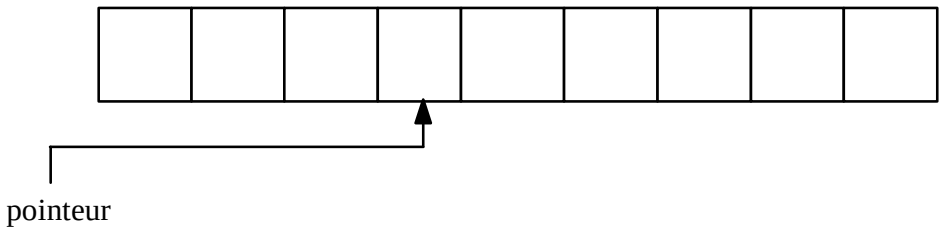
LES FICHIERS - GENERALITÉS (1)

Un fichier est un ensemble d'informations stockées sur une mémoire de masse (disque dur, disquette, bande magnétique, CD-ROM).

En C, un fichier est une suite d'octets. Les informations contenues dans le fichier ne sont pas forcément de même type (un **char**, un **int**, une **struct**ure ...)
Un **pointeur** fournit l'adresse d'une information quelconque.



On distingue généralement deux types d'accès aux fichiers:

1- Accès séquentiel (traite les informations dans l'ordre où elles apparaissent sur les bandes magnétiques).

- On accède à une cellule quelconque en se déplaçant (via un pointeur), depuis la cellule de départ.
- On ne peut pas détruire une cellule.
- On peut, par contre, tronquer la fin du fichier.
- On peut ajouter une cellule à la fin.

LES FICHIERS - GENERALITÉS (2)

2- Accès direct (RANDOM I/O) (utilisé sur disques, disquettes, CD-ROM où l'accès séquentiel est possible aussi).

- On peut directement accéder à une cellule.
- On peut modifier (voir détruire) n'importe quelle cellule.

Il existe deux façons de coder les informations stockées dans un fichier:

1- En binaire:

Fichier dit "binaire", les informations sont codées telles que. On désigne par fichier binaire tout fichier autre que le fichier texte seul (ASCII): photos, fichiers sons, tableaux, document conçu par un traitement de texte, etc. Ces fichiers ne sont pas listables (chaque enregistrement logique n'occupe pas un enregistrement physique).

2- en ASCII:

Fichier dit "texte", les informations sont codées en ASCII. Ces fichiers sont listables (chaque enregistrement logique occupe un enregistrement physique). Le dernier octet de ces fichiers est EOF (caractère ASCII spécifique).

MANIPULATION DES FICHIERS

Opérations possibles avec les fichiers: Créer - Ouvrir - Fermer - Lire - Écrire - Détruire - Renommer. La plupart des fonctions permettant la manipulation des fichiers sont rangées dans la bibliothèque standard STDIO.H.

Le langage C ne distingue pas les fichiers à accès séquentiel des fichiers à accès direct, certaines fonctions de la bibliothèque livrée avec le compilateur permettent l'accès direct. Les fonctions standard sont des fonctions d'accès séquentiel.

1 - Déclaration: **FILE *f1;** /* majuscules obligatoires pour FILE */

On définit un pointeur. Ce pointeur fournit l'adresse d'une cellule donnée.

2 - Ouverture: **FILE *fopen(char *nom, char *mode);**
On passe donc 2 chaînes de caractères:

nom: celui figurant sur le disque,
Exemple:

"a : \toto.dat"

mode d’accès aux fichiers (pour les **fichiers TEXTES**):

" r " lecture seule.

" w " écriture seule (destruction de l'ancienne version si elle existe).

" w+ " lecture/écriture (destruction de l’ancienne version si elle existe).

" r+ " lecture/écriture d'un fichier existant (mise à jour); pas de création d'une nouvelle version.

" a+ " lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version, le pointeur est positionné à la fin du fichier.

mode d’accès aux fichiers (pour les **fichiers BINAIRES**):

" rb " lecture seule.

" wb " écriture seule (destruction de l'ancienne version si elle existe).

" wb+ " lecture/écriture (destruction de l’ancienne version si elle existe).

" rb+ " lecture/écriture d'un fichier existant (mise à jour); pas de création d'une nouvelle version.

" ab+ " lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version, le pointeur est positionné à la fin du fichier.

À l’ouverture, le pointeur est positionné au début du fichier (sauf " a+ " et " ab+ ")

Exemple:

```
FILE *fichier ;
fichier = fopen(" a :\toto.dat ", " rb ") ;
```

3 - Fermeture: **int fclose(FILE *fichier);**

Retourne 0 si la fermeture s'est bien passée, EOF en cas d'erreur. Il faut toujours fermer un fichier à la fin d'une session.

Exemple:

```
FILE *fichier ;
fichier = fopen(" a :\toto.dat ", " rb ") ;
/* Ici on place les instructions de traitement */
fclose(fichier) ;
```

4 - Destruction: **int remove(char *nom);**

Retourne 0 si la fermeture s'est bien passée.

Exemple: remove(" a :\toto.dat ") ;

5 - Renommer: **int rename(char *oldname, char *newname);**

Retourne 0 si la fermeture s'est bien passée.

6 - Positionnement du pointeur au début: **void rewind(FILE *fichier);**

7- Écriture dans le fichier: **int putc(char c, FILE *fichier);**

Écrit la valeur de c à la position courante du pointeur, le pointeur avance d'une case mémoire. Retourne EOF en cas d'erreur.

Exemple: putc('A', fichier) ;

8 - Lecture du fichier: **int getc(FILE *fichier);**

Lit un caractère, mais retourne un entier n; retourne EOF si erreur ou fin de fichier; le pointeur avance d'une case.

Exemple:

```
char c ;
c = (char) getc(fichier) ;
```

9 - Gestion des erreurs:

fopen retourne le pointeur NULL en cas d'erreur (*exemple*: impossibilité d'ouvrir le fichier).

fgets retourne le pointeur NULL en cas d'erreur ou si la fin de fichier est atteinte.

10 - Fonction particulière aux fichiers à accès direct: **int fseek(FILE *fichier, int offset, int direction)** déplace le pointeur de *offset* cases à partir de *direction*.

Valeurs possibles pour direction:

0 -> à partir du début du fichier.

1 -> à partir de la position courante du pointeur.

2 -> en arrière, à partir de la fin du fichier.

Retourne 0 si le pointeur a pu être déplacé;

Bibliothèque standard du C

La bibliothèque standard est une extension intelligente du langage C, cette bibliothèque constituée d'utilitaires sous forme de programmes laisse le langage lui-même sous une forme plus concise, plus essentielle. Le programmeur peut alors lui-même enrichir cette bibliothèque ou créer une bibliothèque particulière à un projet déterminé.

La bibliothèque dite standard du C comprend les fonctions pour traités:

- *Les signaux, contrôle de processus*
- *Les opérations sur caractères et chaînes*
- *Les fonctions de gestion de la mémoire*
- *Les fonctions de conversions*
- *Les calculs sur les heures*
- *Les fonctions mathématiques*
- *Les fonctions de gestion des entrées-sorties*
- *Les fonctions de communications*
- *Les fonctions de manipulations de fichiers*
- *Les fonctions de gestion systèmes*

Pour accéder à chacune de ces fonctions, la directive #**include** du préprocesseur est utilisée.

Bibliothèque d’entrées-sorties standard <stdio.h>

Pour **lire un caractère** sur un flux les fonctions suivantes sont utilisées:

```
int  getchar (void);
int  getc(FILE *stream);
int  fgetc(FILE *stream);
int  ungetc(int c, FILE *stream);
```

*FILE *stream* est un flux déjà existant; *int c* est un caractère à remettre dans le flux. Ces fonctions retournent un int avec la valeur du prochain caractère dans le flux. S'il n'y a plus de caractères, un indicateur EOF est positionné pour le flux et la fonction retourne EOF.

getchar est utilisé pour lire un unique caractère sur l'entrée standard:

```
int c;
while ((c = getchar()) != EOF) {
  <tant que non EOF, traiter le caractère>
  }
```

---

getc est utilisé pour lire un unique caractère sur un flux:

```
int c; FILE *stream;
if ((stream = fopen ("filename", "r")) != NULL) {
   while((c = getc(stream)) != EOF) {
    <traiter chaque caractère>
   }
}
else { <faire le traitement d'erreur>
}
```

---

fgetc est utilisé pour lire un unique caractère sur un flux:

```
int c;
FILE *stream;
if ((stream = fopen ("filename", "r")) != (FILE *) 0)
   {while((c = fgetc(stream)) != EOF)
     { <traiter chaque caractère> }
    }
else
{ <faire le traitement d'erreur>
}
```

Pour **écrire un caractère dans** un flux les fonctions suivantes sont utilisées:

int putchar(**int** c);
int putc(**int** c, FILE *stream);
int fputc(**int** c, FILE *stream);

*FILE *stream* est un flux déjà existant; *int c* est un caractère à écrire dans le flux. Ces fonctions retournent le caractère écrit en cas de succès. Si une erreur d'écriture intervient, l'indicateur d'erreur est positionné pour le flux et la fonction retourne EOF.

putc écrit un caractère vers le flux:

```
int c = 'x';
FILE *stream;
if ((stream = fopen ("filename", "w")) ! = NULL)
{
   putc(c, stream);
}
else {
   <traitement d'erreur>
}
```

---

putchar écrit un caractère sur la sortie standard:

---

putchar ('x')

---

fputc écrit un caractère vers le flux:

```
int c;
FILE *stream;
c='y';
if ((stream = fopen ("filename", "w")) != NULL)
{
   fputc (c, stream);
}
else
{
   <traitement d'erreur>
}
```

Les arguments de "main" / la sortie d’un programme

Un programme C peut recevoir, de la part de l'interpréteur de commandes qui a lancé son exécution, une liste d'arguments. La fonction **main** a deux paramètres appelés par convention **argc**, "argument count", et **argv**, "argument vector". **argc** indique le nombre de mots composant la ligne de commande, y compris le nom de la commande qui a servi à lancer l'exécution du programme. **argv** est un tableau de chaînes de caractères contenant chacune un mot de la ligne de commande. Le premier élément de ce tableau contenant le nom de la commande, le deuxième élément contenant le premier paramètre de la commande et ainsi de suite.

Le prototype ANSI de **main** s'écrit: **int main (int argc, char *argv []) ;**

Si un programme désire par exemple communiquer le bon ou le mauvais déroulement de son exécution à l'interpréteur de commandes il peut le faire avec les fonctions **abort** et **exit**. Pour pouvoir les utiliser, il faut inclure le fichier **stdlib.h**.

Le prototype ANSI de **abort** s'écrit: **void abort (void) ;**

Le prototype ANSI de **exit** s'écrit: **void exit (int etat) ;**

Quelque soit la fonction appelée, **abort** ou **exit**, il n'y a pas de retour dans la fonction appelante.

Interface UNIX/C (1)

À partir d'un programme C, on a diverses possibilités de faire appel aux services du système d'exploitation.

Trois manières différentes de réaliser cela sont les suivantes:

- au travers de fonctions prédéfinies,
- au travers de la fonction "system",
- en appelant directement des primitives du système.

Les bibliothèques de fonctions prédéfinies:

Il existe des *fonctions C prédéfinies* qui accèdent aux services du système d'exploitation, par exemple les fonctions d'entrée/sortie. L'avantage est qu'un programme qui utilise exclusivement les services de cette bibliothèque est portable sur diverses plateformes.

La fonction "system":

La norme ANSI définit la *fonction system*. Celle-ci permet de lancer, à partir du programme courant, l'exécution d'un autre programme. L'exécution du programme courant est temporairement suspendue et reprend lorsque le programme appelé a terminé son exécution.

Interface UNIX/C (2)

Le prototype de la fonction system est le suivant:

int system (const char *commande) ;

La commande à lancer est fournie à la fonction system sous forme d'une chaîne de caractères comme dans l'exemple ci-dessous:

system ("date") ;

Les primitives UNIX:

Une troisième façon de faire appel aux services du système hôte est d'utiliser directement des *primitives du système d'exploitation*. Exemple de programme faisant appel à la primitive UNIX **fork** (cette primitive crée dynamiquement un processus fils du processus courant):

```
void main () {
  int n;
  n = fork ();
  if (n != 0) printf ("je suis le processus père\n");
  else printf ("je suis le processus fils");
  return;
}
```

Compilateur C sous UNIX

Le compilateur C sous UNIX s'appelle **cc**. Cette commande active des programmes divers, le **préprocesseur**, le **compilateur** proprement dit, l'**assembleur** et, enfin, l'**éditeur de liens**. Par défaut, cc produit un fichier exécutable à partir de modules source, de modules prétraités, de modules assembleur ou de modules objet.

Les fichiers source sont suffixés ".c", les fichiers prétraités ".i", les fichiers assembleur ".s" et les fichiers objet ".o". Par défaut, l'exécutable produit s'appelle **a.out**. On peut influer le déroulement des opérations avec diverses options de la commande cc.

Son format général est le suivant: **cc [options] fichiers [bibliothèques]**

Les bibliothèques sont annoncées par la chaîne -l suivi du nom de la bibliothèque. Par exemple, pour lier le programme avec la bibliothèque mathématique on fournira la chaîne -lm à cc. L'éditeur de liens ira chercher la bibliothèque de nom **libm.a** dans le répertoire /usr/lib/. À l'exception des fonctions mathématiques, les autres fonctions que nous avons présentées se trouvent dans la bibliothèque standard du langage C, bibliothèque incluse par défaut par l'éditeur de lien.

Options du compilateur C sous UNIX

- **-c** supprime l'édition de liens et produit un module objet pour tout module fourni en argument.
- **-C** empêche le préprocesseur d'enlever les commentaires.
- **-D*nom*[= *chaîne*]** définit une macro nom qui est à substituer par chaîne.
- **-E** n'active que le préprocesseur, le résultat est envoyé sur le fichier standard de sortie.
- **-g** produit des informations symboliques nécessaires au débogueur dbx.
- **-I *chemin*** ajoute chemin à la liste des chemins qui seront parcourus pour retrouver les fichiers inclus avec #include
- **-o *ficher*** spécifie le nom à donner au fichier produit en remplacement du nom par défaut a.out.
- **-P** n'active que le préprocesseur et produit un module prétraité.
- **-R** n'active que le préprocesseur et le compilateur, un module assembleur est produit.
- **-w** supprime les avertissements.

Environnement de développement UNIX – outils (1)

En dehors du compilateur C, l'environnement de développement UNIX dispose des outils suivants:

lint

lint est un utilitaire qui procède à un contrôle renforcé d'un programme C. Le compilateur C étant à l'origine assez permissif, cet outil est d'un intérêt certain. En effet, il fait des contrôles renforcés au niveau des types, de la portabilité du code, de l'utilisation des objets (par exemple utilisation de variables avant leur initialisation), du déroulement du programme (par exemple détection de certaines boucles infinies), des valeurs retournées par les fonctions et de la qualité des expressions.

dbx

dbx est un débogueur symbolique et à ce titre dispose des fonctionnalités auxquelles on s'attend de la part d'un tel outil de mise au point. Il permet entre autres de définir des points d'arrêt (*break points*) dans un programme, de demander l'affichage de traces diverses durant l'exécution, de procéder à des exécutions pas à pas, de consulter les valeurs des variables pendant l'exécution du programme etc.

Environnement de développement UNIX – outils (2)

make

make est un gestionnaire de modules. Dans une application de taille réelle, on est souvent confronté à un problème de complexité dû au nombre impressionnant de modules intervenant dans celle-ci. En cas de modification des sources, par exemple, il est parfois difficile de garder une trace de toutes les opérations qui doivent être effectuées afin d'obtenir un exécutable à jour. make permet d'automatiser ceci. Dans un fichier makefile on décrit les dépendances des modules sous forme arborescente. On décrit également des règles spécifiant la manière d'obtenir un module à jour à partir des modules dont il dépend. Il suffit alors de lancer la commande make pour mettre à jour une application. C'est l'utilitaire make qui se chargera de comparer les dates de dernière mise à jour des modules pour savoir exactement quelles sont les opérations à effectuer.

Exemple d’un makefile

```
##
# Makefile automatically generated by genmake 1.0, Sep-14-99
# Modified and commented by Vladimir Makarenkov
##
# This is the C++ compiler. You could also use g++, for instance
CC= CC
#CC= g++
DEFS=
PROGNAME= ovw
INCLUDES=
# This includes the mathematical library
LIBS = -lm
# Replace -O with -g in order to debug
DEFINES= $(INCLUDES) $(DEFS) -DSYS_UNIX=1
CFLAGS= -O $(DEFINES)
SRCS = brent.c f1dim.c frprmn.c linmin.c main.c mnbrak.c myfunc.c nrutil.c
OBJS = brent.o f1dim.o frprmn.o linmin.o main.o mnbrak.o myfunc.o nrutil.o
.c.o:
	rm -f $@
	$(CC) $(CFLAGS) -c $*.c
all : $(PROGNAME)
$(PROGNAME) : $(OBJS)
	$(CC) $(CFLAGS) -o $(PROGNAME) $(OBJS) $(LIBS)
clean:
	rm -f $(OBJS) $(PROGNAME) core
```

Environnement de développement UNIX – outils (3)

cb

cb est un utilitaire qui permet de mettre en forme un programme C.

lex/yacc

lex et yacc sont des générateurs de code C. lex génère un analyseur lexicographique à partir d'expression régulières et yacc (*yet another compiler-compiler*) génère un analyseur syntaxique à partir d'une grammaire. Ces utilitaires répondent à nombre de besoins dans le domaine de la compilation. Ils sont conçus de telle façon qu'ils peuvent se compléter sans problème, l'analyseur généré par yacc peut donc utiliser l'analyseur produit par lex.

sccs

sccs, la "*source code control system*", constitue un système de gestion de base de données spécialisé. Cet utilitaire répond aux problèmes que l'on rencontre lorsqu'on a à maintenir des produits en versions multiples. Il garde une trace des modifications qu'on a effectuées sur un programme et permet de revenir en arrière à des états bien déterminés.