

4ILPR-1-A Informatique: Langage procédural

Olivier Kesteloot

January 5, 2015

Sommaire

1	Introduction	7
2	Déclarer les variables	9
2.1	Identification	9
2.2	Déclaration	9
2.2.1	Entier	9
2.2.2	Réels	9
2.3	Initialisation	10
2.4	Constantes	10
2.4.1	Entières	10
2.4.2	Réelles	10
2.4.3	Caractères	10
2.4.4	Entier octal/hexadécimal	10
3	Printf	11
3.1	Directives \	11
3.2	Directives %	11
3.3	Taille	12
3.3.1	Entier	12
3.3.2	0entier	12
3.3.3	-entier	12
3.3.4	Réels	12
4	Opérateurs	13
4.1	Arithmétique	13
4.2	Relationnel	13
4.3	Affectation	13
4.4	Logique	13
4.5	Logique binaire	14
4.6	Décalage	14
4.7	Incrément et décréement	14
4.8	Affectation	14
4.9	Adresse	15
4.10	Conditionnel	15
4.11	Sizeof	15
4.12	Virgule	15
4.13	Changement de type	15
5	Structures de contrôle	17
5.1	if(cond)	17
5.2	if(cond) else	17
5.3	if(cond) else if(cond2) else if(cond3) else	17
5.4	while(cond)	17
5.4.1	break	17
5.4.2	continue	18

5.5	do.....while(cond)	18
5.6	for(initialisation; condition; évolution)	18
5.7	switch() case	18
6	scanf("chaîne",&variables);	19
6.1	Directives %	19
6.2	Quelques fonctions supplémentaires	20
6.2.1	getche()	20
6.2.2	putchar()	20
6.3	Caractères importants du code ASCII	20
7	Fonction	21
7.0.1	Type des fonctions	22
8	Types, classes et formes	23
8.1	Classe d'une variable	23
8.1.1	Les variables static externe	24
8.1.2	La classe registre	24
9	Les vecteurs	25
9.1	Déclaration	25
9.2	Éléments d'un vecteur	25
9.3	Initialisation	25
9.3.1	Vecteur auto	25
9.3.2	Vecteur static	25
9.4	Vecteurs et fonctions	26
10	Pointeurs	27
10.1	Déclaration	27
10.2	Adresse	27
10.3	Adressage indirect	27
10.4	Pointeurs et fonctions	28
10.5	Les pointeurs, les vecteurs et les fonctions	28
10.5.1	Adresse et Tableaux	28
10.5.2	Tableaux et fonctions	28
10.5.3	Calcul sur pointeurs	28
11	Les chaînes de caractères	31
11.1	Utilisation	31
11.2	Chaînes et fonctions	31
11.3	La directive %s	31
12	argc, argv et redirections	33
12.1	argc, argv	33
12.2	exit()	34
13	Les fichiers	35
13.1	Ouvrir	35
13.2	Fichiers textes	35
13.2.1	Caractère par caractère	35
13.2.2	Mot par mot	36
13.2.3	Ligne par ligne	36
13.3	Pointeurs de fichier: stdin, stdout, stderr	36
13.3.1	Normal (défaut)	36
13.3.2	Redirection	36
13.3.3	Travail par blocs	37
13.3.4	Gestion des erreurs	37
13.4	Se positionner dans un fichier	38

13.4.1	ftell	38
13.4.2	fseek	38
14	La récursivité	39
15	Les tables à plusieurs dimensions	41
15.1	Tables à deux dimensions	41
15.1.1	Déclaration et initialisation	41
15.1.2	Éléments et adresses	41
15.1.3	Fonctions	41
15.2	Tables à trois dimensions	41
15.2.1	Déclaration et initialisation	41
15.2.2	Éléments et adresses	42
15.2.3	Fonctions	42
15.3	Tables à ? dimensions	42
15.3.1	Déclaration et initialisation	42
15.3.2	Fonctions	42
16	Les pointeurs de pointeurs	43
16.1	Tri selon les vecteurs de pointeurs	43
16.2	Pointeurs de pointeurs	43
17	Structures et unions	45
17.1	Les structures et les fonctions	45
17.2	Vecteur de structures	46
17.3	Structure de structure	46
17.4	Unions	47
18	Gestion dynamique de la mémoire	49
18.1	malloc	49
18.2	calloc	49
18.3	realloc	49
18.4	free	50
Annexes		50
A	Exercices	51
A.1	Structures de contrôle	51
A.1.1	Minimum de 3 entiers	51
A.1.2	Résolution d'une équation du second degré	51
A.1.3	Tableau ASCII	52
A.2	Scanf	52
A.2.1	Nombre binaire	52
A.3	Les fonctions	53
A.3.1	Nombre mystère (résolution dichotomique)	53
A.3.2	Pyramide	54
A.3.3	Nombre de Armstrong	54
A.3.4	Addition et soustraction d'entiers	56
A.4	Les vecteurs	57
A.4.1	Calculatrice polonaise	57
A.4.2	Permutation d'un vecteur d'entiers	59
A.5	Les pointeurs	60
A.5.1	Echange de deux entiers à l'aide de pointeurs	60
A.5.2	Permutation d'un vecteur d'entiers : variante 1	60
A.5.3	Permutation d'un vecteur d'entiers : variante 2	62
A.5.4	Permutation d'un vecteur d'entiers : variante 3	63
A.6	Les chaînes de caractères	64
A.6.1	Fonction string	64

A.6.2	Fonction string copy	65
A.6.3	Fonction string cat	65
A.6.4	Fonction string length	65
A.6.5	Fonction string compare	65
A.6.6	Recherche de caractère	66
A.7	argc et argv	66
A.7.1	Pyramide d'étoiles	66
A.8	Les fichiers	66
A.8.1	Réécrire la commande type du DOS	66
A.8.2	Réécrire la commande copy du DOS	67
A.8.3	Compter le nombre de mots dans un fichier	67
A.8.4	Compter le nombre de lignes dans un fichier	67
A.8.5	Compter les caractères affichés par bloc de 512	68
A.8.6	Copier fichier	68
A.8.7	Copier avec tous les messages d'erreur	69
A.8.8	Copier fichier dans racine du disque autrepart	69
A.9	La récursivité	70
A.9.1	Ecrire la fonction factorielle en récursivité	70
A.9.2	Pgcd	70
A.9.3	Pgcd en utilisant d'autres formules	70
A.9.4	Fibonnaci	71
A.9.5	Combinaison	71
A.10	Les tables à plusieurs dimensions	72
A.10.1	Remplir une table	72
A.10.2	Tri à bulles 1	72
A.10.3	Tri à bulles 2	73
A.10.4	Tri à bulles 3	74
A.10.5	Tri à bulles 4	74
A.10.6	Tri à bulles 5	75
A.11	Structures et unions	76
A.11.1	Chèque 1	76
A.11.2	Chèque 2	77
A.11.3	Chèque 3	78
A.11.4	Unions	80
A.12	Gestion dynamique de la mémoire	80
A.12.1	Exercice 1	80
A.12.2	Exercice2	81
A.13	Les listes chaînées	81
A.13.1	Exercice 1	81
A.13.2	Exercice 2	81
A.13.3	Exercice 3	81
B	Priorité et associativité des opérateurs	83
C	Résumé sur l'initialisation	85
C.1	Paramètres	85
C.2	Variables	85
C.2.1	auto ou register	85
C.2.2	static	85

Chapter 1

Introduction

Exemple de programme en C:

```
1  /*
2  */
3  main()
4  {
5      printf("\nOn est dans main");
6      fonct();
7      printf("\nRetour a main");
8  }
9  fonct()
10 {
11     printf("\nEt maintenant dans fonct");
12 }
```

Le C est composé de fonctions disjointes et d'une seule fonction main. Dans le C, tout est fonction (ex: printf). On peut ajouter des bibliothèques contenant des fonctions:

```
1  #include "stdio.h"      /*#include "nom_de_la_bibliotheque*/
2                          /*stdio signifie standard input output*/
```

En C, tout doit être écrit en minuscule et les espaces ne sont utilisés que par clarté.
Pour écrire des commentaires:

```
1  /*ouverture des commentaires ==> fermeture des commentaires */
```


Chapter 2

Déclarer les variables

2.1 Identification

Les variables sont composées de lettres, de chiffres ou d'underscore "_". Attention, elles ne peuvent pas commencer par un chiffre et il est fortement déconseillé de commencer par un "_" (mis à part pour les variables système)

Conventions: i, j, n pour des entiers et x, y, z pour des réels.

2.2 Déclaration

syntaxe: type nom;

2.2.1 Entier

- char byte -128 à 127
- short 2 bytes -32768 à 32767
- long 4 bytes $-2^{31} \Rightarrow 2^{31} + 1$
- int 2 ou 4 bytes (selon les compilateurs)

Pour avoir les non-signés, on ajoute "unsigned" devant le type.

Attention, il n'existe pas de variables booléennes en C.

exemples:
unsigned char 0 à 255
unsigned short 0 à 65535

2.2.2 Réels

- float
- double

```
1 main()
2 {
3     short i, j;
4     float x;
5
6     i=3;
7     j=i+1;
```

```

8  x=3.14;
9  printf("j=%d",j);    /*le symbole % sera explique dans le chapitre printf (directive)*/
10 }

```

2.3 Initialisation

On peut déclarer une variable avec une valeur initiale:

```

1  short i,j=0;          /*attention, il n'y a 0 que dans j. Dans i, il y a n'importe quoi*/
2  short i=0, j=0;       /*les deux variables sont initialisees a 0*/

```

2.4 Constantes

2.4.1 Entières

int 2 -4 2L (pour avoir des "longs")

2.4.2 Réelles

double -1.3

2.4.3 Caractères

```

1  ex:  char c;
2        c='A';          /*c'est la meme chose que d'ecrire c=65*/
3        c=c+1;          /*'B' (code ASCII)*/

```

2.4.4 Entier octal/hexadécimal

"012" en octal équivaut à $1 * 8^1 + 2 * 8^0 = 10$ (en décimal)

Le zéro devant les chiffres indique que c'est de l'octal.

```

1  ex:  int n=10;          /*identique a int n=012;*/

```

"0x12" en hexa équivaut à $1 * 16^1 + 2 * 16^0 = 18$

```

1  int n=18;              /*identique a n=0x12;*/

```

Le "0x" devant les chiffres indique que c'est de l'hexadécimal.

```

1  int n=18;              /*identique int n=0x12;*/

```

Chapter 3

Printf

La fonction `printf` se trouve dans la bibliothèque `stdio.h`. Le "f" signifie formaté. Elle permet de faire des prints formatés selon un certain format. Syntaxe: `printf("chaîne", val1, val2, ...);`

Dans la chaîne, tout est reproduit sauf les directives

3.1 Directives \

- `\n` retour à la ligne
- `\r` retour chariot
- `\t` tabulation
- `\b` backspace
- `\f` nouvelle page
- `\o` zéro
- `\\` forcer pour afficher \
- `\'` forcer pour afficher '
- `\"` forcer pour afficher "

3.2 Directives %

Attention, il faut mettre une directive % pour chaque valeur et une valeur pour chaque directive.

- `%c` pour afficher un caractère
- `%d` pour afficher un entier en décimal signé
- `%u` pour afficher un entier en décimal non-signé
- `%f` pour afficher un réel en notation virgule flottante
- `%e` pour afficher un réel en notation exponentiel
- `%s` voir plus bas (chapitre spécifique sur cette directive)
- `%o` pour afficher un entier en octal
- `%x` pour afficher un entier en hexadécimal
- `%%` pour afficher des %

```

1 char c;
2 short n=12;
3 float x=3.1416;
4 c='A';
5
6 printf("Hello");
7 printf("_a_tous\n\n");          /* Hello a tous
8
9                                */
10
11 printf("n=%d",n);              /*n=12*/
12 printf("\n%d_et_%d_et_%d",n); /*erreur une seule valeur pour 3 directives*/
13 printf("\nc=%d_et_c=%c", c, c); /*c=65 et c=A*/
14 printf("\ncaract=%c_et_%c_Suivant=%c", 67, c, c+1); /*caract=C et A. Suivant=B*/

```

3.3 Taille

3.3.1 Entier

```
1 printf("\%5d",n);          /* 12*/
```

3.3.2 0entier

Pour afficher des "0" à la place des espaces.

```
1 printf("\%05d",n);        /*00012*/
```

3.3.3 -entier

Pour cadrer à gauche.

```
1 printf("\%-5d",,);        /*12 */
```

3.3.4 Réels

On peut donner une taille totale et une taille pour les décimaux.

```

1 printf("\%7.2f",x);        /* 3.14*/
2 printf("\%.2f",x);         /*3.14*/

```

Si la décimal est trop petite, il tronque (il n'arrondit jamais).

```
1 printf("\%03.2f",x);       /*003.14*/
```

Attention, pour la partie entière, il ne tronque jamais.

Chapter 4

Opérateurs

4.1 Arithmétique

()
/ %
+ -

Pour l'exposant, il faut des fonctions.

exemples:

$7/2 = 3$

$7.0/2 = 3.5$

$7\%3 = 1$

Attention, pour le modulo, les opérandes doivent être entières.

4.2 Relationnel

< <=
> >=
== !=

Tout est vrai sauf 0 (c'est une logique semi-booléennes)

4.3 Affectation

=

```
1  z=y=x=3;  
2  if(x=3)           /* toujours vrai car il affecte 3 dans x */  
3  
4  if(x==3)          /* teste d'abord la valeur de x */
```

4.4 Logique

&& || ! deviennent respectivement et, ou, négation

Ce sont des opérateurs cour-circuits: dès qu'il connaît, il arrête d'évaluer.

```
1  if((d!=0) && (n/d==3)) /* si d!=0, il continue avec (n/d==3) */  
2                          /* si d==0, il s'arrete pour eviter une division par 0 */  
3  
4  if((l==1) || ((c=getche())!='\n'))  
5      /* si l=1, il s'arrete */  
6      /* si l!=1, il continue */  
7  
8  /* Attention: ((c=getche())!='\n') est different de (c=getche()!='\n') */
```

```
9 /*Dans le deuxieme, = est dernier au niveau des priorites*/
```

4.5 Logique binaire

& | ~ ^

deviennent respectivement et, ou, négation, ou exclusif.

Attention, les opérandes doivent être entières.

```
1 int n=11;
2 n=n & 6;
3 /*n=11 devient 00001011
4 6      devient 00000110
5 -----
6      devient 00000010 = 2 (en notation decimale)*/
```

4.6 Décalage

<< >>

Opérandes doivent être entières

```
1 int n<6;
2 n=n<<3;      /*n=6      devient 0000 0110
3              <<3      devient 0011 0000 = 48*/
```

4.7 Incrément et décrémente

++ --

```
1 short n,m=3;
2 m++;      /*m=m+1*/
3 ++m;      /*m=5*/
```

Si on met le signe à gauche, on pré-incrémente.

Si on met le signe à droite, on post-incrémente.

```
1 n=m++      /*n=3      m=4*/
2 n=++m;      /*m=4      n=4*/
3
4 short n=4,m=4;
5 printf("%d",--n);      /*n=3 puis affiche 3*/
6 printf("%d",m--);      /*affiche 4 puis m=3*/
7
8 if(n--)      /*il teste PUIS decremente*/
```

4.8 Affectation

+= -= *= /= %=

```
1 n=n+5;      /*est identique a n+=5;*/
```

&= |= <<= >>= ÷=

```
1 n=n&6;      /*est identique a n&=6;*/
```

4.9 Adresse

&

&n signifie adresse de la variable n à mettre dans un pointeur.

4.10 Conditionnel

? :

syntaxe: condition?val1:val2

Le tout est une valeur.

```
1 short a,b, min;
2 min=a>b?b:a;
3 printf("\nValeur absolue de a=%d", a>0?a:-a);
```

4.11 Sizeof

Donne la taille en bytes d'une variable ou d'un type.

```
1 short m;
2 printf("%d_%d", sizeof(long), sizeof(n)); /*4 2*/
```

4.12 Virgule

,

Sert à relier des instructions.

```
1 short a, b,tmp;
2 tmp=a, a=b, b=tmp;
```

4.13 Changement de type

On met le type voulu entre ()

```
1 short n=7, m=2;
2 float f;
3 f=n/m; /* f=3.0 */
4 f=(float)n/m; /* f=3.5 */
```


Chapter 5

Structures de contrôle

5.1 if(cond)

S'il y a plusieurs instructions, on met des accolades{ }.

```
1  if(a>1)
2      a=7;          /* si condition vrai */
3      b=1;          /* cette instruction se fera meme si la condition est fausse */
```

```
1  if(a>1)
2  {
3      a=7;          /* les instructions ne se font que si la condition est vraie */
4      b=1;
5  }
```

5.2 if(cond) else

Remarque: quand les if sont imbriqués, on conseille de mettre des accolades dans les if.

```
1  if(a>0)
2      if(a%2==0)
3          printf("\nPositif et pair");
4  else
5      printf("\nNegatif");          /* le else se rapporte au 2eme if */
```

5.3 if(cond) else if(cond2) else if(cond3) else

```
1  /* afficher si un nombre est positif, negatif ou nul */
2  if(a>0)
3      printf("Positif");
4  else if(a<0)
5      printf("Negatif");
6  else
7      printf("Zero");
```

5.4 while(cond)

```
1  while(3)          /* il boucle a l'infini */
2  {
3      /* n'importe quoi */
4  }
```

5.4.1 break

Il casse la répétitive et sort de la structure de contrôle.

5.4.2 continue

Il revient à tester la condition de la répétitive.

```

1 while (3)
2   if (a>3)
3     continue; /*retourne controler la condition du while*/

```

5.5 do....while(cond)

La boucle est faite au moins une fois et il teste en fin de boucle.

```

1 do
2   { /*les accolades sont obligatoires*/
3
4   } while (cond); /*ne pas oublier le ";" apres le while(cond)*/

```

5.6 for(initialisation; condition; évolution)

```

1 for (i=1; i<11; ++i)
2
3 for (i=1, j=10; i<j; ++i, --j)
4
5 for (i=3; a!=b; c+=9)

```

Aucune des parties n'est obligatoire.

```

1 for (; a<b; ) /*identique a un while(a<b)*/
2
3 for (;;) /*boucle sans fin*/
4
5 for (; (c<3)&&(e>1); printf("a"); --j) /*cela s'arrete des que la condition est fausse*/

```

5.7 switch() case

Permet de tester un entier avec un switch.

Dans le ou les case, il faut toujours des constantes entières.

```

1 char c;
2 switch(c)
3 {
4   case 'a':
5     /*instructions*/
6     break;
7   case 'b':
8   case 'B':
9     /*instructions*/
10   case 'c':
11   case 67:
12     /*instructions*/
13     break;
14   default: /*il est facultatif*/
15     /*instructions*/
16     break;
17 }

```

Chapter 6

scanf(”chaine”,&variables);

Fonction permettant d’entrer des données au clavier de façon formaté. Cette fonction se trouve dans la bibliothèque ”stdio.h”.

6.1 Directives %

- %c pour entrer une variable de type char
- %hd pour entrer une variable de type entier short décimal
- %ho pour entrer une variable de type entier short octal
- %dx pour entrer une variable de type entier short hexadécimal
- %ld pour entrer une variable de type entier long décimal
- %lo pour entrer une variable de type entier long octal
- %lx pour entrer une variable de type entier long hexadécimal
- %d pour entrer une variable de type entier int décimal
- %o pour entrer une variable de type entier int octal
- %x pour entrer une variable de type entier int hexadécimal
- %f pour entrer une variable de type réel à virgule flottante
- %e pour entrer une variable de type réel exponentiel
- %lf pour entrer une variable de type réel long flottant
- %le pour entrer une variable de type réel long exponentiel

```
1 short n, p;  
2 float f;  
3 scanf(”%hd_%hd”, &n, &p); /* les espaces sont autorises dans la chaine*/
```

6.2 Quelques fonctions supplémentaires

6.2.1 getch()

Cette fonction permet de lire un et un seul caractère au clavier. Elle se trouve dans la bibliothèque "conio.h". Elle lit un caractère et le renvoie.

```
1 char c;
2 c=getche();           /*c='A' c=65*/
```

Attention, cette fonction ne marche qu'en Windows. Si on veut utiliser un équivalent en linux, il n'est pas nécessaire de rajouter une bibliothèque. La fonction à utiliser se nomme getc.

```
1 char c;
2 c=getc(stdin);        /*stdin correspond a l'entree au clavier*/
3 printf("%c",c);       /*ne pas oublier la directive %c meme pour les chiffres*/
```

6.2.2 putchar()

Cette fonction affiche un et un seul caractère. Elle se trouve dans la bibliothèque "stdio.h".

```
1 char c=65;
2 putchar(c);           /*il affiche A*/
3 putchar('c');         /*il affiche c*/
4 putchar(66);          /*il affiche B*/
5 putchar(c+3);         /*il affiche D*/
6 putchar(c+32);        /*il affiche a*/
```

6.3 Caractères importants du code ASCII

Caractères	Code ASCII
'0'	48
'A'	65
'a'	97

Chapter 7

Fonction

Un programme en C est une succession de fonctions disjointes dont une seule porte le nom de main. En C, tous les passages de paramètres se font par valeur.

Syntaxe:

```
1 fonction(type nom; parametre)
2 {
3     type nom;
4 }

1 void main()                /*le type void est explique +bas*/
2 {
3     short n=1;
4     printf("\nn=%d",n);
5     fonction(n);
6     printf("\nNow_n=%d",n);    /*Now n=1*/
7 }
8
9 void fonction(short n)
10 {
11     printf("\nDans_f, n=%d",n);
12     n++;
13     printf("\nPuis_n=%d",n);
14 } /*destruction des variables locales a la fin de la fonction*/
```

Avec le mot clé "return", on peut retourner un et un seul résultat.

```
1 intf5 ();                /*prototype*/
2 shortf6 ();              /*prototype*/
3 shortf7 ();              /*prototype*/
4
5 void main()
6 {
7     short n;
8     n=f5 ();
9     printf("\nf5_retourne %d", n);
10    n=f6 ();
11    printf("\nf6_retourne %d", n);
12    printf("\nf7_retourne %d", f7 ());
13 }
14
15 f5 ()    /*on peut ajouter int devant mais c'est facultatif*/
16 {
17     return (5);          /*constante*/
18 }
19
20 short f6 ()
21 {
22     short n=6;
23     return(n);           /*variable*/
24 }
25
26 short f7 ()
```

```
27 {  
28   short n;  
29   return (n=7); /*expression*/  
30 }
```

7.0.1 Type des fonctions

Les fonctions ont un type qui est le type qu'elles renvoient.

```
1 long f()  
2 {  
3   long l;  
4   /*instructions*/  
5   return(l);  
6 }
```

Par défaut, le type est "int". Si je ne mets rien, elle renvoie du "int". Une fonction qui ne retourne rien est de type "void".

Important, il faut également respecter les prototypes des fonctions. C'est la même chose que la déclaration mais avec un point virgule à la fin. Le prototype doit être écrit avant main et la fonction en tant que telle s'écrit tout en bas.

```
1 long f(long c, long d);      /*prototype*/  
2  
3 void main()                  /*fonction principale*/  
4 {  
5   /*instructions*/  
6 }  
7  
8 long f(long c, long d)  
9 {  
10  /*instructions*/  
11 }
```

Chapter 8

Types, classes et formes

Un variable peut aussi avoir une classe.

```
1 /*exemple*/
2 short fonction(),
3 void main()
4 {
5     printf("\n1er_appel_donne_%d", fonction()); /*donne 1*/
6     printf("\n2eme_appel_donne_%d", fonction()); /*donne 1*/
7     /*car les variables sont detruites a la fin de chaque appel*/
8 }
9
10 short fonction()
11 {
12     short n=1; /*pour garder la valeur, il faut ecrire: static short n=1*/
13     return(n++);
14 }
```

8.1 Classe d'une variable

Détermine la durée de vie de la variable. Il y a deux classes:

- classe auto (pour automatique): la durée de vie est brève. La variable est créée et détruite pour chaque appel (elle va sur la pile: "stack segment"). C'est ce qui est pris par défaut.
- Classe static: la durée de vie dure tout le programme et les variables sont créées lors du chargement du programme (elles vont dans les "data segment").

Attention à l'initialisation des variables: les static ne sont créés qu'une seule fois en début.

Si on n'initialise pas, les variables auto contiennent n'importe quoi dedans et les variables static sont initialisées d'office à 0.

```
1 void main /*normalement il faut le prototype avant*/
2 {
3     short i;
4     for (i=1; i<=10; ++i)
5         fonct();
6 }
7
8 void fonct()
9 {
10     short n=15;
11     static short=15;
12     printf("\nn=%d m=%d", ++n, ++m);
13     /*      n=16      m=16
14             n=16      m=17
15             n=16      m=18
16             etc... */
17 }
```

8.1.1 Les variables static externe

On les appelle les variables globales: elles sont déclarées en dehors des fonctions. En C les variables globales ne sont connues que des fonctions qui suivent.

```

1  static short n1;          /*la variable est accessible par tout le monde*/
2  void main()
3  {
4
5  }
6  static short n2;          /*var globale initialisee a 0 accessible par tout le monde
7                             sauf par main*/
8  void f1()
9  {
10  static short n3;          /*var locale initialise a 0 a f1 et accessible qu'a f1 et
11                             garde sa valeur*/
12  short n4;                 /*var locale n'importe quoi dedans et accessible qu'a f1*/
13  }
14  void f2()
15  {
16  short n1;                 /*impossible car deja une var nommee n1 accessible f2*/
17  }

```

8.1.2 La classe registre

Elle est très peu utilisé. Elle peut s'utiliser pour les paramètres ou pour les variables automatiques (à la place d'auto). Elles ne sont pas créés en mémoire centrale mais placées dans les registres du microprocesseur: elles sont donc limitées. L'intérêt c'est qu'elles sont plus rapides pour les calculs (si elles sont souvent utilisées).

Le nombre de registre étant limité, elles ne seront pas toutes mises dans le registre du microprocesseur (en général, il y a 3 registres). Si il n'y a plus de place, le reste est placé sur la pile.

Puisque ces variables ne sont pas en mémoire centrale, elles n'ont pas d'adresse. Donc ne jamais les utiliser avec scanf() qui a besoin de l'adresse.

On ne peut pas mettre n'importe quoi dans les variables registres seulement:

- char
- short
- int
- pointeur

```

1  /*exemple*/
2  register short i, a, b, c, d, e;          /*il y a n'importe quoi dedans
3  for(i=0;i<30000;++i)                     /*i tournera beaucoup plus vite car variable registre*/
4  {
5  /*instructions quelconques*/
6  }

```


Chapter 9

Les vecteurs

Un vecteur est un nombre fini d'éléments de même type sous un même nom.

9.1 Déclaration

```
1 long v[10] /*vecteur de 10 entiers*/
```

10 DOIT être une constante car le programme doit savoir combien il doit en créer. Long est le type de chaque élément du tableau. La taille du tableau vaut 40 bytes (1 long = 4 bytes).

```
1 sizeof(v) /*cela ne marche pas*/
```

9.2 Éléments d'un vecteur

```
1 short tab[6];
```

--	--	--	--	--	--

 Le premier élément se trouve dans tab[0] et le dernier dans tab[5].
L'adresse du ième élément = adresse du 0ème + i*taille d'un élément.
Pour mettre 12 dans la 6ème case:

9.3 Initialisation

9.3.1 Vecteur auto

On ne peut pas les initialiser.

9.3.2 Vecteur static

On peut les initialiser.

```
1 static short tab[6]={1,10,2,20,3,30};
```

1	10	2	20	3	30
---	----	---	----	---	----

```
1 static short tab[6]={3,6}; /*tout le reste est 0*/
```

3	6	0	0	0	0
---	---	---	---	---	---

Si on n'initialise pas, il met tout à 0.

9.4 Vecteurs et fonctions

On ne peut jamais faire de return pour un vecteur.

```
1 void main()
2 {
3     static short v[5]={1,2,3,4,5};
4     f1(v);
5     f2(v[1]);
6 }
7
8 void f1(short tab[])
9 {
10    tab[2]=25;
11    tab[7]=6;    /* il ne contr le rien et met 6 */
12 }
13
14 void f2(short n)
15 {
16    n=18;        /* il ne passe rien car c'est en local */
17 }
```

Chapter 10




Pointeurs

C'est une variable qui peut contenir l'adresse d'une autre variable. Une adresse doit forcément être placée dans un pointeur, on ne peut pas la mettre ailleurs.

10.1 Déclaration

```
1 type *pointeur;
```

```
1 Exemples :
2 short a, b;
3 short *p1, *p2;
4 char c1, c2;
5 char *p3;
```

a
1000 1001
b
 1002 1003
c1
 1004
c2
 1005

Dans les pointeurs, il y a de base n'importe quelle adresse.

10.2 Adresse

&

```
1 p1=&a; /*p1 pointe a*/
2 p2=&c1; /*incorrect car pas le meme type*/
3 p3=&c1; /*p3 pointe l'adresse de c2*/
4 p3=&c2; /*p3 pointe l'adresse de c2 et perd l'adresse de c1*/
```

10.3 Adressage indirect

*

Cela permet d'affecter une valeur dans la variable pointée par le pointeur.

```
1 *p1=12; /*p1 met 12 dans la variable a*/
2 *p3=65; /*il mettra 'A' dans c2*/
3 *p2=18; /*incorrect car p2 ne pointe rien*/
4 p2=p1; /*p2 pointe la meme adresse que p1*/
5 *p2=*p1+3; /*12+3 est affecte a la variable que pointe p2*/
6 *p2=p1; /*incorrect*/
```

```

1 exemple
2 a*c;           /*a multiplie par c*/
3 char *c;       /*declarer c comme pointeur de char*/
4 b=*c;          /*dans b on met ce qui est pointe par c*/
5 b**c;          /*b multiplie par ce qui est pointe par c*/
6 *c*=b          /*ce qui pointe par c est multiplie par b*/

```

10.4 Pointeurs et fonctions

```

1 Exemple: fonction qui multiplie un nombre par 2
2
3 void main()
4 {
5     short n;
6     n=1;
7     fois2(&n);
8     printf("\n%d",n);
9 }
10 void fois3(short *p)
11 {
12     *p*=2      /*ce qui est pointe par p est multiplie par 2 puis destruction du pointeur p*/
13 }

```

10.5 Les pointeurs, les vecteurs et les fonctions

10.5.1 Adresse et Tableaux

c

--	--	--	--	--	--

```

1 short n,*p,v[6];      /*on declare un entier, un pointeur d'entier et un vecteur d'entier*/
2 p=&v[0];               /*adresse de la premiere case*/
3 p=&v[5];               /*adresse de la

```

Remarque: en C, le nom du tableau non-indicé (sans crochet) est l'adresse constante du tableau.

```

1 p=&v[0]                /*identique a p=v;

```

10.5.2 Tableaux et fonctions

```

1 main()
2 {
3     short n, *p,v[6];
4     p=v;
5     p=&v[5];
6     f(v);           /*met la valeur 3 dans la troisieme case*/
7 }
8
9 void f(short tab[]) /*identique a short *tab)
10 {
11     tab[2]=3;
12 }

```

10.5.3 Calcul sur pointeurs

w

--	--	--	--	--	--

```

1 long h,*p1,w[5],*p2;
2 p1=&l;      /*p1 pointe l*/
3 p2=w;      /*p2 pointe un vecteur*/
4 p2++;      /*p2 pointe une case plus loin (2eme case)*/
5 p2+=2;      /*p2 pointe deux cases plus loin (4eme case)*/

```

Les opérateurs pour les vecteurs: `==` `!=` `<` `>=`

Il existe l'adresse NULL qui est zéro (cela pointe sur rien). Cela permet d'éviter qu'un pointeur pointe sur n'importe quoi.

Voici 4 façons de faire la même chose:

```
w[2]=5;
*(p+2)=5;
p[2]=5;
*(w+2)=5;
```

Remarque:

```
1 p=w;           /*p pointe la 1eme case*/
2 p+2;           /*p pointe la 3eme case*/
3 *(p+1)=10      /*on met 10 dans la 4eme case*/
4 w++;           /*ERREUR on ne peut pas le bouger car c'est l'adresse CONSTANTE*/
```


Chapter 11

Les chaines de caractères

Une chaine de caractères est encadrée de "".

Ex: "Bob"

En fait, c'est un tableau de 4 caractères:

'B'	'o'	'b'	'\0'
66	111	98	0

Le \0 signifie la fin de la chaine.

11.1 Utilisation

```
1 char message[25]; /*chaine de 24 caracteres car il faut place pour \0*/
2 static char nom[4]="Bob"; /*il prend la taille par default*/
3 /*ou
4 static char nom[4]={ 'B', 'o', 'b', '\0' };*/
```

11.2 Chaines et fonctions

'S'	'a'	'l'	'u'	't'	'\0'
-----	-----	-----	-----	-----	------

```
1 char message[25], p;
2 static char nom[4]="Bob";
3 p=nom;
4 printf("Salut");
5 printf(p);
6 printf(nom+2);
7 printf(&nom[1]); /*affiche "ob"*/
```

Toutes fonctions qui manipulent une chaine de caractères, il y a un paramètre qui est un pointeur de char (qui reçoit le début du vecteur).

11.3 La directive %s

```
1 static char nom[30]="Bob";
2
3 printf("Nom=%s", nom); /*affiche Bob*/
4
5 printf("Caract=%c", nom[2]); /*identique a *(nom+2)*/
```

Le %s fonctionne également pour le scanf.

```
1 scanf("%s", nom+2); /*remplace le deuxieme 'b'*/
```

La directive %s s'arrête à l'espace.

Ex: Si on entre "DE HENAU" enter. Il aura pris "DE".

Chapter 12

argc, argv et redirections

```
1  /*etoiles.c*/
2  #include "stdio.h"
3  #define DEFALT 50      /*definit la constante DEFALT a 50*/
4  /*quand il rencontre DEFALT dans le texte, il le remplace par la valeur 50*/
5
6  void main()
7  {
8      short cb;
9      printf("Combien d'etoiles");
10     scanf("%hd",&cb);
11     if(cb<=0)
12     {
13         cb=DEFALT;
14     }
15     while(cb-->0)
16     {
17         putchar('*');
18     }
19     putchar('\n');
20 }
```

L'idée est que ce programme si on le lance, il nous demande combien d'étoiles, on rentre notre réponse et il affiche un nombre d'astérisques équivalent. Ce que l'on voudrait faire c'est de pouvoir lancer le programme en ligne de commande avec déjà le nombre d'étoiles :

etoile 5

Pour faire cela, nous avons besoin de argc et argv.

12.1 argc, argv

main est une fonction particulière et comme toute fonction, elle peut recevoir des paramètres via la ligne de commande. Pour pouvoir les utiliser, elle doit savoir 2 choses: combien il y en a et où ils sont.

argc est le compteur d'argument et permet de savoir combien il y a de paramètres. argv est le vecteur d'arguments et permet de savoir où sont les paramètres.

'p'	'r'	'o'	'g'	'\0'	'2'	'\0'	'a'	'b'	'c'	'3'	'1'	'\0'
-----	-----	-----	-----	------	-----	------	-----	-----	-----	-----	-----	------

Pour utiliser le nombre 31, on a besoin des fonctions qui permettent de prendre le début d'un vecteur et tout ce qui suit:

atoi(chaine) alpha vers integer

atol(chaine) alpha vers long

atof(chaine) alpha vers float

Ces fonctions se retrouvent dans la bibliothèque "stdlib.h".

```
1  void main(int argc, char *argv[])
2  {
3      printf("\n%d arguments passés", argc);
4      printf("\nNom du programme est %s", argv[0]);
5      printf("\n2eme argument est %s", argv[2]);
}
```

```

6  n=atoi(argv[3]);      /*mets 31 dans n*/
7  }

1  /*etoiles.c*/
2  #include "stdio.h"
3  #include "stdlib.h"
4  #define DEFAULT 50
5
6  void main(int argc, char*argv[])
7  {
8      short cb;
9      printf("Combien_d'etoiles");
10     scanf("%hd",&cb);
11     if(argc!=2)
12     {
13         cb=DEFAULT;
14     }
15     else
16     {
17         cb=atoi(argv[1]);
18     }
19     if(cb<=0)
20     {
21         cb=DEFAULT;
22     }
23     while(cb--)
24     {
25         putchar(' ');
26     }
27     putchar('\n');
28 }

```

12.2 exit()

La fonction exit permet de quitter le programme où que l'on soit dans le programme en renvoyant un code de sortie (de type int) au système d'exploitation. Elle se trouve dans "stdlib.h" et elle est de type void.

On renvoie zéro quand tout se passe bien (c'est une convention).

```

1  void exit(int);

1  /*etoiles.c*/
2  #include "stdio.h"
3  #include "stdlib.h"
4
5  void main(int argc, char*argv[])
6  {
7      short cb;
8      printf("Combien_d'etoiles");
9      scanf("%hd",&cb);
10     if(argc!=2)
11     {
12         printf("\nPas_bon_nombre_d'arguments");
13         exit(3);
14     }
15     if((cb=atoi(argv[1]))<=0)
16     {
17         printf("\nMauvais_argument");
18         exit(1);
19     }
20     while(cb--)
21     {
22         putchar(' ');
23     }
24     putchar('\n');
25     exit(0);
26 }

```

On peut utiliser return pour renvoyer un code de sortie mais dans ce cas il faut que la fonction main soit de type "int" et pas "void".

Chapter 13

Les fichiers

13.1 Ouvrir

`fopen(nom,mode);`

```
1  exemple :  
2  fopen("nomFichier","w");
```

Les différents modes sont:

- "r" en lecture
- "w" en écriture (efface ce qu'il y avait avant)
- "a" ajouter à la fin

Lorsque l'on ouvre un fichier, il renvoie un nombre qui représente un fichier qu'on doit mettre dans un pointeur de fichiers.

```
1  FILE *pf; /*envoie un pointeur sur le fichier (pointeur de fichier)*/  
2  pf=fopen("f1.txt","r"); /*si on ne sait pas ouvrir alors pf=0*/
```

13.2 Fichiers textes

13.2.1 Caractère par caractère

Lire un caractère `fgetc(pointeurDeFichier);`

Remarque: cela renvoie le caractère ou la marque de fin de fichier (EOF). Pour les fichier de caractères, il ne faut jamais utiliser des "char" mais bien des "short".

```
1  FILE *pf;  
2  short c; /*on remplace char par short*/  
3  if (pf=fopen("f1.txt","r"))!=NULL  
4  {  
5      c:fgetc(pf); /*envoie un char ou EOF(fin de fichier)=-1*/  
6                  /*sur 2 ou 4 bytes*/  
7  }
```

Ecrire un caractère `fputc(caractère, pointeur de fichier);`

S'il ne sait pas écrire, cela renvoie EOF (donc -1).

Fermer un fichier `fclose(pointeur_fichier);`

Cela ferme le fichier dont le pointeur est indiqué. Remarque: `exit(num);` ferme tous les fichiers.

13.2.2 Mot par mot

Lire fscanf(pointeurFichier,"...",adresse)

```

1  /*Exemple*/
2  char mot[20];
3  short n;
4  fscanf(pf,"%s",mot);    /*"abc" "12" "b" "242"*/
5  fscanf(pf,"%s %d",mot, &n); /*mot="abc", n=12*/
6  fprintf(pf2,"%s et %d", mot, n); /*ecrit dans un fichier*/

```

13.2.3 Ligne par ligne

fgets(où,taille,pointeurFichier);

Cela renvoie NULL lorsque l'on est à la fin du fichier.

```

1  char ligne[128];
2  fgets(ligne,128,pf);    /*lit une chaine dans un fichier et s'arrete au \n*/
3  fputs("OK",pf2);        /*ecrit une chaine dans un fichier*/
4                          /*si cela ne s'ecrit pas, renvoi NULL*/

```

13.3 Pointeurs de fichier: stdin, stdout, stderr

Ce qui suit est transparent à l'utilisateur. Pour tout programme en C quand il est exécuté, il accède à 3 flux de données.

Le flux d'entrée standard, le flux de sortie standard, le flux de sortie des messages d'erreur.

13.3.1 Normal (défaut)

13.3.2 Redirection

Un fichier est ouvert automatiquement en lecture pour l'entrée standard et un fichier est ouvert automatiquement en écriture pour la sortie standard.

stdin et stdout

stdin: c'est le pointeur de fichier associé au fichier ouvert en lecture. Par défaut c'est le clavier.

stdout: c'est le pointeur de fichier associé au fichier ouvert automatiquement en écriture. Par défaut c'est l'écran.

Remarque:

in	c=fgetche(stdin);	idem	c=getche();
out	fputc(c,stdout);	idem	putchar(c);
in	fscanf(stdin,"%s %s", m, &n);	idem	scanf("%s %d",m, &n);
out	fprintf(stdout,"Le %d",a);	idem	printf("Le %d",a);
in	fgets(nom,taille,stdin);	idem	getf(nom);
out	fputs(chaine,stdout);	idem	puts(chaine);

stderr

C'est le pointeur de fichier ouvert automatiquement pour la sortie des messages d'erreurs (c'est toujours l'écran).

```

1  /*etoiles.c*/
2
3  void main(int argc, char *argv[])
4  {
5      if(argc!=2)
6      {
7          printf("\nPas d'arguments");
8          exit(2);
9      }
10 }

```

Si je lance le programme : etoiles 3

Si je lance le programme: etoiles
Pas d'arguments

Si je lance le programme: etoiles 3>f
Il n'affiche rien à l'écran mais écrit *** dans f.

Si je lance le programme: etoiles >f
Il n'affiche rien à l'écran et écrit Pas d'arguments dans f.

Pour éviter cela, on utilise stderr. On remplace donc la ligne pour que cela affiche toujours le message d'erreur à l'écran.

```
fprintf(stderr, "\nPas d'arguments");
exit(2);
```

13.3.3 Travail par blocs

Il existe deux autres fonctions pour lire et écrire dans un fichier: fread() et fwrite()
Elles permettent de lire et d'écrire n'importe quoi.

fread

fread se trouve dans "stdio.h" et s'écrit:
int fread(où, taille, combien, pointeurFichier);
où = où on va stocker ce qui a été lu
taille = taille de ce qui a été lu
combien = combien on veut en lire

Cela renvoie le nombre d'informations lues.

```
1 short n,v[10],nb;
2 nb = fread(&n;2,1,f);
3 nb=fread(v+9,2,5,f);
4 fread() /*renvoie 0*/
```

fwrite

int fwrite(où, taille, combien, pointeurFichier);
où = où il prend en mémoire centrale
taille = taille du fichier qu'il prend
combien= combien il prend
pointeurFichier = pointeur où il va stocker

13.3.4 Gestion des erreurs

```
1 if((c=getc(f))==EOF)
2 /*renvoie soit si fin de fichier ou s'il y a une erreur donc rajout de */
3 {
4     if(feof(f))
5         /*fin de fichier*/
6     else
7         /*erreur sur fichier*/
8     }
9
10 if((c=getc(f))==EOF)
11 {
12     if(!ferror(f))
13         /*erreur sur le fichier*/
14     else
```

```

15  /*fin de fichier*/
16  }

```

Il existe la fonction `perror(nomFichier)`; qui affiche un diagnostique de l'erreur

13.4 Se positionner dans un fichier

13.4.1 ftell

Renvoie en nombre de bytes la position actuelle dans le fichier. Dès que le fichier est ouvert au début, elle renvoie 0.

```

1  ftell(pointeur_fichier)

```

13.4.2 fseek

Pour se positionner ou pour se déplacer à l'intérieur du fichier.

```

1  int fseek(pointeur_fichier ,rang ,comment)

```

Le rang c'est le nombre de byte, positif comme négatif. positif on avance, négatif on recule.

Comment: si c'est 0, c'est par rapport au début du fichier, si c'est 1 c'est par rapport à la position courante et si c'est 2 c'est par rapport à la fin du fichier.

Elle renvoie 0 si cela n'a pas marché.

```

1  int  taille=100;
2
3  fseek(f,0L,0); /*0l permet de forcer en long*/
4
5  fseek(f,(nb-1)*taille,0);

```

Chapter 14

La récursivité

Le C est un langage récursif: une fonction peut s'appeler elle-même.

Chapter 15

Les tables à plusieurs dimensions

15.1 Tables à deux dimensions

15.1.1 Déclaration et initialisation

Pour les déclarer, on utilise deux crochets (toujours utiliser des constantes dans les crochets):

```
1 short m[4][2]; /*d'abord les lignes puis les colonnes*/
```

Pour les initialiser:

```
1 static short m[4][2]={1,2,3,4,5};          /*remplit de gauche a droite puis ligne suivante*/
2                                           /*les cases non initialisees le sont a 0*/
3 static short m[4][2]={ {1,2},
4                       {3},
5                       {4}};
```

15.1.2 Éléments et adresses

m: adresse de &m[0][0]

m[2]: adresse de&m[2][0]

m[2][1]: c'est la valeur de la case

Les tables passent toujours des adresses.

15.1.3 Fonctions

```
1 f1(m,...);
2
3 void f1(short tab[][2])          /*chaque ligne a deux elements*/
4 {
5     tab[1][1]=10;              /*cela modifie la table*/
6 }
7 void f2(short tab [...])        /*adresse d'un vecteur*/
8 {
9     tab[2]=25;                 /*cela modifie la table*/
10 }
11 void f3(short m, ...)          /*adresse d'un lment */
12 {
13     m=30;                      /*cela ne modifie pas la table car il n'y a pas d'adresse*/
14 }
```

15.2 Tables à trois dimensions

15.2.1 Déclaration et initialisation

Il faut trois crochets:

```
1 short m[3][4][2]; /*table a 4 lignes sur 2 colonnes reproduites 3*/
```

Pour initialiser:

```
1 static short m[3][4][2]={1,2,3,4,5,6,7,8,9};
2
3                               ={{1},{2}}; /*on regroupe par ligne*/
4 /*1 dans la 1er case de la 1er ligne et 2 dans la 1er case de la 2eme ligne*/
5                               ={{{1}},{2,3},{4,5}}; /*on regroupe par matrice*/
6 /*1 dans 1er case 1er matrice et le reste dans la 2eme matrice*/
```

15.2.2 Éléments et adresses

m: adresse de &m[0][0][0]
m[2]: adresse de &m[2][0][0]
m[i]: adresse de &m[i][0][0]
m[1][1]: adresse de &m[1][1][0]
m[0][1][1]: valeur d'une case

15.2.3 Fonctions

```
1 f4(m,...);
2 f1(m[1],...); /*on peut le passer qu'a f1*/
3 f2(m[2][0],...); /*on le passe a f2 car adresse d'une ligne*/
4 f3(m[1][1][1],...); /*on le passe a f3 car adresse d'un element*/
5
6 void f4(short tab[][4][2]); /*seul le 1er est optionel*/
7 {
8     tab[0][0][1]=30
9 }
```

15.3 Tables à ? dimensions

15.3.1 Déclaration et initialisation

```
1 short t[7][6][5][3][4][2]; /*table a 6 dimensions*/
```

15.3.2 Fonctions

```
1 f1(t[2][1][0][1],...); /*adresse d'une table a deux dimensions*/
2 f?(t[0]0,...); /*pas de fonction car adresse table 4 dimensions*/
3 f4(t[6][5][4],...);
```

Chapter 16

Les pointeurs de pointeurs

16.1 Tri selon les vecteurs de pointeurs

Exercice selon la méthode des vecteurs de pointeurs.

```
1 /*Tri a bulle dans pour des chaines de caracteres*/
2 static char *noms[3]={ "Bob" ," Marie" ," Jo" };
3 bulles (noms,3);
4
5 void bulles(char *v[], short t){
6
7 }
```

O	k	\0			
P	l	U	s	\0	
M	o	i	n	s	\0
B	o	b	\0		
'o'	'k'	\0			

16.2 Pointeurs de pointeurs

```
1 /*Syntaxe:*/
2 char **p; /*pour un pointeur de char*/
```

Exercice selon la méthode des vecteurs de pointeurs.

```
1 static char *noms[3]={ "Bob" ," Marie" ," Jo" };
2 bulles (noms,3);
3
4
5
6 char **p=noms;
7 *(p+1) /*c'est l'adresse de Marie*/
8 **(p+1) /*c'est le 'M' de Marie*/
9
10 (*p)+1 /*adresse 1001 = adresse de la lettre 'o' de Bob*/
11 *(p+1) /*c'est le 'o' de Bob*/
12
13 *((p+2)+1) /*C'est le 'o' de Jo*/
```

Formule générale:

$(*(p+i)+j)+k$: ième chaine, de la jème caractères et modification du caractère en question avec k.

Amélioration du argv:

```
void main(int argc, char **argv[])
```

Exemples:

Si étoiles 5a, le a est trouvé avec $*(*(argv+1)+1)=='a'$

Chapter 17

Structures et unions

C'est l'équivalent des enregistrements en pascal ou en algorithmies. C'est un ensemble fini de données de type différent.

```
1  /*exemple*/
2  void main(){
3      struct exemple{
4          /*le fait de donner un nom a struc permet de le reutiliser*/
5              int nb;
6              char mot[11];
7              double x;
8              }ex1,ex2;
9
10
11  /*On peut declarer struct avant le main comme cela il est connu*/
12  struct exemple{
13      int nb;
14      char mot[11];
15      double x;
16      }ex1,ex2;
17
18  void main(){
19      struct exemple ex1,ex2;
20      ex1.nb=1; /*affecter valeur a nb*/
21      strcpy(ex1.mot,"Bob"); /*mettre Bob dans mot*/
22      ex1.mot[0]='B' /*mettre B dans la premiere case de mot*/
23  }
```

17.1 Les structures et les fonctions

Cela permet d'effectuer des listes chaînées.

```
1  struct exemple{
2      int nb;
3      char mot[11];
4      double x;
5      }ex1,ex2;
6
7  void f1(...); /*le prototype doit se mettre apres struct*/
8
9  void main(){
10     struct exemple ex1,ex2;
11     ex1.nb=1; /*affecter valeur a nb*/
12     strcpy(ex1.mot,"Bob"); /*mettre Bob dans mot*/
13     ex1.mot[0]='B' /*mettre B dans la premiere case de mot*/
14     f1(ex1);
15     f2(&ex1);
16     /*on peut faire des vecteurs de structures*/
17 }
```

```

18 }
19
20 void f1(struct exemple ex){
21     ex.nb++;      /*rien n'est modifie*/
22 }
23
24 void f2(struct exemple *p){
25     /*quand on a un pointeur de structure, on utilise la fleche*/
26     p->nb=4;      /*c'est modifie*/
27 }

```

17.2 Vecteur de structures

On peut faire des vecteurs de structures

```

1 struct exemple{
2     int nb;
3     char mot[11];
4     double x;
5     }ex1,ex2;
6
7
8 void main(){
9     struct exemple ex1,ex2,v[10];
10    ex1.nb=1;      /*affecter valeur a nb*/
11    strcpy(ex1.mot,"Bob");      /*mettre Bob dans mot*/
12    ex1.mot[0]='B'      /*mettre B dans la premiere case de mot*/
13    f1(ex1);
14    f2(&ex1);
15    f1(v[1]);
16    v[1].nb      /*adresse du debut de la structure*/
17    f2(v+1);
18    (v+1)->nb=5;      /*identique a v[1].nb=5;*/
19 }
20
21 void f1(struct exemple ex){
22     ex.nb++;      /*rien n'est modifie*/
23 }
24
25 void f2(struct exemple *p){
26     /*quand on a un pointeur de structure, on utilise la fleche*/
27     p->nb=4;      /*c'est modifie*/
28 }

```

17.3 Structure de structure

```

1 struc exen{
2     short n;
3     struct exemple v[5];
4     char m[6];
5     };
6
7     /*exemple*/
8     struct exen e;
9     scanf("%d",&e.v[e.n+1].nb);
10
11     struct exen e, *p=&e;
12     p->v[p->n+1].nb

```

Pour faire des listes chaînées:

```

1 struct exemple{
2     int nb;
3     char mot[11];
4     struct exemple *next;
5     }*debut,ex;

```

17.4 Unions

Une union est un moyen de permettre à différents types d'occuper le même espace mémoire. Une union ne peut contenir qu'une valeur d'un des types.

On définit un schéma de l'union:

```
1  union exemple
2      {
3          short n;
4          char mes[10];
5          long l;
6      };
7
8  union exemple e1, e2, *p;      /*c'est soit un short, soit un char soit un long*/
9  e1.n=0;
10 strcpy(e1.mes,"Bob"); /*le n=0 est perdu car il a ete ecrase par "Bob"*/
11 p=>l=1040;
```

Un champs de l'union peut être une structure.

Chapter 18

Gestion dynamique de la mémoire

Créer des variables pendant l'exécution du programme et les détruire quand on en a plus besoin (à la fin du programme).

Plusieurs façons de le faire (bibliothèque "alloc.h" ou "stdlib.h"):

18.1 malloc

prototype:

char *malloc(unsigned int taille);

Ancienne façon de l'écrire. La nouvelle façon est : void *malloc(unsigned int taille); crée une variable et cela renvoie son adresse. Cela renvoie NULL quand ce n'est pas possible.

```
1 char *p;
2
3 p = malloc(10);
4 strcpy(p, "Bob");
5
6 /* avec la nouvelle écriture, on écrit :
7 p=(char*) malloc(10);*/
```

malloc n'initialise pas à zéro et il y a donc n'importe quoi.

18.2 calloc

Pour initialiser des tableaux.

Prototype:

void *calloc(unsigned int elements, unsigned int taille);

```
1 short *p2;
2 p2= (short*) calloc(5, sizeof(short));
3 *(p2+2)=8; /* pour mettre 8 dans la 3eme case*/
4 p2[2]=8; /* autre facon d'ecrire la meme chose*/
```

calloc initialise tout à zéro.

18.3 realloc

Permet de remodifier la taille d'une zone dynamique.

Prototype:

void *realloc(void *ancienneAdresse, unsigned int taille);

Cela renvoie l'adresse du nouveau bloc ou null si cela ne marche pas.

```
1 printf("%s_%p", p, p); /* affiche Bob adresse de p*/
2 p=(char*) realloc(p, 20);
3 printf("%s_%p", p, p); /* affiche Bob et nouvelle adresse de p*/
4
```

```
5  if(pNew=(short*)realloc(p2,10*sizeof(short)))
6  {
7      p2= pNew;      /*pour eviter que null efface p2*/
8  }
9  else
```

18.4 free

Pour libérer l'espace mémoire.

Prototype:

void free(void *adresse);

```
1  free(p);      /*libere p*/
```

Appendix A

Exercices

A.1 Structures de contrôle

A.1.1 Minimum de 3 entiers

Enoncé Créer une fonction qui permet d'afficher le minimum de 3 entiers encodés par l'utilisateur.

```
1 #include "stdio.h"
2 #include "stdlib.h"
3
4 int main()
5 {
6     short a,b,c,min;
7     printf("\Entrez 3 entiers");
8     scanf("%hd_%hd_%hd",&a,&b,&c);
9
10    if(a<b)
11        min=a;
12    else
13        min=b;
14
15    if(c<min)
16        min=c;
17
18    printf("\nMinimum=%d", min);
19 }
```

```
1 /*Autre solution*/
2
3 min = (a<b)?a:b;
4 printf("\nMinimum=%d", ((c<min)?c:min);
```

A.1.2 Résolution d'une équation du second degré

Enoncé Créer un programme qui permet de résoudre une fonction du second degré.

```
1 #include <stdio.h>
2 /*equation du seconde degre
3    a, b, c entier
4    racines reelles*/
5
6 #include "stdlib.h"
7 #include "math.h"
8
9 main()
10 {
11     short a,b,c,delta;
12     printf("\nEntrez les valeurs a, b et c pour la fonction
13 du second degre ax^2+bx+c=0\n");
14     scanf("\n%hd_%hd_%hd", &a, &b, &c);
15
16     if(a!=0)
```

```

17 {
18     delta=b*b-4*a*c;
19
20     if(delta > 0)
21     {
22         printf("\n2_solutions_%.2f_et_%.2f",
23             (-b+sqrt(delta)/(2*a)), (-b-sqrt(delta)/(2*a)));
24     }
25     else if(delta == 0)
26     {
27         printf("\nLa_resolution_de_la_fonction_est_de_%.2f",
28             -(float)b/(2*a));
29     }
30     else
31         printf("\nIl_n'y_a_pas_de_solution_pour_cette_equation");
32     }
33     else
34         printf("\nPas_du_second_degre");
35 }

```

A.1.3 Tableau ASCII

Enoncé Afficher une table des codes ASCII selon la présentation suivante:

il y a 256 caractères où chaque caractères est écrit: dec=065, oct=101, hex=41 ASCII='A'

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     short i;
7     printf("\nTableau_ASCII_(en_decimal ,_octal ,_hexa_et_code_ASCII)");
8     for (i=1;i<32;i++)
9     {
10         printf("\n%03d\t%03o\t%02x\tCtr",i,i,i);
11     }
12
13     for (;i<256;++i)
14     {
15         printf("\n%03d\t%03o\t%02x\t'%c'",i,i,i,i);
16     }
17
18 }

```

A.2 Scanf

A.2.1 Nombre binaire

Enoncé Entrer un nombre entier de 2 bytes (max 16 entrées) en binaire, l'afficher en décimale et en hexa. L'entrée s'arrête dès que l'introduction n'est pas bonne ou que l'entier est rempli.

exemple:

entrez un nombre binaire

1

1

0

enter ou tout autre touche que 1 et 0

cet entier vaut 6 ou 6 en hexadécimal

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "conio.h"
4
5 main()
6 {

```

```

7  short nb=0, cpt=1;
8  char c;
9  printf("\nEntrez un nombre entier binaire");
10 c=getche(); /* am lioration: ne pas faire cette ligne et la place pour la suivante
11             while ((cpt++<17)&&(((c=getche())=='0')||(c=='1')))* /
12
13 while((cpt<17)&&((c=='0')||(c=='1')))
14 {
15     nb=2*nb+c-48;
16     ++cpt;          /* a supprimer */
17     c=getche();      /* a supprimer */
18 }
19 printf("\nLe nombre est %d et %x en hexa", nb, nb);
20 }

1 /* autre solution */
2 for(nb=0, cpt=1, printf(); (cpt++<17)&&(((c=getche())=='0')||(c=='1'))); nb=2*nb+c-48)

```

A.3 Les fonctions

A.3.1 Nombre mystère (résolution dichotomique)

Enoncé L'utilisateur choisit un nombre entier entre 1 et 15 et le programme doit essayer de le deviner. Il commence par proposer 8 et l'utilisateur tape H (si c'est trop haut), B (si c'est trop bas), E (si c'est exact) et S (pour stopper le programme). On peut également préciser le nombre de tentatives.

On peut également ajouter une fonction qui convertit automatiquement les caractères tapés en majuscule.

```

1 #include "stdio.h"
2 #include "conio.h"
3
4 char majuscule(char c);          /* prototype */
5
6 void main()
7 {
8     short essai = 8, rang=4, cpt=1;
9     char c;
10    printf("\nJe pense _ %d", essai);
11    while((((c=majuscule(getche()))!='E') && (c!='S')))
12    {
13        switch(c)
14        {
15            case 'B':
16                essai+=rang;
17                rang/=2;
18                cpt++;
19                break;
20
21            case 'H':
22                essai-=rang;
23                rang/=2;
24                cpt++;
25                break;
26
27            default:
28                printf("\nRepondez H, B, S ou E");
29                break;
30        }
31        printf("\nJe pense _ %d", essai);
32    }
33
34    if(c=='E')
35        printf("\nTrouve en %d coups", cpt);
36 }
37
38 char majuscule(char c)
39 {
40     /* Rappel: 'a' <=> 97 <=> 0110 0001

```

```

41      'A' <==> 65 <==> 0100 0001*/
42  if (c>='a')&&(c<='z')) /*on pourrait changer 'a' en 97*/
43      c-=32;
44  return(c);
45  }

```

```

1  /*Autre facon de creer la fonction*/
2
3  char majuscule(char c)
4  {
5      return((c>='a')&&(c<='z')?c-32:c);
6  }

```

```

1  /*Autre facon de creer la fonction: on force le 6eme bit a zero en binaire pour
2      mettre en majuscule*/
3  /*pas question examen*/
4
5  char majuscule (char c)
6  {
7      /*utilisation d'un masque AND: le 6eme bit est a zero et le reste est a 1
8      ex:  1101 1111
9           D      F*/
10     return((c>='a')&&(c<='z')?c&0xDF:c);
11 }

```

Remarque Il existe des fonctions permettant de déterminer si c'est une majuscule, une minuscule, ...

```

1  #include "ctype.h"      /*ajout de la bibliotheque*/
2
3  islower(char c)
4  isupper(char c);
5  isctrl(char c);
6  isdigit(char c);

```

```

1  /*Autre facon d'utiliser la fonction*/
2  char majuscule(char c)
3  {
4      return(islower(c)?c&0xDF:c);
5  }

```

A.3.2 Pyramide

Enoncé Programme qui affiche cette pyramide.

```

    1
   232
  34543
 4567654
456898765

```

```
0123456789876543210
```

A.3.3 Nombre de Armstrong

Un nombre d'Armstrong est un entier dont la somme des cubes des chiffres qui le composent est égale à lui même.

ex: $153 = 1^3 + 5^3 + 3^3$

Enoncé Programme principal: afficher tous les nombres d'Armstrong entre 1 et 1000. Utiliser deux fonctions: une fonction carré et une fonction cube, qui calcule le cube d'un nombre (cube doit appeller carré)

```

1  #include <stdio.h>
2
3  short carre(short a);           /*prototype*/
4  short cube(short a);           /*prototype*/
5
6  void main()
7  {
8
9      short i, somme=0, limite;
10     printf("Choisissez une valeur maximale en entier\n");
11     scanf("%hd",&limite);
12
13     printf("Programme qui affiche tous les nombres d'Armstrong de 1 à %d", limite);
14
15
16     for(i=1;i<=limite;i++)
17     {
18         short a=i;
19         while(a>0)
20         {
21             somme+=cube(a%10);
22             a/=10;
23         }
24
25         if(somme==i)
26         {
27             printf("%d",somme);
28         }
29         somme=0;
30     }
31 }
32
33 short cube(short i)
34 {
35     short n;
36     n=carre(i);
37     return(n*i);
38 }
39
40 short carre(short i)
41 {
42     return(i*i);
43 }

```

```

1  /*Solution du prof*/
2  #include "stdio.h"
3
4  short carre(short n);  /*prototype*/
5  short cube(short n);  /*prototype*/
6
7  void main()
8  {
9      short i, som, tmp;
10     for(i=1;i<=10000;++i)
11     {
12         tmp=i;
13         som=0;
14         while(tmp!=0)      /*peut s'ecrire while(tmp)*/*
15         {
16             som+=cube(tmp%10);
17             tmp/=10;
18         }
19         if(i==som)
20             printf("nnbr_Armstrong_",i);
21     }
22 }
23
24 short cube(short n)

```

```

25 {
26     return(n*carre(n));
27 }
28
29 short carre(short n)
30 {
31     return(n*n);
32 }

```

A.3.4 Addition et soustraction d'entiers

Enoncé Deux fonctions: la première qui permet d'entrer la valeur d'un entier caractère par caractère (ex: '1' '3' + enter donne 13). On ne gère que les positifs. Si c'est n'importe quoi, on considère que c'est 0.

Deuxième fonction qui affiche le menu de calcul:

```

1  2 nombres:      appel de la fonction 1  12
2                                     13
3      Menu
4      1. Addition de 12 et 23
5      2. Soustraire 12 de 23
6      3. Soustraire 23 de 12
7      Choix?

```

Attention, le choix ne sera pas lu dans la fonction mais bien dans le programme principal.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  // #include "conio.h"      /*pour le moment conio.h ne fonctionne pas sur codeblock*/
4
5  void menu(a,b);          /*prototype*/
6
7  int main()
8  {
9      short a,b,choix;
10     printf("\nChoisissez 2 entiers supérieurs à 0\n");
11     scanf("%hd%hd", &a,&b);
12     while((a<0)|| (b<0))
13     {
14         printf("\nLes 2 entiers doivent être plus grands que 0\n");
15         scanf("%hd%hd", &a,&b);
16     }
17
18
19     menu(a,b);
20     scanf("%hd", &choix);
21
22     switch(choix)
23     {
24         case 1:
25             printf("\n%d + %d = %d", a, b, a+b);
26             break;
27
28         case 2:
29             printf("\n%d - %d = %d", a, b, a-b);
30             break;
31
32         case 3:
33             printf("\n%d - %d = %d", b, a, b-a);
34             break;
35
36         default:
37             printf("Choix non recevable");
38             break;
39     }
40
41     return 0;
42 }
43
44 void menu(a,b) /*fonction menu*/

```



```

45     {
46         printf("\nMenu");
47         printf("\n1. _Addition _de _%d _et _%d", a, b);
48         printf("\n2. _Soustraire _%d _de _%d", a, b);
49         printf("\n3. _Soustraire _%d _de _%d", b, a);
50         printf("\nChoix?\n");
51     }

1  /* Solution du prof */
2  #include "stdio.h"
3  #include "conio.h"
4  #include "ctype.h"
5
6  short get_nbre();
7  void menu(short a, short b);
8
9  void main()
10 {
11     short n1, n2;
12     printf("\nEntrez _2 _entiers _positifs");
13     if((n1=get_nbre())&&(n2=get_nbre()))
14     {
15         menu(n1, n2);
16         switch (get_nbre())
17         {
18             case 1:
19                 printf("\n%d+_%d=_%d", n1, n2, n1+n2);
20                 break;
21             case 2:
22                 printf("\n%d-_%d=_%d", n2, n1, n2-n1);
23                 break;
24             case 3:
25                 printf("\n%d-_%d=_%d", n1, n2, n1-n2);
26                 break;
27             default:
28                 printf("\nChoix _non _valable");
29         }
30     }
31 }
32
33 short get_nbre()
34 {
35     char i;
36     short nb=0;
37     while((c=getche())>='0')&&(c<='9'))    /*ou: while(isdigit(c=getche()))*/
38         nb=nb*10+c-48;
39     return (c=='\n'?nb:0);
40 }
41
42 void menu(short a, short b)
43 {
44     printf("\n\nMENU");
45     printf("\n1. _Addition _de _%d _et _%d", a, b);
46     printf("\n2. _Soustraire _%d _de _%d", b, a);
47     printf("\n3. _Soustraire _%d _de _%d", a, b);
48     printf("\nChoix _?");
49 }

```

A.4 Les vecteurs

A.4.1 Calculatrice polonaise

Enoncé Créer une calculatrice en notation polonaise inversée et la simuler en créant un vecteur de 20 cases pour y placer les nombres.

Elle n'accepte que du chiffre, que le +, I (pour afficher le dernier calcul) et S pour stopper le programme.

12

25


```

59 .....default :
60 .....printf("\nQuoi?");
61 .....ok=0;
62 .....break;
63 .....}
64 .....}
65 .....}
66 .....}

```

A.4.2 Permutation d'un vecteur d'entiers

Enoncé Une fonction qui affiche n'importe quel vecteur de short et une deuxième fonction qui fait une permutation symétrique de n'importe quel vecteur de short.

Il faut les utiliser dans un programme avec un vecteur de 7 short à remplir, à afficher, à permuter et à afficher à nouveau.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void affiche(short v[], short n);
5  void permutation(short v[], short n);
6
7  int main()
8  {
9      printf("\nProgramme qui remplit un vecteur de 7 entiers, l'affiche, le permute et le réaffiche");
10     static short v[7];
11     short i;
12     for (i=0; i<7; i++)
13     {
14         printf("\nEntrez un entier pour la %dème position\n", i+1);
15         scanf("%hd", &v[i]);
16     }
17     affiche(v, 7);
18     printf("\n");
19     permutation(v, 7);
20     affiche(v, 7);
21     return 0;
22 }
23
24 void affiche(short v[], short n)
25 {
26     short i;
27     for (i=0; i<n; i++)
28     {
29         printf("%d", v[i]);
30     }
31 }
32
33 void permutation(short v[], short n)
34 {
35     short tmp, i;
36     for (i=0; i<n; i++)
37     {
38         tmp=v[n-1];
39         v[n-1]=v[i];
40         v[i]=tmp;
41         n--;
42     }
43 }

```

```

1  /* Solution du prof */
2  #include "stdio.h"
3
4  void echange(short *p1, short *p2);
5  void affiche(short v[], short tail);
6  void permute(short v[], short tail);
7
8  void main()
9  {

```

```

10  short v[7], i;
11  printf("\nRemplir");
12  for(i=0; i<7; ++i)
13  {
14      scanf("%hd", &v[i]);
15  }
16  printf("\nAvant\n");
17  affiche(v, 7);
18  permute(v, 7);
19  printf("\nAprès\n");
20  affiche(v, 7);
21  }
22
23  void affiche(short v[], short tail)
24  {
25      short i;
26      for(i=0; i<tail; i++)
27          printf("%d ", v[i]);
28  }
29  permute(short v[], short tail)
30  {
31      short i, tmp;
32      for(i=0; i<tail/2; ++i)
33      {
34          tmp=v[i];
35          v[i]=v[tail-i-1];
36          v[tail-i-1]=tmp;
37          /* si la fonction echange avait ete utilisee toutes les intructions auraient ete remplacees par :
38          echange(&v[i], &v[tail-i-1]) */
39      }
40  }

```

A.5 Les pointeurs

A.5.1 Echange de deux entiers à l'aide de pointeurs

Enoncé Fonction qui échange deux entiers à l'aide des pointeurs.

```

1  /* Solution */
2  #include "stdio.h"
3
4  void permute(short *p1, short *p2);    /* prototype */
5
6  void main()
7  {
8      short a=10, b=15;
9      printf("\nFonction qui permute 2 entiers %d et %d", a, b);
10     permute(&a, &b);
11     printf("\nEntiers permutes : %d et %d", a, b);
12 }
13
14 void permute(short *p1, short *p2)
15 {
16     short tmp;
17     tmp=*p1;
18     *p1=*p2;
19     *p2=tmp;
20 }

```

A.5.2 Permutation d'un vecteur d'entiers : variante 1

Enoncé Refaire l'exercice précédent avec un pointeur et aucun `v[]` dans la fonction de permutation.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void affiche(short v[], short n);
5  void permutation(short v[], short n);

```

```

6
7 int main()
8 {
9     printf("\nProgramme qui remplit un vecteur de 7 entiers, l'affiche, le permute et le reaffiche");
10    static short v[7];
11    short i;
12    for (i=0; i<7; i++)
13    {
14        printf("\nEntrez un entier pour la %deme position\n", i+1);
15        scanf("%hd", &v[i]);
16    }
17    affiche(v, 7);
18    printf("\n");
19    permutation(v, 7);
20    affiche(v, 7);
21    return 0;
22 }
23
24 void affiche(short *v, short n)
25 {
26     short i;
27     for (i=0; i<n; i++)
28     {
29         printf("%d", v[i]);
30     }
31 }
32
33 void permutation(short *v, short n)
34 {
35     short tmp, i, *p;
36     p=v;
37     p+=(n-1);
38     for (i=0; i<n/2; i++)
39     {
40         tmp=*(v+i);
41         *(v+i)=*p;
42         *p=tmp;
43         p--;
44     }
45 }

```

```

1  /* Solution du prof*/
2  #include "stdio.h"
3
4  void echange(short *p1, short *p2);
5  void affiche(short *v, short tail);
6  void permute(short *v, short tail);
7
8  void main()
9  {
10     short v[7], i;
11     printf("\nRemplir");
12     for (i=0; i<7; ++i)
13     {
14         scanf("%hd", &v[i]);
15     }
16     printf("\nAvant\n");
17     affiche(v, 7);
18     permute(v, 7);
19     printf("\nAprès\n");
20     affiche(v, 7);
21 }
22
23 void affiche(short *v, short tail)
24 {
25     short i;
26     for (i=0; i<tail; i++)
27         printf("%d_", *(v+i)); /*les parentheses sont obligatoires*/
28     /*on aurait pu écrire cela *v++ au lieu de *(v+i)*/
29 }
30 permute(short v[], short tail)

```

```

31 {
32     short i, tmp;
33     for (i=0; i<tail/2; ++i)
34     {
35         tmp=*(v+i);
36         *(v+i)=*(v+tail-i-1);
37         *(v+tail-i-1)=tmp;
38         /* si la fonction echange avait ete utilisee toutes les intructions auraient ete remplacees par:
39         echange((v+i),(v+tail-i-1))*/
40     }
41 }

```

A.5.3 Permutation d'un vecteur d'entiers : variante 2

Enoncé Refaire l'exercice précédent avec un pointeur au début et un à la fin du vecteur.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void affiche(short v[], short n);
5  void permutation(short v[], short n);
6
7  int main()
8  {
9      printf("\nProgramme qui remplit un vecteur de 7 entiers, l'affiche, le permute et le réaffiche");
10     static short v[7];
11     short i;
12     for (i=0; i<7; i++)
13     {
14         printf("\nEntrez un entier pour la %deme position\n", i+1);
15         scanf("%hd", &v[i]);
16     }
17     affiche(v, 7);
18     printf("\n");
19     permutation(v, 7);
20     affiche(v, 7);
21     return 0;
22 }
23
24 void affiche(short *v, short n)
25 {
26     short i;
27     for (i=0; i<n; i++)
28     {
29         printf("%d", v[i]);
30     }
31 }
32
33 void permutation(short *v, short n)
34 {
35     short tmp, i, *p1, *p2;
36     p1=v;
37     p2=v;
38     p2+=(n-1);
39     for (i=0; i<n/2; i++)
40     {
41         tmp=*p2;
42         *p2=*p1;
43         *p1=tmp;
44         p2--;
45         p1++;
46     }
47 }

```



```

1  /* Solution du prof*/
2  #include "stdio.h"
3
4  void echange(short *p1, short *p2);
5  void affiche(short *v, short tail);
6  void permute(short *v, short tail);

```

```

7
8 void main()
9 {
10     short v[7], i;
11     printf("\nRemplir");
12     for (i=0; i<7; ++i)
13     {
14         scanf("%hd", v+i);
15     }
16     printf("\nAvant\n");
17     affiche(v, 7);
18     permute(v, 7);
19     printf("\nAprès\n");
20     affiche(v, 7);
21 }
22
23 void affiche(short *v, short tail)
24 {
25     short i;
26     for (i=0; i<tail; i++)
27         printf("%d_", *(v+i));    /* les parenthesises sont obligatoires */
28     /* on aurait pu écrire cela *v++ au lieu de *(v+i) */
29 }
30 permute(short v[], short tail)
31 {
32     short *deb, *fin, tmp;
33     for (deb=v, fin=v+tail-1; deb<fin; deb++, fin--)
34     {
35         tmp=*deb;
36         *deb=*fin;
37         *fin=tmp;
38         /* si la fonction échange avait été utilisée toutes les instructions auraient été remplacées par :
39         échange(deb, fin) */
40     }
41 }

```

A.5.4 Permutation d'un vecteur d'entiers : variante 3

Enoncé Refaire l'exercice précédent avec l'utilisation de la fonction permettant d'échanger 2 entiers.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void affiche(short v[], short n);
5 void permutation(short v[], short n);
6 void échangeEntier(short *p1, short *p2);
7
8 int main()
9 {
10     printf("\nProgramme qui remplit un vecteur de 7 entiers, l'affiche, le permute et le réaffiche");
11     static short v[7];
12     short i;
13     for (i=0; i<7; i++)
14     {
15         printf("\nEntrez un entier pour la %dème position\n", i+1);
16         scanf("%hd", &v[i]);
17     }
18     affiche(v, 7);
19     printf("\n");
20     permutation(v, 7);
21     affiche(v, 7);
22     return 0;
23 }
24
25 void échangeEntier(short *p1, short *p2)
26 {
27     short tmp;
28     tmp=*p1;
29     *p1=*p2;
30     *p2=tmp;

```

```

31 }
32
33 void affiche(short *v, short n)
34 {
35     short i;
36     for (i=0; i<n; i++)
37     {
38         printf("%d", v[i]);
39     }
40 }
41
42 void permutation(short *v, short n)
43 {
44     short i, a, b, *p1, *p2;
45     p1=v;
46     p2=v;
47     p2+=(n-1);
48     for (i=0; i<n/2; i++)
49     {
50         a=*p1;
51         b=*p2;
52         echangeEntier(&a, &b);
53         *p1=a;
54         *p2=b;
55         p2--;
56         p1++;
57     }
58 }

```

A.6 Les chaines de caractères

A.6.1 Fonction string

Ecrire la fonction gets qui permet d'introduire une chaine au clavier (gets(message,25);). Attention, on ignore tous les caractères de contrôle sauf le backspace ('\b').

Ecrire puts qui affiche la chaine de caractères et puis passe à la ligne (puts("Bob");). Pour afficher, on peut se servir de printf("%c") ou putchar.

Les fonctions gets et puts se retrouvent dans la bibliothèque "stdio.h".

```

1 void puts(char *ch)
2 {
3     while(*ch) /*ch doit etre different du code ASCII 0 donc doit etre vrai*/
4     {
5         putchar(*ch++);
6     }
7     putchar('\n');
8     /*quand il a fini, il pointe '\0'*/
9 }

```

```

1 /*Solution du prof*/
2 void gets(char *ch, short tail)
3 {
4     short i=1;
5     while((i<tail)&&((*ch=getche())!='\n'))
6     {
7         if(*ch=='\b')
8         {
9             if(i>1)
10            {
11                ch--;
12                i--;
13            }
14            elseif(!isctrl(*ch))
15            {
16                ch++;

```



```

17         i++;
18     }
19 }
20 }
21 }

```

A.6.2 Fonction string copy

Cette fonction strcpy(ch2,ch1); qui copie la chaine 1 dans la chaine 2. Si la chaine 2 n'est pas assez grande, elle déborde.

```

1  /*Solution du prof*/
2  void strcpy(char *ch2,char *ch1)      /*se trouve dans "string.h"*/
3  {
4      while(*ch1)
5      {
6          *ch2++=*ch1++;
7      }
8      *ch2='\0';
9  }

```

```

1  /*Autre solution*/
2  void strcpy(char *ch2,char *ch1)
3  {
4      while(*ch2++=*ch1++);
5  }

```

A.6.3 Fonction string cat

La fonction strcat(ch2,ch1); qui ajoute la chaine 1 à la fin de la chaine 2. Il n'y a toujours pas de contrôle.

```

1  /*Solution du prof*/
2  void strcat(char *ch2,char *ch1)      /*se trouve dans "string.h"*/
3  {
4      while(*ch2++); /*le pointeur s'arrete juste apr s le \0*/
5      strcpy(--ch2,ch1);
6  }

```

A.6.4 Fonction string length

La fonction strlen(ch); qui renvoie la longueur de la chaine. Il n'y a toujours pas de contrôle.

```

1  /*Solution du prof*/
2  short strlen(char *ch) /*se trouve dans "string.h"*/
3  {
4      short cpt=0;
5      while(*ch)
6      {
7          ++cpt;
8          ++ch;
9      }
10     return(cpt);
11 }

```

A.6.5 Fonction string compare

La fonction strcmp(ch1,ch2) permet de comparer 2 chaines. Cela renvoie la différence des codes ASCII des 2 premiers caractères différents.

Ex:

```

strcmp("ABCD","ABEFG");
printf(char *c) /*renvoie -2*/

```

```

strcmp("abc","abc"); /*renvoie 0*/

```

```

1  /* Solution du prof */
2  short strcmp(char *ch1, char *ch2)      /* se trouve dans "string.h" */
3  {
4      while ((*ch1 == *ch2) && (*ch1))
5      {
6          ch1++;
7          ch2++;
8      }
9      return (*ch1 - *ch2);
10 }

```

A.6.6 Recherche de caractère

Fonction qui recherche un caractère dans une chaîne de caractères et qui renvoie où il se trouve (renvoie une adresse).

```

1  char *strchr(char *ch, char c)
2  {
3      while ((*ch) && (*ch != c))
4      {
5          ++ch;
6      }
7      return (*ch == c ? ch : NULL);
8  }
9
10 /* a l'utilisation on pourrait faire
11 static char mes[] = "Banane";
12 char *p;
13 p = strchr(mes, 'a'); */

```

A.7 argc et argv

A.7.1 Pyramide d'étoiles

Ecrire un programme qui affiche une pyramide de n lignes d'étoiles.

pyramide 4 donne

```

*
***
*****
*****

```

A.8 Les fichiers

A.8.1 Réécrire la commande type du DOS

```

1  type fl.txt "enter" ==> affiche le contenu du fichier    l'cran

```

```

1  /* Solution du prof */
2  #include "stdio.h"
3  #include "stdlib.h"
4
5  void main(int argc, char *argv[])
6  {
7      FILE *f;
8      short c;
9      if (argc != 2)
10     {
11         printf("\nMal_lance");
12         exit(1);
13     }
14     if ((f = fopen(argv[1], "r")) == NULL)
15     {
16         printf("\nProbleme_ouverture_du_fichier");
17         exit(2);

```

```

18     }
19     while((c=fgetc(f))!=EOF)
20         putchar(c);
21     exit(0);
22 }

```

A.8.2 Réécrire la commande copy du DOS

copy f1.txt f2.txt "enter" copie f1 dans f2.

```

1  /* Solution du prof */
2  /* copy f1 f2 caractere par caractere */
3
4  #include "stdio.h"
5  #include "stdlib.h"
6
7  void erreur(char *nomFichier);
8
9  void main(int argc, char *argv[])
10 {
11     FILE *f1, *f2;
12     short c;
13     if(argc!=3)
14     {
15         printf("\nMal_lance");
16         exit(1);
17     }
18     if((f1=fopen(argv[1], "r"))==NULL)
19         erreur(argv[1]);
20     if(!(f2=fopen(argv[2], "r")))
21         erreur(argv[2]);
22     while((c=fgetc(f1))!=EOF)
23         fputc(c, f2);
24     exit(0);
25 }
26
27 void erreur(char *nomFichier)
28 {
29     printf("\nProbleme_ouverture_du_fichier_%s", nomFichier);
30     exit(3);
31 }

```

A.8.3 Compter le nombre de mots dans un fichier

```

1  /* Solution du prof */
2  #include "stdio.h"
3  #include "stdlib.h"
4
5  void erreur(char *fnom);
6
7  void main(int argc, char *argv[])
8  {
9      FILE *f;
10     char mot[60];
11     short cpt=0;
12     if(argc!=2)
13     {
14         fprintf(stderr, "\nMal_lance");
15         exit(2);
16     }
17     if(!(f=fopen(argv[1], "r")))
18         erreur(argv[1]);
19     while(fscanf(f, "%s", mot)!=EOF)
20         cpt++;
21     fprintf(stdout, "\nle_fichier_%s_a_%d_mots", argv[1], cpt);
22     exit(0);
23 }

```

A.8.4 Compter le nombre de lignes dans un fichier

```

1  /* Solution du prof */
2  #include "stdio.h"
3  #include "stdlib.h"
4
5  void erreur(char *fnom);
6
7  void main(int argc, char *argv[])
8  {
9      FILE *f;
10     char ligne[252];
11     short cpt=0;
12     if(argc!=2)
13     {
14         fprintf(stderr, "\nMal_lance");
15         exit(2);
16     }
17     if(!(f=fopen(argv[1], "r")))
18         erreur(argv[1]);
19     while(fgets(ligne, 252, f)!=NULL)
20         cpt++;
21     fprintf(stdout, "\nle_fichier_%s_a_%d_lignes", argv[1], cpt);
22     exit(0);
23 }

```

A.8.5 Compter les caractères affichés par bloc de 512

```

1  /* Solution du prof */
2  #include "stdio.h"
3  #include "stdlib.h"
4
5  void erreur(char *fnom);
6
7  void main(int argc, char *argv[])
8  {
9      FILE *f;
10     char bloc[512];
11     short cpt=0, n;
12     if(argc!=2)
13     {
14         fprintf(stderr, "\nMal_lance");
15         exit(2);
16     }
17     if(!(f=fopen(argv[1], "r")))
18         erreur(argv[1]);
19     while(n=fread(bloc, 1, 512, f))
20         cpt++;
21     fprintf(stdout, "\nle_fichier_%s_a_%d_caracteres", argv[1], cpt);
22     exit(0);
23 }

```

A.8.6 Copier fichier

/*fcopy[f1[f2]]*/

```

1  /* Solution du prof */
2  #include "stdio.h"
3  #include "stdlib.h"
4
5  void erreur(char *fnom);
6
7  void main(int argc, char *argv[])
8  {
9      short c;
10     FILE *f1, *f2;
11     if(argc>3)
12     {
13         fprintf(stdout, "\nMal_lance");
14         exit(1);
15     }
16     if(argc==3)
17     {

```

```

18     if (!(f2=fopen(argv[2], "w")))
19         erreur(argv[2]);
20     }
21     else
22         f2=stdout;
23     if (argc>=2)
24     {
25         if (!(f1=fopen(argv[1], "r")))
26             erreur(argv[1]);
27     }
28     else
29         f1=stdin;
30     while((c=fgetc(f1))!=EOF)
31         fputc(c, f2);
32     exit(0);
33 }

```

A.8.7 Copier avec tous les messages d'erreur

copyerr f1 f2 travail_par_bloc_de_10000_bytes

```

1  /* Solution du prof */
2  #include "stdio.h"
3  #include "stdlib.h"
4
5  void erreur(char *fnom);
6
7  void main(int argc, char *argv[])
8  {
9      char bloc[10000];
10     short n;
11     FILE *f1, *f2;
12     if (argc!=3)
13     {
14         fprintf(stdout, "\nMal_lance");
15         exit(1);
16     }
17     if (!(f1=fopen(argv[1], "r")))
18         erreur(argv[1]);
19     if (!(f2=fopen(argv[2], "w")))
20         erreur(argv[2]);
21     while((n=fread(bloc, 1, 10000, f1))==10000)
22         if (fwrite(bloc, 1, 10000, f2)!=10000)
23             erreur(argv[2]);
24     if (ferror(f1))
25         erreur(argv[1]);
26     else
27     {
28         if (fwrite(bloc, 1, n, f2)!=n)
29             erreur(argv[2]);
30     }
31     exit(0);
32 }
33
34 void erreur(char *fnom)
35 {
36     perror(fnom);
37     exit(2);
38 }

```

Attention, un * de fichier peut recevoir l'adresse de stdout mais pas l'inverse.

A.8.8 Copier fichier dans racine du disque autrepart

Copier le fichier se trouvant c:\autoexec.bat dans c:\autoexec.bak

```

1  Solution du prof
2
3  #include "stdio.h"

```

```

4 #include "stdlib.h"
5
6 void erreur(char *nomfic);
7
8 void main()
9 {
10 FILE *f1, *f2;
11 short c;
12 if (! (f1=fopen("c:\\autoexec.bat","r")))
13     erreur("c:\\autoexec.bat");
14 if (! (f2=fopen("c:\\autoexec.bak","w")))
15     erreur("c:\\autoexec.bak");
16 while (! feof(f1))
17     fputc(fgetc(f1),f2);          /*caractere lu dans f1 est passe a f2*/
18 exit(0);
19 }

```

A.9 La récursivité

A.9.1 Ecrire la fonction factorielle en récursivité

La fonction factorielle :

```

n! { =1 si n=0
    { =n(n-1)! sinon

```

```

1 /* Solution du prof */
2 short factoriel(short n)
3 {
4     return(n?n*factoriel(n-1):1);
5 }

```

A.9.2 Pgcd

Selon la méthode d'Euclide (mais en récursif):

On divise l'entier par l'autre, si le reste est différent de 0, on divise le diviseur par le reste et si c'est 0, le pgcd est le diviseur.

```

1 short pgcd(short a, short b)
2 {
3     short reste;
4     if ((a%b)==0)
5         return(b);
6 }

```

```

1 /* Solution du prof */
2 short pgcd(short a, short b)
3 /* a et b >=1 et a >=b */
4 {
5     return(a%b?pgcd(b,a%b):b);
6 }

```

A.9.3 Pgcd en utilisant d'autres formules

Créer un programme qui calcule le pgcd en récursif selon les formules suivants

```

pgcd(a,b){ =pgcd(a,b-a) si b>a
           { =pgcd(a-b,b) si a>b
           { =a=b si a=b

```

```

1  /* Solution du prof */
2  short pgcd(short a, short b)
3  /* a et b >= 1 */
4  {
5      if (a > b)
6          return (pgcd(a-b, b));
7      else
8          return (a < b ? pgcd(a, b-a) : a);
9  }

```

A.9.4 Fibonnaci

F0=0
F1=1
Fn=Fn-1 +Fn-2

```

1  /* Solution du prof */
2  short fibo(short n)
3  /* n >= 1 */
4  {
5      return ((n==0) || (n==1) ? n : fibo(n-1) + fibo(n-2));
6  }

```

A.9.5 Combinaison

Analyse combinatoire: nombre de façon de choisir i objets par j.
Travailler selon le triangle de pascal.

Il faut en plus créer un programme principal avec le nombre de colonne du triangle de Pascal.

```

1  /* Solution du prof */
2  short combi(short i, short j)
3  /* i et j >= 0 et i <= j */
4  {
5      return ((i==j) || (i==0) ? 1 : combi(i, j-1) + combi(i-1, j-1));
6  }

```

```

1  /* Solution du prof */
2  /* Triangle nb */
3  #include "stdio.h"
4  #include "stdlib.h"
5
6  short combi(short i, short j);
7
8  void main(int argc, char *argv[])
9  {
10     short nb_ligne, i, j;
11     if (argc != 2)
12     {
13         fprintf(stderr, "\nPas d'argument");
14         exit(1);
15     }
16     if (((nb_ligne = atoi(argv[1])) <= 0) || (nb_ligne > 10))
17     {
18         fprintf(stderr, "\nMauvais argument");
19         exit(2);
20     }
21     for (j=0; j<=nb_ligne; ++j)
22     {
23         for (i=0; i<=j; ++i)
24             printf("%4d", combi(i, j));
25         putchar('\n');
26     }
27     exit(0);
28 }

```

A.10 Les tables à plusieurs dimensions

A.10.1 Remplir une table

Table à 3 dimensions de dimensions variables de maximum 20 sur 20 sur 20.

Remplir la table en appelant une fonction qui remplit une matrice.

Créer une fonction pour afficher la table à 3 dimensions.

Fonction qui reçoit la table à 3 dimensions et qui effectue une permutation cyclique des plans vers le bas.

Ensuite on réaffiche à nouveau.

On peut également créer une fonction qui décale les lignes dans une matrice (amélioration).

Fonction qui décale dans un vecteur ou dans une ligne (amélioration).

```

1  #include "stdio.h"
2
3  void remplir(tab[prof][lig][col]);
4
5  void main()
6  {
7      printf("\nProgramme qui crée une table à 3 dimensions (max 20 sur 20 sur 20)");
8      short lig, col, prof;
9      while((lig < 1) || (lig > 20) || (col < 1) || (col > 20) || (prof < 1) || (prof > 20))
10     {
11         printf("\nEntrez le nombre de ligne de votre table");
12         scanf("%hd", &lig);
13         printf("\nEntrez le nombre de colonnes de votre table");
14         scanf("%hd", &col);
15         printf("\nEntrez la profondeur de votre table");
16         scanf("%hd", &prof);
17     }
18     short t[prof][lig][col];
19     printf("%d %d %d", lig, col, prof);
20 }
21
22 void remplir(tab[prof][lig][col])
23 {
24     short i, j, k;
25     for(k=1; k<prof; k++)
26     {
27         printf("\nPour la matrice %d", k);
28         for(i=1; i<lig; i++)
29         {
30             printf("\nPour la ligne %d", i);
31             for(j=1; j<col; j++)
32             {
33                 printf("\nPour la colonne %d", j);
34                 scanf("%hd", &tab[k][i][j]);
35             }
36         }
37     }
38 }
```

A.10.2 Tri à bulles 1

A COMPLETER

```

1  /* Solution du prof */
2  /* plusieurs chaines - tables a 2 dimensions */
3  #include "stdio.h"
4  #include "string.h"
5
6  void affiche(char t[][21], short n);
7  void triBulles(char tab[][21], short n);
8
9  void main(){
10     char tab[10][21];
11     /* si on veut initialiser les 10 mots;
```



```

12  char tab[10][21]={ "Ernest", "Bob" ,... };*/
13  short i;
14  /*entrer les 10 mots*/
15  for(i=0;i<10;++i)
16      gets(tab[i]);
17  /*afficher les mots*/
18  affiche(tab,10);
19  triBulles(tab,10);
20  printf("\nAprès\n");
21  affiche(tab,10);
22  }
23
24  void affiche(char t[][21],short n){
25      short i;
26      for(i=0;i<n;++i)
27          puts(t[i]);
28  }
29
30  void triBulles(char tab[][21],short n){
31      char tmp[21];
32      short i,j;
33      for(i=n-1;i!=0;--i)
34          for(j=0;j<i;++j)
35              if(strcmp(tab[j],tab[j+1])>0){
36                  strcpy(tmp,tab[j]);
37                  strcpy(tab[j],tab[j+1]);
38                  strcpy(tab[j+1],tmp);
39              }
40  }

```

A.10.3 Tri à bulles 2

```

1  /*Solution du prof*/
2  /*plusieurs chaines - vecteur de pointeurs*/
3  #include "stdio.h"
4  #include "string.h"
5
6  void affiche(char *v[],short n);
7  void triBulles(char *v[],short n);
8
9  void main(){
10     static char *v[10]={ "Ernest", "Bob" ,... };
11     /*afficher les mots*/
12     printf("\nAvant\n");
13     affiche(v,10);
14     triBulles(v,10);
15     printf("\nAprès\n");
16     affiche(v,10);
17 }
18
19 void affiche(char *v[],short n){
20     short i;
21     for(i=0;i<n;++i)
22         puts(v[i]);
23 }
24
25 void triBulles(char *v[],short n){
26     char *tmp;
27     short i,j;
28     for(i=n-1;i!=0;--i)
29         for(j=0;j<i;++j)
30             if(strcmp(v[j],v[j+1])>0){
31                 tmp=v[j];
32                 v[j]=v[j+1];
33                 v[j+1]=tmp;
34             }
35 }

```

A.10.4 Tri à bulles 3

Refaire l'exercice du tri à bulles en utilisant des pointeurs de pointeurs

```

1  /*Solution du prof*/
2  /*plusieurs chaines - vecteur de pointeurs et pointeur de pointeur*/
3  #include "stdio.h"
4  #include "string.h"
5
6  void affiche(char *v[], short n);
7  void triBulles(char *v[], short n);
8
9  void main(){
10     static char *v[10]={ "Ernest", "Bob", ... };
11     /*afficher les mots*/
12     printf("\nAvant\n");
13     affiche(v,10);
14     triBulles(v,10);
15     printf("\nAprès\n");
16     affiche(v,10);
17 }
18
19 void affiche(char *v[], short n){
20     /*le parametre *v[] peut etre remplace par **v*/
21     short i;
22     for(i=0;i<n;++i)
23         puts(*(v+i)); /*identique a puts(*v++);*/
24 }
25
26 void triBulles(char *v[], short n){
27     /*le parametre char *v[] peut etre remplace par char **v*/
28     char *tmp;
29     short i, j;
30     for(i=n-1; i!=0; --i)
31         for(j=0; j<i; ++j)
32             if(strcmp(*(v+j), *(v+j+1))>0){
33                 tmp=*(v+j), *(v+j)=*(v+j+1), *(v+j+1)=tmp;
34             }
35 }

```

A.10.5 Tri à bulles 4

Refaire le tri à bulles en utilisant un pointeur qui retient jusqu'où c'est trié.

```

1  /*Solution du prof*/
2  /*plusieurs chaines - vecteur de pointeurs et pointeur de pointeur*/
3  #include "stdio.h"
4  #include "string.h"
5
6  void affiche(char *v[], short n);
7  void triBulles(char *v[], short n);
8
9  void main(){
10     static char *v[10]={ "Ernest", "Bob", ... };
11     /*afficher les mots*/
12     printf("\nAvant\n");
13     affiche(v,10);
14     triBulles(v,10);
15     printf("\nAprès\n");
16     affiche(v,10);
17 }
18
19 void affiche(char *v[], short n){
20     /*le parametre *v[] peut etre remplace par **v*/
21     short i;
22     for(i=0;i<n;++i)
23         puts(*(v+i)); /*identique a puts(*v++);*/
24 }
25
26 void triBulles(char *v[], short n){
27     /*le parametre char *v[] peut etre remplace par char **v*/

```



```

44     quit();
45     break;
46 }
47 else
48     if(f)
49         quit();
50     else
51         if((f=fopen(*argv,"r"))==NULL)
52         {
53             fprintf(stderr,".....");
54             exit(1);
55         }
56 }
57
58 if(f)
59 {
60     while(fgets(ligne,132,f)!=NULL)
61     {
62         i++;
63         if(maj)
64             upper(ligne);
65         printf("\n%3d: %s",i,ligne);
66         if(nb_ligne&&(i%nb_ligne==0))
67         {
68             printf(".....une_touche_pour_continuer...");
69             getche();
70             clrscr();          /*clear screen dans conio.h*/
71         }
72         else
73             quit();
74         exit(0);
75     }
76 }
77 }
78
79 /*transforme tous les caracteres en majuscule*/
80 void upper(char *ligne)
81 {
82     while(*ligne)
83     {
84         if(islower(*ligne))
85             *ligne -=32;
86         ++ligne;
87     }
88 }

```

A.11 Structures et unions

A.11.1 Chèque 1

On travaille avec une structure chèque avec le numéro du chèque (entier), la date du chèque (jour, mois année) entier, le bénéficiaire (21 caractères) et le montant du chèque (double).

Fonction qui remplit un seul chèque (entrer les données au clavier) et on lui passe le numéro à y mettre.

Fonction qui affiche un chèque et on lui passe le numéro du chèque.

Programme avec deux chèques, remplir le chèque 1 avec le numéro 1 et le chèque 2 avec le numéro 2 et puis les afficher.

```

1  /*Solution du prof*/
2  #include
3
4  struct cheque{
5      short num;
6      short j,m,a;
7      char benef[21];
8      long montant;

```

```

9         };
10 void remplir(struct cheque *pch,short n);
11 void afficher(struct cheque ch);
12
13 void main ()
14 {
15     struct cheque ch1, ch2;
16     printf("\nRemplir_cheque_1");
17     remplir(&ch1,1);
18     printf("\nRemplir_cheque_2");
19     remplir(&ch2,2);
20     printf("\nLes_cheques_sont_");
21     afficher(ch1);
22     afficher(ch2);
23 }
24
25 void remplir(struct cheque *pch, short n)
26 {
27     pch->num=n;
28     printf("Date_(Jour_Mois_Anee:");
29     scanf("%hd_%hd_%hd", &pch->j, &pch->m, &pch->a);
30     printf("\nBeneficiaire");
31     gets(pch->benef);
32     printf("\nMontant");
33     scanf("%ld", &pch->montant);
34 }
35
36 void afficher(struct cheque ch)
37 {
38     printf("\nCheque_n   _%d_du_%d/%d/%d_pour_%s_de_%d_euros", ch.num, ch.j, ch.m, ch.o, ch.benef,
39     ch.montant);
40 }

```

A.11.2 Chèque 2

On travaille avec un vecteur de 25 chèques à gérer avec un menu.

Ajout (ajouter un chèque)

Voir (affiche tous les chèques remplis)

Quitter (quitter le programme)

```

1  /* Solution du prof */
2  #include
3
4  struct cheque{
5      short num;
6      short j,m,a;
7      char benef[21];
8      long montant;
9  };
10 void remplir(struct cheque *pch,short n);
11 void afficher(struct cheque ch);
12 char menu();
13
14 void main ()
15 {
16     struct cheque v[25];
17     char choix;
18     short cpt=0,i /*pour savoir ou on est dans le vecteur de cheques*/
19     while((choix=menu())!='Q')
20     {
21         switch(choix)
22         {
23             case 'A':
24                 if(cpt<25)
25                     remplir(&v[cpt],++cpt); /*&v[cpt] peut s'ecrire v+cpt*/
26                 else
27                     printf("\nPlus_de_place");
28                 break;

```

```

29     case 'V':
30         if(cpt)
31             for(i=0;i<cpt;++i)
32                 afficher(*(v+i));          /**(v+i) peut s'ecrire v[i]**/
33         else
34             printf("\nPas de cheque");
35         break;
36     default:
37         printf("\nPas valable");
38         break;
39     }
40 }
41 }
42
43 void remplir(struct cheque *pch, short n)
44 {
45     pch->num=n;
46     printf("Date_(Jour_Mois_Anee:");
47     scanf("%hd_%hd_%hd", &pch->j, &pch->m, &pch->a);
48     printf("\nBeneficiaire");
49     gets(pch->benef);
50     printf("\nMontant");
51     scanf("%ld", &pch->montant);
52 }
53
54 void afficher(struct cheque ch)
55 {
56     printf("\nCheque_ n   %d_du %d/%d/%d_pour %s_de %d_euros", ch.num, ch.j, ch.m, ch.o, ch.benef,
57     ch.montant);
58 }
59
60 char menu()
61 {
62     /*renvoyer lettre et transformer les minuscules en majuscules*/
63 }

```

A.11.3 Chèque 3

Soit un fichier de 50 chèques, créer avec 50 chèques à 0 (chaque numéro(premier champ) des chèques = 0). Il y a 10 chèques par client donc 5 clients.

Entrer un numéro de client entre 1 et 5 avec un petit menu.

1. Nouveau chèque
2. Voir chèque
3. Nouveau carnet (si 10 chèques d'un client sont remplis, on efface tous ses chèques et on recommence un chèque)
0. Quitter

```

1  /*exemple de creation de vecteur de 50 cheques*/
2  void main()
3  {
4      static struct cheque vch[50]; /*tous les numeros sont a zero*/
5      FILE *pf;
6      int i;
7      pf=fopen("cheque.dat", "w");
8      fwrite(&ch, sizeof(struct cheque), 50, pf);
9      fclose(pf);          /*creer 50 cheques*/
10 }

```

```

1  /*Solution du prof*/
2  /*Travail avec un fichier de 50 cheques pour 5 clients + menu*/
3  #include
4
5  struct cheque{
6      short num;
7      short j,m,a;
8      char benef[21];
9      long montant;

```

```

10         };
11 void remplir(struct cheque *pch,short n);
12 void afficher(struct cheque ch);
13 char menu();
14
15 void main()
16 {
17     FILE *pf;
18     short numcli,choix,cpt,i;
19     struct cheque v[10];
20     pf=fopen("cheques.dat","r+w");
21     printf("\nNumero_client");
22     scanf("%hd",&numcli);
23     while ((numcli>=1)&&(numcli<=5))
24     {
25         fseek(pf,(numcli-1)*10*sizeof(struct cheque),0);
26         fread(v,sizeof(struct cheque),10,pf);
27         cpt=0;
28         while((cpt<10)&&((v+cpt)->num))
29             cpt++;
30
31         while(choix==menu())
32         {
33             switch (choix)
34             {
35                 case 1:
36                     if(cpt<10)
37                         remplir(v+cpt,++cpt);
38                     else
39                         printf("\nCarnet_rempli");
40                     break;
41                 case 2:
42                     if(cpt)
43                         for(i=0;i<cpt;++i)
44                             afficher(*(v+i));
45                     else
46                         printf("\nCarnet_vide");
47                     break;
48                 case 3:
49                     if(cpt=10)
50                         for(i=0;i<10;++i)
51                         {
52                             (v+i)->num=0;
53                         }
54                     else
55                         printf("\nCarnet_pas_rempli");
56                     break;
57                 default:
58                     printf("\nQuoi?");
59                     break;
60             }
61         }
62         fseek(pf,(numcli-1)*10*sizeof(struct cheque),0);
63         /*fseek(pf,-10*sizeof(struct cheque),1);*/ /*autre_facon_d'ecrire_ligne_precedente*/
64         fwrite(v,sizeof(struct cheque),10,pf);
65         printf("\nNumero_client:");
66         scanf("%hd",&numcli);
67     }
68 }
69
70 void _remplir(struct cheque_*pch,_short n)
71 {
72     pch->num=n;
73     printf("Date (Jour Mois Annee:");
74     scanf("%hd %hd %hd",&pch->j,&pch->m,&pch->a);
75     printf("\nBeneficiaire");
76     gets(pch->benef);
77     printf("\nMontant");
78     scanf("%ld",&pch->montant);
79 }
80

```

```

81 void _afficher (struct _cheque _ch)
82 _{
83 _printf("\nCheque n   %d du %d/%d/%d pour %s de %d euros", _ch.num, ch.j, ch.m, ch.o, ch.benef,
84 _ch.montant);
85 _}
86
87 _char _menu()
88 _{
89
90 _}

```

A.11.4 Unions

Soit un programme qui permet de lire un fichier de nom: soit de short soit de float.
 Pour le lancer, je fais: prog f fichier (ou prog s fichier).

```

1  /*Solution du prof*/
2  /*TYPE
3  #include ....
4
5  void main(int argc, char **argv)
6  {
7      union {          short s;
8                      float f;
9      } u;    /*pas besoin de donner un nom car cree dans main*/
10 FILE *f;
11
12 if ((argc!=3)||(*(*(argv+1)+1)!='\0')||( !( f=fopen (*(argv+2),"r" ) )))
13 {
14     printf("stderr","...");
15     _exit(1);
16 _}
17
18 _else
19 _{
20     _switch (**(argv+1))
21     _{
22     _case _'f':
23     _case _'F':
24     _while (fread(&u.f, sizeof(float), 1, f))
25     _printf("%f ", u.f); _/_u comme union*/
26     _break;
27
28     _case _'s':
29     _case _'S':
30     _while (fread(&u.s, 2, 1, f))
31     _printf("%d ", u.s);
32     _break;
33
34     _default:
35     _fprintf(stderr, _" .....");
36     _exit(1);
37     _break;
38     _}
39
40 _}
41 _exit(0);
42 _}

```

A.12 Gestion dynamique de la mémoire

A.12.1 Exercice 1

Fonction qui permet de créer une chaîne de char de la bonne taille de façon dynamique.

```

1  /*ancienne facon de travailler*/
2  maint()

```



```

3 {
4   char nom[30];
5   printf("Entrez nom");
6   gest(nom);
7 }
8
9 /*nouvelle facon de travailler*/
10 main()
11 {
12   char *nom;
13   printf("Entrez nom");
14   nom=lireChaine();      /*la chaine aura la bonne taille*/
15 }

```

A.12.2 Exercice2

On entre un tableau de 10 noms pour lequel chaque nom est inscrit de façon dynamique. Même exercice mais on entre 10 de façon dynamique, on affiche, on trie puis on réaffiche.

A.13 Les listes chaînées

A.13.1 Exercice 1

Soit une liste simplement chaînée de 6 composantes obligatoirement où chaque composant contient un numéro et un texte.

Tout d'abord, il faut créer la liste et le numéro pour la première soit 1 et le texte, Mot 1 Créer la liste, l'afficher et la détruire.

A.13.2 Exercice 2

Même exercice mais avec des fonctions pour l'ajout, suppression.

A.13.3 Exercice 3

Même exercice mais en récursif.

```

1  /*REVISION*/
2  void main()
3  {
4      static char *mes[5]={" Jean", " Lundi", " Bob", " Vendredi", " Marie" };
5      f(mes);
6  }
7
8  void f(char **ch)
9  {
10     printf("%c",**ch);           /*J*/
11     printf("%c",** (ch+1));      /*L*/
12     printf("%c",** (*ch+1));     /*e*/
13     printf("%c",** ch+1);        /*K du J de Jean*/
14     ++ch;
15     printf("%c",**(++ch));        /*B*/
16     printf("%c",**(++ch+1));     /*M*/
17     printf("%c",** (*ch+1));     /*M*/
18     printf("%c",** ((ch--)+1));  /*M puis --*/
19     printf("%c",** (*ch+1));     /*o*/
20     printf("%c",** (ch++)+1);    /*c du b de Bob puis ++*/
21     printf("%c",** (ch+1));      /*M*/
22
23     printf("%s", *ch);           /*Vendredi*/
24     printf("%s", *ch+1);        /*endredi*/
25     printf("%s", *(ch+1));      /*Marie*/
26
27     printf("%c",** ((ch+1)+1)+1); /*b du a de Marie*/
28 }

```


Appendix B

Priorité et associativité des opérateurs

Type/opérateur	Priorité	Opérateur	Groupage	Description
Primaire	1	(,) [,] -> .	de gauche à droite	accéder à un membre d'une structure
Monodique	2	! ~ ++ -- - (type) & sizeof *	de droite à gauche	ont besoin d'une opérande
Arithmétique	3	* / %	de gauche à droite	
Arithmétique	4	+ -	de gauche à droite	
Décalage	5	<< >>	de gauche à droite	
Relationnel	6	< <= > >=	de gauche à droite	
Relationnel	7	== !=	de gauche à droite	
Logique binaire	8	&	de gauche à droite	
	9	exposant	de gauche à droite	
	10		de gauche à droite	
Logique	11	&&	de gauche à droite	
	12		de gauche à droite	
Conditionnel	13	? :	de droite à gauche si on les combine	
Affectation	14	= *= /= %= += -= <<= >>= & =	de droite à gauche	

Appendix C

Résumé sur l'initialisation

C.1 Paramètres

On ne peut pas les initialiser.

```
1 void fonction(short a=5) /*pas correct pour les parametres le "=5" doit etre enleve*/
```

C.2 Variables

C.2.1 auto ou register

Il y a 2 types: auto et register. On peut initialiser les deux.

```
1 auto short a=3;  
2 auto short b=a+2;
```

C.2.2 static

Une seule manière de les initialiser: avec une constante. Si on n'initialise pas, elle vaut 0.

```
1 static short n=8;
```