Programmation

Chapitre1 : Rappel en C

SOMMAIRE

Rappel langage C

I) Les pointeurs

Déclaration :

```
Int * y ;
Char *c ;
Long *b ;
```

Soit int * p ;
Et int x ;
Si p=&x

Alors **Y = *P+1** ⇔ **Y = X+1**
***P = *P+10** ⇔ **X = X+10**
***P += 2** ⇔ **X += 2**
++*P ⇔ **++X**
(*P)++ ⇔ **X++**

Notation :

Après les instructions:

```
int A;
int *P;
P = &A;
```

A désigne le contenu de A
&A désigne l'adresse de A

P désigne l'adresse de A
***P** désigne le contenu de A

En outre:
&P désigne l'adresse du pointeur P
***A** est illégal (puisque A n'est pas un pointeur)

Pointeurs et tableaux :

En déclarant un tableau A de type **int** et un pointeur P sur **int**,

```
int A[10];
int *P;
```

l'instruction:
P = A; est équivalente à **P = &A[0];**

Ainsi, après l'instruction,

```
P = A;
```

le pointeur P pointe sur A[0], et

***(P+1)** désigne le contenu de A[1]

***(P+2)** désigne le contenu de A[2]

... **...**

***(P+i)** désigne le contenu de A[i]

Allocation dynamique :

Exemple: la fonction **malloc**

```
char *pc;
int *pi,*pj,*pk;
pc = (char*)malloc(10);          /* on réserve 10 cases mémoire, soit la
place pour 10 caractères */
pi = (int*)malloc(16);           /* on réserve 16 cases mémoire, soit la
place pour 4 entiers  */
 pj = (int*)malloc(sizeof(int)); /* on réserve la taille d'un entier en mémoire
*/
pk = (int*)malloc(3*sizeof(int));      /* on réserve la place en mémoire
pour 3 entiers */
```

Libération de la mémoire: la fonction **free**

```
free(pi);                /* on libère la place précédemment réservée pour i */
free(pr);                /* on libère la place précédemment réservée pour r */
```

Exercices à préparer sur feuille pour le lundi 09 septembre

Exercice1 :
Soit le code suivant :

```
Int main()
{
Int * ptr ;
*ptr=5 ;
printf(« %d »,ptr) ;
}
```

Que donne ce code à l'exécution ?

Exercice 2 :

1. En Langage C, déclarez une variable « varfloat » de type float initialisée, à la déclaration, avec la valeur 1.7,
2. Affichez l'adresse de varFloat,
3. Déclarez une variable permettant de stocker l'adresse de la variable varFloat,
4. Dessinez la configuration de la mémoire,
5. Affichez le contenu de la zone pointée par le pointeur, en utilisant le pointeur ;

Exercice 3 :
Modifiez ces deux programmes pour n'afficher qu'une valeur sur deux du tableau

```
1 int main(void)
2 {
3 int tabint[15] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14} ;
4 int *tabptr = tabint ;
5 int i ;
6 for (i=0 ; i<15 ; i++)
7 {
8 printf(« %d  »,*(tabptr+i) ) ;
9 }
10 }
```

```
int main(void)
2 {
3 int tabint[15] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14} ;
4 int *tabptr = tabint ;
5 int i ;
6 for (i=0 ; i<15 ; i++)
7 {
8 printf(« %d »,*(tabptr) ) ;
9 tabptr++;
10 }
11 }
```

II) Les fonctions

Exemple :

```
Float  pow(float x, float y);
int atoi(const char *s);

Main()
{
float x=2, y=3 ;
int conv ;

float res=pow(x,y) ;

conv=atoi(« 56 ») ;

}
```

Prototype d'une fonction ?

Type-retour nom-fonction (type-param1,…,…)

Exercices à préparer sur feuille pour le vendredi 9 septembre

Exercice :

Créez un programme en C qui gère un tableau de 20 entiers au max et utilise 3 fonctions :

- une fonction qui empile des valeurs dans le tableau (LIFO Last In First Out)
- Une fonction qui dépile le tableau cad que l'on récupérera la dernière valeur entrée.
- Une fonction affiche qui affiche tout le contenu du tableau.

1° Etablir le prototype de vos fonctions
2° Définissez vos fonctions
3°Créez un programme qui appelle ces différentes fonctions

III) Les structures

Déclaration :

Typedef struct {	ou	struct fiche {
} fiche ;		};
fiche a ;		struct fiche a ;

Exemple :

```
typedef struct          /* On définit un type struct */
{
char nom[10];
char prenom[10];
int age;
float note;
}
fiche;
```

On déclare des variables par exemple: **fiche f1,f2;**

puis, par exemple:
```
strcpy(f1.nom,"DUPONT");
strcpy(f1.prenom,"JEAN");
f1.age = 20;
f1.note = 11.5;
```

structures et tableaux :

Exemple: (à partir de la structure définie précédemment)

Déclaration: **fiche f[10]; /* on déclare un tableau de 10 fiches */**

Utilisation:
```
strcpy(f[i].nom,"DUPONT") /* pour un indice i quelconque */
strcpy(f[i].prenom,"JEAN");
f[i].age = 20;
f[i].note = 11.5;
```

structures et pointeurs

Un symbole spécial a été créé pour les pointeurs de structures, il s'agit du symbole **->**

Exemple: (à partir de la structure définie précédemment)

Déclaration: **fiche *f; /* on déclare un pointeur de fiche */**

Utilisation:
```
f = (fiche*)malloc(sizeof(fiche)); /* réserve de la place */
strcpy(f->nom,"DUPONT");
strcpy(f->prenom,"JEAN");
f->age = 20;
f->note = 11.5;
```

Exercices à préparer sur feuille pour le jeudi 12 septembre

Définir une structure de données Heure permettant de représenter une heure au format hh/mm/ss, puis écrire les fonctions suivantes :
1) Ecrire une fonction de conversion d'un élément de type Heure en un nombre de secondes (entier)
2) Ecrire une fonction de conversion d'un nombre de secondes (entier) en un élément de type Heure
3) Ecrire une fonction qui retourne un temps correspondant à l'addition de deux éléments de type Heure
4) Créer un programme permettant de saisir un temps en H/M/S puis convertir ce temps en nombre de secondes.
Ensuite saisir un nombre de secondes et le convertir en temps au format H/M/S . Puis ajoutez les deux temps précédemment saisis et affichez le.

IV) Les fichiers

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

On définit un pointeur sur FILE. Ce pointeur fournit l'adresse d'une cellule donnée (l'adresse du tampon).

2 - Ouverture: **FILE *fopen(char *nom, char *mode);**

On passe donc 2 chaînes de caractères
nom: celui figurant sur le disque, exemple: « a :\toto.txt»

mode (pour les fichiers TEXTES) :

« r » lecture seule
« w » écriture seule (destruction de l'ancienne version si elle existe)
« w+ » lecture/écriture (destruction 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.

A l'ouverture, le pointeur est positionné au début du fichier (sauf « a+ »)

Exemple :
```
FILE *fichier ;
fichier = fopen(« a :\toto.txt », « r+ ») ;
```

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.txt », « r+ ») ;
/* Ici instructions de traitement */
fclose(fichier) ;
```

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

5- Ecriture dans le fichier:

int putc(char c, FILE *fichier);
Ecrit 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) ;**

int putw(int n, FILE *fichier);

Idem, n de type int, le pointeur avance du nombre de cases correspondant à la taille d'un entier (4 cases en C standard).
Retourne n si l'écriture s'est bien passée.

int fputs(char *chaîne, FILE *fichier); idem avec une chaîne de caractères, le pointeur avance de la longueur de la chaine ('\0' n'est pas rangé dans le fichier).
Retourne EOF en cas d'erreur.

Exemple : **fputs(« BONJOUR ! », fichier) ;**

int fprintf(FILE *fichier, char *format, liste d'expressions);
Retourne EOF en cas d'erreur.

Exemples: **fprintf(fichier,"%s","il fait beau");**
fprintf(fichier,%d,n);
fprintf(fichier,"%s%d","il fait beau",n);

Le pointeur avance d'autant.

6 - Lecture du fichier:

int getc(FILE *fichier); lit 1 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) ;

int getw(FILE *fichier); idem avec un entier; le pointeur avance de la taille d'un entier.
Exemple: **int n ;**
n = getw(fichier) ;

char *fgets(char *chaine,int n,FILE *fichier); lit n-1 caractères à partir de la position du pointeur et les range dans chaine en ajoutant '\0'.

int fscanf(FILE *fichier, char *format, liste d'adresses); analogue à fprintf en lecture.
Exemple : fscanf(fichier, « %d »,&nb) ;

7 - Gestion des erreurs:

fopen retourne le pointeur NULL si erreur (Exemple: impossibilité d'ouvrir le fichier).

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

la fonction **int feof(FILE *fichier)** retourne 0 tant que la fin du fichier n'est pas atteinte.
8 – Les entrées/sorties binaires

Les fonctions d'entrées-sorties binaires permettent de transférer des données dans un fichier sans transcodage. Elles sont donc plus efficaces que les fonctions

d'entrée-sortie standard, mais les fichiers produits ne sont pas portables puisque le codage des données dépend des machines.

Elles sont notamment utiles pour manipuler des données de grande taille ou ayant un type composé. Leurs prototypes sont :

size_t fread(void *pointeur, size_t taille, size_t nombre, FILE *flot);
size_t fwrite(void *pointeur, size_t taille, size_t nombre, FILE *flot);

où *pointeur* est l'adresse du début des données à transférer, *taille* la taille des objets à transférer, *nombre* leur nombre. Rappelons que le type size_t, défini dans stddef.h, correspond au type du résultat de l'évaluation de sizeof. Il s'agit du plus grand type entier non signé.

La fonction fread lit les données sur le flot *flot* et la fonction fwrite les écrit. Elles retournent toutes deux le nombre de données transférées.

Par exemple, le programme suivant écrit un tableau d'entiers (contenant les 50 premiers entiers) avec fwrite dans le fichier sortie, puis lit ce fichier avec fread et imprime les éléments du tableau.

```
#include <stdio.h>
#include <stdlib.h>

#define NB 50
#define F_SORTIE "sortie"

int main(void)
{
  FILE *f_in, *f_out;
  int *tab1, *tab2;
  int i;

  tab1 = (int*)malloc(NB * sizeof(int));
  tab2 = (int*)malloc(NB * sizeof(int));
  for (i = 0 ; i < NB; i++)
    tab1[i] = i;

  /* ecriture du tableau dans F_SORTIE */
  if ((f_out = fopen(F_SORTIE, "w")) == NULL)
    {
      fprintf(stderr, "\nImpossible d'ecrire dans le fichier %s\n",F_SORTIE);
      return(EXIT_FAILURE);
    }
  fwrite(tab1, NB * sizeof(int), 1, f_out);
  fclose(f_out);

  /* lecture dans F_SORTIE */
  if ((f_in = fopen(F_SORTIE, "r")) == NULL)
    {
      fprintf(stderr, "\nImpossible de lire dans le fichier %s\n",F_SORTIE);
      return(EXIT_FAILURE);
    }
```

```
  fread(tab2, NB * sizeof(int), 1, f_in);
  fclose(f_in);
  for (i = 0 ; i < NB; i++)
    printf("%d\t",tab2[i]);
  printf("\n");
  return(EXIT_SUCCESS);
}
```

Les éléments du tableau sont bien affichés à l'écran. Par contre, on constate que le contenu du fichier sortie n'est pas encodé.

9 – Les fichiers ini

Les fichiers INI sont des fichiers texte : ils peuvent être manipulés avec un logiciel courant de type éditeur de texte.

Les fichiers sont divisés en sections. Chaque section comporte un certain nombre de paramètres de configuration. Chaque section commence par un titre placé entre crochets « [» et «] ».

La valeur de chaque paramètre de configuration est indiquée par une formule : paramètre = valeur.

Les fichiers peuvent contenir des commentaires. Les commentaires sont souvent utilisés pour décrire les paramètres et les valeurs à introduire. Ils sont précédés d'un point-virgule ou plus rarement d'un dièse.

exemple.ini

```
[serveur pedago]
adresse=192.168.64.20
port=5002

[serveur web]
adresse=192.168.64.52
port=80
```

ANNEXES

Les types de données en langage C:

Type de donnée	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	1	-128 à 127
unsigned char	Caractère non signé	1	0 à 255
short int	Entier court	2	-32768 à 32767
unsigned short int	Entier court non signé	2	0 à 65535
int	Entier	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	-32768 à 32767 -2147483647 à 2147483647
unsigned int	Entier non signé	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	0 à 65535 0 à 4294967295
long int	Entier long	4	-2 147 483 648 à 2 147 483 647
unsigned long int	Entier long non signé	2	0 à 4 294 967 295
float	flottant (réel)	4	$3.4*10^{-38}$ à $3.4*10^{38}$
double	flottant double	8	$1.7*10^{-308}$ à $1.7*10^{308}$
long double	flottant double long	10	$3.4*10^{-4932}$ à $3.4*10^{4932}$

Nombre entier (*int*)

Un nombre entier est un nombre sans virgule qui peut être exprimé dans différentes bases:

- Base décimale: L'entier est représenté par une suite de chiffres unitaires (de 0 à 9) ne devant pas commencer par le chiffre 0
- Base hexadécimale: L'entier est représenté par une suite d'unités (de 0 à 9 ou de A à F (ou a à f)) devant commencer par *0x* ou *0X*
- Base octale: L'entier est représenté par une suite d'unités (incluant uniquement des chiffres de 0 à 7) devant commencer par *0*

Les entiers sont signés par défaut, cela signifie qu'ils comportent un signe.

Nombre à virgule (*float*)

Un nombre à virgule flottante est un nombre à virgule, il peut toutefois être représenté de différentes façons:

- un entier décimal: 895
- un nombre comportant un point (et non une virgule): 845.32
- une fraction: 27/11
- un nombre exponentiel, c'est-à-dire un nombre (éventuellement à virgule) suivi de la lettre *e* (ou *E*), puis d'un entier correspondant à la puissance de 10 (signé ou non, c'est-à-dire précédé d'un + ou d'un -)
  2.75e-2
  35.8E+10
  .25e-2

En réalité, les nombres réels sont des nombres à virgule flottante, c'est-à-dire un nombre dans lequel la position de la virgule n'est pas fixe.

Les nombres de type **float** sont codés sur 32 bits dont:

- 23 bits pour la mantisse
- 8 bits pour l'exposant
- 1 bit pour le signe

Les nombres de type **double** sont codés sur 64 bits dont:

- 52 bits pour la mantisse
- 11 bits pour l'exposant
- 1 bit pour le signe

Les nombres de type **long double** sont codés sur 80 bits dont:

- 64 bits pour la mantisse
- 15 bits pour l'exposant
- 1 bit pour le signe

La précision des nombres réels est approchée. Elle dépend par le nombre de positions décimales, suivant le type de réel elle sera au moins:

- de 6 chiffres pour le type **float**
- de 15 chiffres pour le type **double**
- de 17 chiffres pour le type **long double**

Caractère (*char*)

Le type **char** (provenant de l'anglais *character*) permet de stocker la valeur ASCII d'un caractère, c'est-à-dire un nombre entier!

Par défaut les nombres sont signés, cela signifie qu'ils comportent un signe.

TP n°1 - feuille 1/2	
Classe	BTS 2ème année
Préparation Durée TP	Aucune 6h00
Groupe	1 étudiant

OBJECTIFS DU TP : Rappel sur notions de tri, de fonctions, de pointeurs et de structures.
PREREQUIS : Toutes les notions données au dessus.

Rappel
TP numéro 1

Exercice n°1 :

Objectif :
Vous écrirez une fonction échange qui échange les valeurs des deux variables entières qui lui sont transmises en argument.
Contraintes :
Ces deux variables, déclarées dans le main, seront affichées avant et après l'appel à la fonction échange dans le programme principal.

Exercice n°2 :

Objectif : Soit un tableau tab, qui contient au max 100 entiers rangés dans l'ordre croissant et qui sera initialisé par l'utilisateur, vous réaliserez un programme qui recherche une valeur (donnée par l'utilisateur) dans ce tableau.
Contraintes :
La méthode de recherche par dichotomie vous est imposée.
Si la valeur est trouvée vous affichez sa position dans le tableau sinon vous affichez un message d'erreur.
Vous devrez créer trois fonctions :

- Une pour l'initialisation (qui insérera les valeurs que donnent l'utilisateur de manière à ce que le tableau soit rangé dans l'ordre croissant),
- Une pour la recherche (qui recherchera la valeur donnée par l'utilisateur dans le tableau en utilisant la méthode de recherche par dichotomie),
- Une pour l'affichage (qui affiche le tableau et si la valeur donnée par l'utilisateur est présente ou non dans ce dernier et si oui donne sa position).

Exercice n°3 :

Soit la structure nommée « student » ayant 4 membres représentant respectivement le nom, le prenom, l'adresse et la classe de l'élève.
L'application qui sera développée permettra de gérer au max 100 élèves.
Vous devrez utiliser pour ce TP les fonctions fread et fwrite.

Question n° 0

Réaliser le diagramme des cas d'utilisations de ce programme.

Question n° 1

Réaliser en C le programme répondant aux exigences fonctionnelles suivantes (par ordre de priorité).

Exigence fonctionnelle n° 1

Les fonctions du programme sont articulées autour d'un menu.

Exigence fonctionnelle n° 2

Une fonction « addStudent » permet de saisir les informations d'**un élève**. Cette fonction recevra l'adresse de la structure qui est à remplir, le nombre d'élève déjà gérés et retournera le nombre total d'élèves enregistrés après la saisie.

Exigence fonctionnelle n° 3

Une fonction « displayStudent » permet d'afficher les informations de tous les élèves enregistrés.

Exigence fonctionnelle n° 4

Une fonction « saveStudent » permet de sauvegarder toutes les structures dans un fichier dont le nom et le chemin d'accès seront donnés par l'utilisateur dans le programme principal et transmis à cette fonction.

Exigence fonctionnelle n° 5

Une fonction « loadStudent » permet de charger toutes les informations du fichier (dont le nom et le chemin d'accès seront soit donnés par l'utilisateur soit stockés dans un fichier ini au choix) dans un tableau de structures .