L'agenda électronique

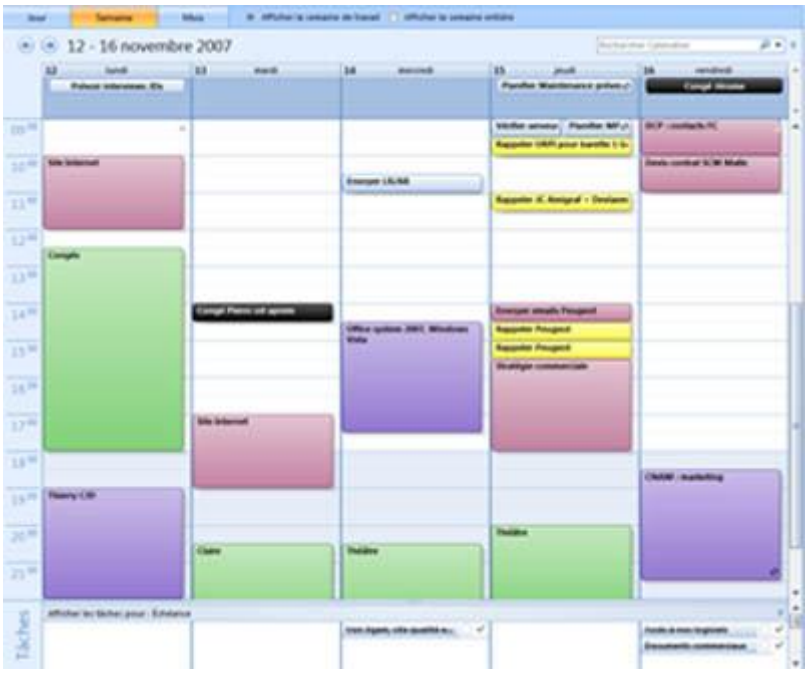

Figure 1 : vision d'un agenda électronique pour une semaine donnée

Dans cette application, nous allons programmer en C++ les sous-programmes et le programme nécessaire à la saisie et à l'affichage d'un agenda électronique pour une semaine en cours. Notre interface homme-machine sera rudimentaire, puisque nous ne disposerons que du clavier pour lire des données et d'une console Unix pour afficher l'agenda.

Nous émettons les hypothèses simplificatrices suivantes :

- un événement ne peut débuter avant 8 heures le matin.
- un événement ne peut finir après 20 heures en soirée.
- l'unité de temps est une demi-heure, la durée d'un événement est un multiple d'une demi-heure de même que pour les heures de début et fin d'un évènement.
- 8 heures 30 minutes se traduit par le réel 8.50, 9 heures se traduit par le réel 9.00.

Par exemple :

```
{MERCREDI, 9.00, 11.00, "ALG", "Controle Algorithmique 2"}
```

est un événement valide.

```
{MARDI, 18.45, 18.47, "ALG", "Je revise mon Controle Algorithmique"}
```

est un événement non valide.

```
{MARDI, 20.50, 22.50, "ALG", "Je bois pour oublier que je n'ai pas revise"}
```

est un événement non valide.

```
{MARDI, 13.50, 15.00, "NOC", "Controle noyau C+"}
```

est un événement valide.

Afin de rendre l'application modulaire, nous avons choisi de la scinder en plusieurs fichiers. En effet, l'application met clairement en évidence deux types de données à manipuler : un *événement* et un *agenda*.
C'est ainsi que nous avons choisi d'utiliser 5 fichiers :

- `evenement.h` contenant la structure des données liées à un événement et les entêtes des sous-programmes manipulant un événement,
- `evenement.C` contenant le corps des sous-programmes manipulant un événement,
- `agenda.h` contenant la structure des données liées à un agenda et les entêtes des sous-programmes manipulant un agenda,
- `agenda.C` contenant le corps des sous-programmes manipulant un agenda,
- `clientAgenda.C` contenant le programme principal de l'application permettant d'éditer un agenda.

1) Les événements

Représentation interne :

Un événement est composé d'un numéro de jour dans la semaine (compris entre un et sept), d'une heure de début (de type réel terminant par .0 ou .5), d'une heure de fin (de type réel terminant par .0 ou .5), d'une description abrégée encodée par une chaîne de 3 caractères, et d'une description étendue encodée par une chaîne de caractères.

Les jours de la semaine seront codés par des constantes symboliques et les chaînes de caractères seront représentées par des *string* (classe standard de C++).

Opérations :

Nous souhaitons disposer des opérations suivantes :

```
-- teste la validité d'un événement
-- nécessite jour dans la semaine compris entre 1 et 7
-- nécessite heureDébut compris entre 8.0 et 19.5 heures
-- nécessite heureFin compris entre 8.5 et 20.0 heures
-- nécessite partie décimale des heures égales à .0 ou .5
-- nécessite heureFin - heureDébut >= 0.5
procédure validerEvénement(entrée e <Evenement>)
    déclenche jourInvalide, heureDébutIncorrecte, heureFinIncorrecte, DuréeIncorrecte;
```

```
-- retourne vrai si la partie décimale de h est égale à 0.0 ou 0.5
fonction heureValide(entrée h <Réel>) retourne <Booléen> ;
```

```
-- saisit un événement e au clavier
-- nécessite jour dans la semaine compris entre 1 et 7
-- nécessite heureDébut compris entre 8.0 et 19.5 heures
-- nécessite heureFin compris entre 8.5 et 20.0 heures
-- nécessite partie décimale des heures égales à .0 ou .5
-- nécessite heureFin - heureDébut >= 0.5
procédure saisirEvénement(sortie e <Evenement>)
    déclenche jourInvalide, heureDebutIncorrecte, heureFinIncorrecte, DuréeIncorrecte;
```

```
-- initialise un événement e à partir de j : jour de la semaine,
-- hD : heure de début, hF : heure de fin, a : description abrégé et
-- d : description détaillée
-- nécessite 1 <= jour <= 7
-- nécessite 8.0 <= heureDébut <= 19.5
-- nécessite 8.5 <= heureFin <= 20.0
-- nécessite partieDécimale(heureDébut) = 0.0 ou 0.5
-- nécessite partieDécimale(heureFin) = 0.0 ou 0.5
-- nécessite heureFin - heureDébut >= 0.5
procédure initialiserEvenement(entrée j <Entier>, entrée hD <Réel>,    entrée hF <Réel>
    entrée a <ChaîneCaractères>, entrée d <ChaîneCaractères>, sortie e <Evenement>)
    déclenche jourInvalide, heureDébutIncorrecte, heureFinIncorrecte, DuréeIncorrecte;
```

```
-- affiche un événement e sous la forme
-- heureDebut(4 positions) - heureFin(4 positions) : description
-- par exemple :
--  8.0 -  9.5 : Controle Acsi MLD
--  9.5 - 11.0 : Controle Acsi MLD suite et fin
procédure afficherEvenement(entrée e <Evenement>);
```

```
-- teste si l'événement e1 est antérieur à l'événement e2
fonction estAnterieur(entrée e1 <Evenement>, entrée e2 <Evenement>) retourne <Booléen>;
```

```
-- affiche le jour j de la semaine en toutes lettres
-- nécessite 1 <= j <= 7
procédure afficherJourSemaine(entrée j <Entier>) déclenche jourInvalide;
```

```
-- affiche les 2 premières lettres du jour de la semaine j
-- "Lu", "Ma", ...
-- nécessite 1 <= j <= 7
procédure afficherDebutJourSemaine(entrée j <Entier>) déclenche jourInvalide;
```

Travail à réaliser :

Définir les fichiers `evenement.h` et `evenement.C` puis un programme de test contenant le `main()` qui permet de vérifier le bon fonctionnement de tous les sous-programmes. Pour la mise au point, on définira un fichier `makefile` de l'application en précisant toutes les dépendances entre fichiers.

Remarques :
Pour la fonction *heureValide()*, on pourra se servir de la fonction standard *floor()* qui retourne la partie entière d'une réel :

```
float floor(float x );
```

Son utilisation nécessite l'inclusion de la bibliothèque standard C *math* :

```
#include <cmath>
```

Pour la fonction *afficherEvenement()*, on pourra utiliser les manipulateurs sur le flot standard de sortie.
Leur utilisation nécessite l'inclusion de la bibliothèque standard C++ *iomanip* :

```
#include <iomanip>
```

2) L'agenda

Représentation interne :

Un agenda est un ensemble d'événements placés dans une semaine.
Voilà sa représentation en langage algorithmique :

```
constante  NB_EVENEMENTS <Entier> = 7 * 5 ;
type Semaine : tableau [1 à NB_EVENEMENTS] de <Evenement> ;

type Agenda : enregistrement
       semaine <Semaine>, -- tableau des événements de la semaine
       nbEvenements <Entier> ; -- nb évènements de la semaine
```

Opérations :

Nous souhaitons disposer des opérations suivantes :

```
-- initialise à vide l'agenda a
procédure initialiserAgenda(sortie a <Agenda>);
```

```
-- ajoute à la bonne place un événement e dans l'agenda a trié par ordre
-- chronologique des événenements
-- nécessite agenda non plein
procedure placerEvenement(màj a <Agenda> a, entrée e <Evenement>) déclenche débordement;
```

```
-- édite l'agenda a en affichant les événements pour chaque jour
-- de la semaine
procédure afficherAgendaDetail(entrée a <Agenda>);
```

```
// édite l'agenda global a
procédure afficherAgendaGlobal(entrée a <Agenda>);
```

Remarques :

* La procédure `afficherAgendaDetail()` affiche sur la sortie standard l'agenda de la manière suivante :

```
*** AFFICHAGE AGENDA DETAIL ***

Lundi   :
     8.0 -  9.5 : Controle Acsi MLD
     9.5 - 11.0 : Controle Acsi MLD suite et fin
    14.0 - 15.5 : Permanence Algo et Noyau C+ DD

Mardi   :
    13.5 - 15.0 : Controle noyau C+
    16.0 - 18.0 : Controle Mathematiques Discretes

Mercredi:
     9.0 - 11.0 : Controle Algorithmique 2
    17.0 - 18.5 : Comptabilite Rattrapage

Vendredi:
    11.0 - 12.5 : Architecture Rattrapage
```

On peut remarquer que :

- tous les événements sont affichés dans l'ordre d'apparition dans la semaine en cours.
- Jeudi, Samedi et Dimanche n'apparaissent pas car ces journées ne contiennent pas d'événement à afficher.

On pourra bien entendu définir d'autres sous-programmes s'il y a lieu pour décrire le fonctionnement global.

* La procédure `afficherAgendaGlobal()` affiche sur la sortie standard l'agenda de la manière suivante :

```
*** AFFICHAGE AGENDA GLOBAL ***

    08 -  09 -  10 -  11 -  12 -  13 -  14 -  15 -  16 -  17 -  18 -  19

Lu  MLD......MLD......                  ALG......
Ma                                   NOC......      MTD.........
Me        ALG.........                                      CPT......
Je
Ve                    ARC......
Sa
Di
```

On peut remarquer que :

- le nom abrégé d'un événement se compose de 3 caractères et que la durée minimale d'un événement est de ½ heure,
- le nombre de points lorsqu'il existe est un multiple de 3 et il correspond au nombre de demi-heures -1 de la durée de l'événement.

Pour réaliser cet affichage, on se servira d'un tableau de 7 chaînes de caractères (type *string*) initialisé comme suit :

```
"08 -  09 -  10 -  11 -  12 -  13 -  14 -  15 -  16 -  17 -  18 -  19"
"                                                                     "
"                                                                     "
"                                                                     "
"                                                                     "
"                                                                     "
"                                                                     "
"                                                                     "
```

Puis on rajoutera les événements de l'agenda dans les lignes correspondantes en se servant de la méthode `replace()` définie dans la classe standard *string*.

Travail à réaliser :

Définir les fichiers `agenda.h` et `agenda.C` puis un programme de test `agendaClient.C` contenant le *main()* qui permet de remplir un agenda avec les événements de la semaine, d'éditer l'agenda détaillé puis éditer l'agenda global associés. Pour la mise au point, on définira un fichier *makefile* de l'application en précisant toutes les dépendances entre fichiers.

3) Pour aller plus loin

a) modifier la procédure `saisirEvenement()` pour qu'elle intercepte toute anomalie et propose à l'utilisateur de saisir à nouveau l'événement (*try* … *catch*).

b) Pourquoi ne pas stocker les événements dans un fichier ?

En C++, les entrées-sorties sont gérées au travers des flots. Un flot peut être assimilé à un canal par lequel transitent des séquences d'octets.

Dans le cas où le canal fournit de l'information, on parle de flot d'entrée, alors que dans le cas où le canal reçoit de l'information, on parle de flot de sortie.

Un flot peut être connecté à un périphérique ou un fichier.

Si le flot est connecté à un périphérique, on parle de flot standard. Les flots standards sont manipulés au travers des classes `istream` (flot standard d'entrée) et `ostream` (flot standard de sortie).

Il existe 3 flots standards d'entrée/sortie définis par le langage (inutile de les déclarer) :

- *cin* (float standard d'entrée associé au clavier),
- *cout* (float standard de sortie associé à l'écran),
- *cerr* (float standard de sortie associé à l'écran)

Les flots connectés à des fichiers sont manipulés au travers des classes `ifstream` (flot d'entrée pour lire des fichiers), `ofstream` (flot de sortie pour écrire dans des fichiers).

Voici quelques opérations associées à un flot connecté à un fichier.

`int open(char []);`	Ouvre un fichier externe et l'associe à un flot. Cette fonction retourne 0 si l'opération ne s'est pas effectuée correctement
`void close();`	Ferme un fichier et le dissocie du flot
`bool is_open();`	Examine si le flot est déjà connecté à un fichier
`bool eof();`	Examine si la fin de fichier est atteinte

Exemple : création d'un fichier de noms

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main ()
{
    ofstream f;
    f.open("noms.dat");
    string n ;
    cout <<"entrer un nom ou # pour terminer" ;
    cin >>n ;
    while (n != "#")
    {
        f <<n <<endl;
        cout <<"entrer un nom ou # pour terminer";
```

```
        cin >>n;
    }
    f.close();
}
```

Exemple : lecture d'un fichier de noms

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
    ifstream f;
    f.open("noms.dat");
    string n;
    f >>n;
    while (!f.eof())
    {
        cout <<n<<"\t";
        f >>n;
    }
    f.close();
}
```

Créer sous l'éditeur de texte un fichier nommé « `ficEvenements` », puis saisir dans le fichier les événements comme ci-dessous :

```
5 11 12 ARC Architecture_Rattrapage
3 17 18.5 CPT Comptabilite_Rattrapage
1 14 15.5 ALG Permanence_Algo_et_Noyau_C+_DD
1 8 9.5 MLD Controle_Acsi_MLD
1 9.5 11 MLD Controle_Acsi_MLD_suite_et_fin
2 13.5 15 NOC Controle_noyau_C+
2 16 18 MTD Controle_Mathematiques_Discretes
3 9 11 ALG Controle_Algorithmique_2
```

Ajouter au type *Agenda* l'opération `lireFichierEvenements()` spécifiée en langage algorithmique par :

```
-- lit le fichier texte "ficEvenements" contenant des événements
-- en construisant au fur et à mesure l'agenda a
-- pour chaque événement du fichier :
-- nécessite jour dans la semaine compris entre 1 et 7
-- nécessite heureDébut compris entre 8.0 et 19.5 heures
-- nécessite heureFin compris entre 8.5 et 20.0 heures
-- nécessite partie décimale des heures égales à .0 ou .5
-- nécessite heureFin - heureDébut >= 0.5
procédure lireFichierEvenements(sortie a <Agenda>)
    déclenche jourInvalide, heureDébutIncorrecte, heureFinIncorrecte, DuréeIncorrecte;
```

Tester l'opération dans l'application `agendaClient.C`.