Langage procédural

Le langage C

Nizar OUARTI

Classe de mémorisation

- La classe de mémorisation est un élément qui influence la gestion des variables
- Lorsqu'elle n'est pas définie, elle a pour valeur implicite **auto**
- Les différentes classes de modifications sont les suivantes : **auto, static, extern, register**

auto

- C'est la classe par défaut
- Donc int a ; et auto int a ; sont équivalents
- Les mêmes règles de portées de variable que précédemment sont valables

static

- Une variable statique n'est crée qu'une seule fois, même si elle est déclaré plusieurs fois
- Une variables static est initialisée par défaut à 0.
- static int x ; // dans ce cas x=0 ;
- Cette variable conserve sa valeur au cours de l'exécution du programme.
- Si la variable est crée en locale, elle continue d'exister
- Si elle est initialisée à une valeur donnée, l'initialisation ne sera pas vue les fois suivantes

static

```c
#include <stdio.h>
main (int argc, char **argv)
{
        for (int i=0; i<5; i++)
        {
                static int a=0;// initialisation "vue"
                    que la première fois
                a++;
                printf("%d",a);
        }
        a++; // variable persistante
        printf("%d",a);
}
```

extern

- Elle sert à travailler avec une variable globale
- En général utilisée lorsque la variable a été déclaré dans un autre fichier
- Déclaration ne crée pas un nouvel objet et pas d'allocation de mémoire
- Une variables extern est initialisée par défaut à 0.
- Une variable globale est par définition extern
- Par exemple int e ; déclarée avant le main sera extern
- Par contre si on veut y accéder d'un autre fichier (i.e. l'importer) il faut écrire extern int e ; dans le fichier cible

Register

- Sert à ranger les variables dans un registre du processeur
- Sert donc à accélérer le calcul sur une variable car l'accès est plus rapide comparé à la RAM
- Si le registre est plein la variable est mise à auto
- l'opérateur d'adressage & est interdit avec les variables register
- Une variable register est intéressante lorsqu'on a souvent besoin d'y accéder

register

```c
#include <stdio.h>
#include <time.h>
main (int argc, char **argv)
{
        long start;//début timer
        long end;//fin timer
        int x;// classe auto
        register r;// classe register

        time(&start);// time fonction qui donne l'heure
        for (int x=0; x<400000000; x++);
        time(&end);
        printf("%ld",end-start);

        time(&start);
        for (int r=0; r<400000000; r++);
        time(&end);
        printf("%ld",end-start);
}
```

getchar

- Lit les données au clavier caractère par caractère
- Le caractère est lu sous la forme d'un entier correspondant au code ASCII du caractère
- int c ; c=getchar() ;

putchar

- Ecrit les données sur l'écran
- putchar(c) ;
- c peut être un caractère ou bien un nombre ASCII qui représente le caractère

getche et getch sous Windows

- Lisent directement au clavier et non un buffer d'entrée
- On a pas à faire "enter" pour avoir une action
- getche affiche le caractère et getch ne l'affiche pas

printf

- Affichage formaté : bien structurer la sortie
- Permet de contrôler l'affichage %
- c : caractère, s : string, d : entier, ld : entier long, f :nombre à virgule flottante, p :pointeur, lf : double, o :octal, x :hexadecimal, e :puissance de 10
- important de respecter l'ordre des variables avec l'ordre attendu d'affichage
- \t tabulation, \n retour à la ligne

scanf

- scanf utilise les même format que printf
- pour scanf on doit utiliser des pointeurs vers les variables pour qu'elles soient modifiables

Fichiers

- Dans un programme C il est utile de conserver des données ou d'y accéder
- Un fichier représente des données qui ne sont pas dans la RAM mais accessible sur le disque dur
- Toutefois on tentera d'éviter des accès répétés à un fichier lors de sa lecture
- Pour cette raison les buffer (ou tampon) sont utilisés

Buffer

- Le buffer est géré par le système et non par le programmeur
- Il permet d'obtenir un bloc de donné assez gros
- Cela permet en lecture ou écriture d'avoir relativement peut d'accès au périphérique (disque dur en l'occurrence)
- L'adresse du tampon est fournie dans la structure FILE (de stdio)
- FILE contient beaucoup d'information sur le fichier
- informations : pointeur vers le buffer, pointeur caractère suivant dans le buffer, nombre de caractère dans le buffer, descripteur du fichier,...
- Déclaration FILE *fp ;

fopen

- On utilise fopen pour ouvrir un programme
- fopen retourne un pointeur vers FILE
- fopen à différents mode d'accès : "r" :read, "w" : write, "a" : append, "r+" : lecture écriture (sans création), "w+" : lecture écriture (avec création)
- il existe aussi une possibilité d'accéder aux fichiers en binaire ex rb, wb+ (utilisation fread et fwrite)

fopen

```
1  #include <stdio.h>
2  main (int argc, char **argv)
3  {
4          File *fp;
5          fp=fopen("toto.dat", "r");
6          if (fp==NULL)
7          {
8                  printf("Erreur le fichier n'existe pas.
                       ");
9          }
10 }
```

fclose

- Les fichier sont fermés à la fin du programme
- Toutefois,on ne peut manipuler qu'un nombre fini de fichiers
- D'autre part les tampon à moitié pleins en écriture ne sont vidés que lorsque l'on ferme les fichiers
- Il est conseillé de fermer les fichiers dès que l'on ne les utilisent plus
- fclose sert à cela : fclose(fp)

Les Pointeurs FILE prédéfinis

- On a déjà dit que **stdin** est un pointeur FILE prédéfini qui lit le clavier (entrée standard)
- **stdout** écrit sur l'écran (sortie standard)
- **stderr** écrit les erreurs à l'écran
- **stdaux** écrit sur le port série
- **stdprn** écrit sur l'imprimante

Rappel redirection entrée et sortie standard

- Pour l'instant on avait défini stdin et stdout au clavier et à l'écran
- Nous avons vu pourtant que l'entrée et la sortie standard d'un programme pouvait être modifié
- progr1 < fichier1.txt (sous la console) cela envoie en entrée standard le fichier1.txt
- progr2 > fichier2.txt (sous la console) cela écrit un fichier2.txt grâce à la sortie standard
- progr1 | progr2 est aussi possible

Lecture et écriture formatée

- **fprintf** et **fscanf**
- Très pratique pour lire ou écrire sous un format en tableau (base de donnée ou excell)
- Chaque ligne représente une instance chaque colonne une variable
- fprintf(fp,"",var1,var2,...)
- fscanf(fp,"",&var1,&var2,...)
- Même usage que printf et scanf sauf qu'on ajoute le pointeur vers FILE

Problème

- Soit un étudiant en alternance
- Créer une structure étudiant possédant son numéro, sa moyenne, sa formation d'origine, son entreprise
- Créer un tableau d'étudiant
- écrire un fichier pouvant écrire (en ajout) ou lire les information sur un étudiant