := 1 to 20 do writeln;
      writeln (´                          MODIFICATION D´´UNE LISTE D´´ACCES´);
      writeln (´                          ------------------------------´);
      writeln;
      writeln;
      ouvrir_personne (num);
      repeat
         lire_personne (num, p, fin)
      until (nom = p.nom_pers);
      writeln (´liste des personnes ayant acces a cet agenda :´);
      writeln;
      for i := 0 to 1 do
         begin
            for j := 1 to 5 do
               begin
                  write (p.nom_acces [i*5+j],´       ´);
               end;
            writeln;
         end;
      writeln;
      writeln;
      writeln (´nouvelle liste  :´);
      writeln;
      r_lst_personne (p.nom_acces);
      reecr_personne (num, p);
      fermer_personne (num);
   end;


{*****************************************************************************}

procedure vue_ressources;

   var i, j, num : integer;
       r : ress;
       fin : boolean;
       temp : char;

   begin
      for i := 1 to 20  do writeln;
      writeln (´                                LISTE DES RESSOURCES´);
      writeln (´                                --------------------´);
      writeln;
      writeln;
      write (´   nom    I  responsable       nom    I  responsable´);
      writeln (´       nom    I  responsable´);
      write (´----------I------------    ----------I------------´);
      writeln (´    ----------I-------------´);
      i := 6;
      ouvrir_ressource (num);
```

```
      lire_ressource (num, r, fin);
      while not fin do
         begin
            while (not fin) and (i < 23) do
               begin
                  j := 0;
                  while (not fin) and (j < 3) do
                     begin
                        write (r.nom_ress, '  I  ', r.nom_resp);
                        lire_ressource (num, r, fin);
                        j := j + 1;
                        if j <> 3 then write ('       ');
                     end;
                  writeln;
                  i := i + 1;
               end;
            while i < 23 do
               begin
                  writeln;
                  i := i + 1;
               end;
            write ('poussez sur <RETURN> ... ');
            repeat read (temp) until eoln;
            i := 0;
         end;
      fermer_ressource (num);
   end;


{*****************************************************************************}


procedure ajout_ressource (nom : string8);

   var i, j, num : integer;
       exact : boolean;
       r : ress;

   begin
      ident_administrateur (nom, exact);
      if exact then
         begin
            for i := 1 to 20 do writeln;
            writeln ('                                   AJOUT D''UNE RESSOURCE');
            writeln ('                                   ---------------------');
            writeln;
            writeln;
            i := 0;
            repeat
               write ('nom : ');
               r_ressource (r.nom_ress, exact);
               if exact
                  then
                     begin
                        i := i + 1;
                        erreur06;
                     end
```

```
                else
                   begin
                      j := 0;
                      repeat
                         write ('nom du responsable : ');
                         r_personne (r.nom_resp, exact);
                         if not exact
                            then
                               begin
                                  j := j + 1;
                                  erreur03;
                               end
                            else
                               begin
                                  ouvrir_ressource (num);
                                  ecrire_ressource (num, r);
                                  fermer_ressource (num);
                               end;
                      until (exact) or (j = 3);
                   end;
             until (exact) or (i = 3) or (j = 3);
          end;
    end;

{*****************************************************************************}

procedure modif_responsable (nom : string8);

    var i, num : integer;
        fin, exact : boolean;
        r : ress;
        n : string8;

    begin
       ident_administrateur (nom, exact);
       if exact then
          begin
              for i := 1 to 20 do writeln;
             write ('                    MODIFICATION DU RESPONSABLE D''UNE');
             writeln (' RESSOURCE');
             write ('                    ----------------------------------');
             writeln ('---------');
             writeln;
             writeln;
             i := 0;
             repeat
                i := i + 1;
                write ('nom : ');
                lire_string (n);
                writeln;
                exact := false;
                ouvrir_ressource (num);
                lire_ressource (num, r, fin);
                while (not fin) and (not exact) do
                   begin
```

```
                    if r.nom_ress = n
                       then exact := true
                       else lire_ressource (num, r, fin);
                 end;
              if not exact then
                 begin
                    erreur02;
                    fermer_ressource (num);
                 end;
           until (exact) or (i = 3);
           if exact then
              begin
                 writeln ('responsable : ',r.nom_resp);
                 writeln;
                 i := 0;
                 repeat
                    write ('nouveau responsable : ');
                    r_personne (r.nom_resp, exact);
                    if not exact
                       then
                          begin
                             i := i + 1;
                             erreur03;
                          end
                       else reecr_ressource (num, r);
                    fermer_ressource (num);
                 until (exact) or (i = 3);
              end;
        end;
  end;

{*****************************************************************************}

begin
   ident_personne (nom, exact);
   if exact then
      begin
         choix_trt (choix);
         while choix <> '9' do
            begin
               case choix of
                  '1' : vue_personnes;
                  '2' : ajout_personne (nom);
                  '3' : modif_mot_de_passe (nom);
                  '4' : modif_acces (nom);
                  '5' : vue_ressources;
                  '6' : ajout_ressource (nom);
                  '7' : modif_responsable (nom);
                  end;
               choix_trt (choix);
            end;
      end;
end.
```

```
{ longueur maximum d'un fichier logique }
lmaxart = 510;
{ nombre maximum de fichiers }
maxfich = 14;

long_pers = 98;
long_ress = 20;
long_activ = 485;
long_apur = 8;
```

```
{ article de donnees }
data = packed array [1..lmaxart] of char;
ok = -1..0;
{ type de SEEK }
seekreq = (abs, rel, plus, absb, relb, plusb);
{ mode d'ouverture }
openmode = (r, w, rw);
{ integer positif }
posint = 0..maxint;
{ descripteur de fichier UNIX }
fdok = -1..maxfich;
{ description de l'article donnees }
artcont = record
              { flag de suppression }
              flagsup : char;
              { article }
              donnee : data;
          end;
```

```
    buffer : artcont;
    sw : ok;
    nbre : integer;

{*****************************************************************************}

function open (nom_fichier : string; mode : openmode) : fdok;
   extern;

function close (num : fdok) : ok;
   extern;

function seek (num : fdok; position : integer; mode : seekreq) : ok;
   extern;

function uread (num : fdok; var buffer : artcont; long : posint) : integer;
   extern;

function uwrite (num : fdok; var buffer : artcont; long : posint) : ok;
   extern;

{*****************************************************************************}

procedure conv_date (no : integer; var buf : string8; var date : real);

   var i : integer;

   begin
      if no = 0
         then
            begin
               date := (ord (buf [1]) - 48) * 100000.0
                     + (ord (buf [2]) - 48) * 10000.0
                     + (ord (buf [3]) - 48) * 1000
                     + (ord (buf [4]) - 48) * 100
                     + (ord (buf [5]) - 48) * 10
                     + (ord (buf [6]) - 48);
            end
         else
            begin
               i := trunc (date / 100000.0);
               buf [1] := chr (i + 48);
               date := date - i * 100000.0;
               i := trunc (date / 10000.0);
               buf [2] := chr (i + 48);
               date := date - i * 10000.0;
               i := trunc (date / 1000);
               buf [3] := chr (i + 48);
               date := date - i * 1000;
               i := trunc (date / 100);
               buf [4] := chr (i + 48);
               date := date - i * 100;
               i := trunc (date / 10);
```

```
            buf [5] := chr (i + 48);
            i := trunc (date - i * 10);
            buf [6] := chr (i + 48);
         end;
   end;

{*****************************************************************************}

procedure conv_heure (no : integer; var buf : string8; var heure : real);

   var i : integer;

   begin
      if no = 0
         then
            begin
               heure := (ord (buf [1]) - 48) * 10
                      + (ord (buf [2]) - 48)
                      + (ord (buf [4]) - 48) / 10
                      + (ord (buf [5]) - 48) / 100;
            end
         else
            begin
               i := trunc (heure / 10);
               buf [1] := chr (i + 48);
               heure := heure - i * 10;
               i := trunc (heure);
               buf [2] := chr (i + 48);
               buf [3] := '.';
               heure := heure - i;
               i := trunc (heure * 10);
               buf [4] := chr (i + 48);
               heure := heure - i / 10;
               i := trunc (heure * 100);
               buf [5] := chr (i + 48);
            end;
   end;

{*****************************************************************************}

procedure conv_pers (no : integer; var buffer : artcont; var p : pers);

   var i, j, k : integer;

   begin
      with buffer do
         begin
            if no = 0
               then
                  begin
                     for i := 1 to 8 do
                        begin
                           p.nom_pers [i] := donnee [i];
```

```
                        p.mot_de_passe [i] := donnee [i + 8];
                     end;
                  for j := 1 to 10 do
                     begin
                        k := 8 + j * 8;
                        for i := 1 to 8 do
                           p.nom_acces [j, i] := donnee [k + i];
                     end;
               end
            else
               begin
                  for i := 1 to 8 do
                     begin
                        donnee [i] := p.nom_pers [i];
                        donnee [i + 8] := p.mot_de_passe [i];
                     end;
                  for j := 1 to 10 do
                     begin
                        k := 8 + j * 8;
                        for i := 1 to 8 do
                           donnee [k + i] := p.nom_acces [j, i];
                     end;
               end;
      end;
   end;

{*****************************************************************************}

procedure conv_ress (no : integer; var buffer : artcont; var r : ress);

   var i : integer;

   begin
      with buffer do
         begin
            if no = 0
               then
                  begin
                     for i := 1 to 8 do
                        begin
                           r.nom_ress [i] := donnee [i];
                           r.nom_resp [i] := donnee [i + 10];
                        end;
                  end
               else
                  begin
                     for i := 1 to 8 do
                        begin
                           donnee [i] := r.nom_ress [i];
                           donnee [i + 10] := r.nom_resp [i];
                        end;
                  end;
         end;
   end;
```

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

procedure conv_activ (no : integer; var buffer : artcont; var a : activ);

   var i, j, k : integer;
       buf : string8;

   begin
      with buffer do
         begin
            if no = 0
               then
                  begin
                     for i := 1 to 6 do buf [i] := donnee [i];
                     conv_date (no, buf, a.date_activ);
                     for i := 1 to 5 do buf [i] := donnee [i + 6];
                     conv_heure (no, buf, a.heure);
                     for i := 1 to 5 do buf [i] := donnee [i + 11];
                     conv_heure (no, buf, a.duree);
                     for i := 1 to 8 do a.sorte [i] := donnee [i + 16];
                     for i := 1 to 128 do a.descr [i] := donnee [i + 24];
                     a.confid := donnee [345];
                     for i := 1 to 8 do
                        begin
                           a.nom_enreg [i] := donnee [i + 345];
                           a.nom_resp [i] := donnee [i + 353];
                        end;
                     for j := 1 to 10 do
                        begin
                           k := 353 + j * 8;
                           for i := 1 to 8 do
                              a.nom_pers_cons [j, i] := donnee [k + i];
                           a.code_reponse [j] := donnee [j + 441];
                        end;
                     for j := 1 to 4 do
                        begin
                           k := 443 + j * 8;
                           for i := 1 to 8 do
                              a.nom_ress_cons [j, i] := donnee [k + i];
                        end;
                  end
               else
                  begin
                     conv_date (no, buf, a.date_activ);
                     for i := 1 to 6 do donnee [i] := buf [i];
                     conv_heure (no, buf, a.heure);
                     for i := 1 to 5 do donnee [i + 6] := buf [i];
                     conv_heure (no, buf, a.duree);
                     for i := 1 to 5 do donnee [i + 11] := buf [i];
                     for i := 1 to 8 do donnee [i + 16] := a.sorte [i];
                     for i := 1 to 128 do donnee [i + 24] := a.descr [i];
                     donnee [345] := a.confid;
                     for i := 1 to 8 do
                        begin
                           donnee [i + 345] := a.nom_enreg [i];
                           donnee [i + 353] := a.nom_resp [i];
```

```
                        end;
                     for j := 1 to 10 do
                        begin
                           k := 353 + j * 8;
                           for i := 1 to 8 do
                              donnee [k + i] := a.nom_pers_cons [j, i];
                           donnee [j + 441] := a.code_reponse [j];
                        end;
                     for j := 1 to 4 do
                        begin
                           k := 443 + j * 8;
                           for i := 1 to 8 do
                              donnee [k + i] := a.nom_ress_cons [j, i];
                        end;
                  end;
         end;
   end;

{*****************************************************************************}

procedure conv_apur (no : integer; var buffer : artcont; var ap : apur);

   var i : integer;
       buf : string8;

   begin
      with buffer do
         begin
            if no = 0
               then
                  begin
                     for i := 1 to 6 do buf [i] := donnee [i];
                     conv_date (no, buf, ap.date_apur);
                  end
               else
                  begin
                     conv_date (no, buf, ap.date_apur);
                     for i := 1 to 6 do donnee [i] := buf [i];
                  end;
         end;
   end;

{*****************************************************************************}

procedure ouvrir_personne (var num : integer);

   begin
      num := open ("personne", rw);
   end;

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

```
procedure fermer_personne (num : integer);

   begin
      sw := close (num);
   end;

{*****************************************************************************}

procedure lire_personne (num : integer; var p : pers; var fin : boolean);

   begin
      sw := seek (num, 0, rel);
      repeat
         nbre := uread (num, buffer, long_pers)
      until (buffer.flagsup <> 'x') or (nbre <> long_pers);
      if nbre = long_pers
         then
            begin
                fin := false;
                conv_pers (0, buffer, p);
            end
         else
            begin
               fin := true;
            end;
   end;

{*****************************************************************************}

procedure ecrire_personne (num : integer; p : pers);

   begin
      sw := seek (num, 0, plus);
      conv_pers (1, buffer, p);
      sw := uwrite (num, buffer, long_pers);
   end;

{*****************************************************************************}

procedure reecr_personne (num : integer; p : pers);

   begin
      sw := seek (num, -(long_pers), rel);
      conv_pers (1, buffer, p);
      sw := uwrite (num, buffer, long_pers);
   end;

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

```
procedure ouvrir_ressource (var num : integer);

   begin
      num := open ("ressource", rw);
   end;

{*****************************************************************************}

procedure fermer_ressource (num : integer);

   begin
      sw := close (num);
   end;

{*****************************************************************************}

procedure lire_ressource (num : integer; var r : ress; var fin : boolean);

   begin
      sw := seek (num, 0, rel);
      repeat
         nbre := uread (num, buffer, long_ress)
      until (buffer.flagsup <> 'x') or (nbre <> long_ress);
      if nbre = long_ress
         then
            begin
               fin := false;
               conv_ress (0, buffer, r);
            end
         else
            begin
               fin := true;
            end;
   end;

{*****************************************************************************}

procedure ecrire_ressource (num : integer; r : ress);

   begin
      sw := seek (num, 0, plus);
      conv_ress (1, buffer, r);
      sw := uwrite (num, buffer, long_ress);
   end;

{*****************************************************************************}

procedure reecr_ressource (num : integer; r : ress);
```

```
   begin
      sw := seek (num, -(long_ress), rel);
      conv_ress (1, buffer, r);
      sw := uwrite (num, buffer, long_ress);
   end;

{*****************************************************************************}

procedure ouvrir_activite (var num : integer);

   begin
      num := open ("activite", rw);
   end;

{*****************************************************************************}

procedure fermer_activite (num : integer);

   begin
      sw := close (num);
   end;

{*****************************************************************************}

procedure lire_activite (num : integer; var a : activ; var fin : boolean);

   begin
      sw := seek (num, 0, rel);
      repeat
         nbre := uread (num, buffer, long_activ)
      until (buffer.flagsup <> 'x') or (nbre <> long_activ);
      if nbre = long_activ
         then
            begin
               fin := false;
               conv_activ (0, buffer, a);
            end
         else
            begin
               fin := true;
            end;
   end;

{*****************************************************************************}

procedure ecrire_activite (num : integer; a : activ);

   begin
      sw := seek (num, 0, plus);
```

```
      conv_activ (1, buffer, a);
      sw := uwrite (num, buffer, long_activ);
   end;

{*****************************************************************************}

procedure reecr_activite (num : integer; a : activ);

   begin
      sw := seek (num, -(long_activ), rel);
      conv_activ (1, buffer, a);
      sw := uwrite (num, buffer, long_activ);
   end;

{*****************************************************************************}

procedure suppr_activite (num : integer);

   begin
      sw := seek (num, -(long_activ), rel);
      buffer.flagsup := 'x';
      sw := uwrite (num, buffer, long_activ);
      buffer.flagsup := ' ';
   end;

{*****************************************************************************}

procedure ouvrir_d_apurement (var num : integer);

   begin
      num := open ("apurement", rw);
   end;

{*****************************************************************************}

procedure fermer_d_apurement (num : integer);

   begin
      sw := close (num);
   end;

{*****************************************************************************}

procedure lire_d_apurement (num : integer; var ap : apur; var fin : boolean);

   begin
      sw := seek (num, 0, abs);
```

```
      repeat
         nbre := uread (num, buffer, long_apur)
      until (buffer.flagsup <> 'x') or (nbre <> long_apur);
      if nbre = long_apur
         then
            begin
               fin := false;
               conv_apur (0, buffer, ap);
            end
         else
            begin
               fin := true;
            end;
   end;

{*************************************************************************}

procedure reecr_d_apurement (num : integer; ap : apur);

   begin
      sw := seek (num, 0, abs);
      conv_apur (1, buffer, ap);
      sw := uwrite (num, buffer, long_apur);
   end;

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