REPUBLIQUE ISLAMIQUE DE MAURITANIE

Honneur-Fraternité-Justice

Ministère de l'Emploi et de la Formation Professionnelle

CENTRE SUPERIEUR D'ENSEIGNEMENT TECHNIQUE

CENTRE DE DEVELOPPEMENT DES COMPETENCES

SECTEUR : SERVICES

Cours Introduction au langage de programmation C

NIVEAU : BTS - Brevet de Technicien Supérieure

SPÉCIALITE : Réseau Informatique

Elaborer par: El Hacen Mohamed Saleck Ramdhane

TABLE DES MATIERES

Présentation et Objectif du Module

Ce cours est une introduction à la programmation C est destiné aux étudiants de Ecole d'Enseignement Technique et de Formation Professionnelle Commerciale, spécialité réseau informatique niveau BTS Le but de ces notes de cours est de permettre aux élevés, d'acquérir les éléments fondamentaux du langage C. présentant donc chaque notion selon une gradation des difficultés et ne cherche pas à être exhaustif. Elle comporte de nombreux exemples, ainsi que des exercices. Ces notes de cours sont composées de sept Thèmes.

Objectifs généraux :

- Découvrir le fonctionnement de la mémoire, des variables, des conditions et des boucles.
- Mobiliser les notions de base pour organiser son code ;
- Se familiariser avec l'environnement de programmation C
- Connaître l'aspect syntaxique et sémantique du langage C
- Manipuler les pointeurs, fonctions et les tableaux ;
- Ecrire des programmes corrects partant des programmes simples à des programmes complexes.

Objectifs spécifiques :

La méthodologie adoptée afin de traiter ce module se base principalement sur des cours magistraux, des travaux dirigés et des travaux pratiques. On vise par ce module la réalisation des objectifs spécifiques suivants :

- Connaître les concepts nécessaires pour la programmation C (identifiants, déclaration, opérateurs, instructions, structures…),
- Apprendre à écrire des programmes en langage C et savoir les compiler et les exécuter sur la machine à l'aide de l' IDE code blocks.
- Maîtriser la manipulation des structures conditionnelles et itératives.
- Acquérir des connaissances et de savoir-faire relatif à la résolution des problèmes complexes en adoptant le langage de programmation C.

Plan de Déroulement des Cours

Module : Programmation en C Durée : 45 h

Thèmes (éléments de compétences/objectif second niveau)	Déroulement des leçons théoriques et pratiques	Durée en Heures	Stratégies d'enseignement	Équipements Outillages & matériels didactiques	Locaux	Indicateurs particuliers de performance	Évaluation
Thème 1 **Introduction Général**	**<u>Leçon 1</u>** **<u>Contenu</u>** - **Introduction** (définition e structure d'un programme, les directives de précompilation, la fonction main, la procédure de compilation, les - **Variables et les constants** (déclarations de variables et les constants, le type de variables) - - **Activités d'apprentissages** Travaux pratiques.	2 H 2H	- Exposé - discussion -Questionnement - Animation Travail individuel et collectif	-Vidéo Projecteur -Ecran - Tableau Blanc - Marqueurs colorés -effaceur -Pc avec codeblocks	Laboratoire	-Découvrir les concepts nécessaires pour la programmation C (identifiants, déclaration, opérateurs, instructions) - Ecrire des programmes en langage C	E. formative

	Leçon 2 **Contenu** - **Les opérations** (arithmétique, d'affectation, d'incrémentation et de décrémentation, Les opérateurs de comparaison et les opérateurs logiques). - **Les fonctions d'affichage et de saisie** - **Les fonctions de manipulation de chaine de caractères** - **Activités d'apprentissages** Travaux pratiques	2H 2H	- Exposé - discussion -Questionnement - Animation Travail individuel et collectif	Vidéo Projecteur -Ecran - Tableau Blanc - Marqueurs colorés -effaceur -Pc avec codeblocks	Laboratoire	-Compiler et exécuter son programme sur la machine à l'aide de l'IDE code blocks	
Thème 2 **Les instructions de contrôle**	**Leçon 3** **Contenu** - **Les instructions de conditionnelles** (if, if-else, switch)	2H	- Exposé - discussion -Questionnement - Animation	– Vidéo Projecteur – Ecran – Tableau Blanc – Marqueurs colorés	Laboratoire	-Manipuler les structures conditionnelles	E. formative

	- **Activités d'apprentissages** Travaux pratiques **<u>Leçon 4</u>** **<u>Contenu</u>** - **Les instructions itérative**s (La boucle for, la boucle while et la boucle do while) - **Les instructions de rupture** - **Activités d'apprentissages** Travaux pratiques	2H 2H 4H	Travail individuel et collectif - Exposé - discussion -Questionnement - Animation Travail individuel et collectif	– effaceur -PC avec codeblocks Vidéo Projecteur – Ecran – Tableau Blanc – Marqueurs colorés – effaceur -PC avec codeblocks		-Manipuler les structures et itératives. - Acquérir des connaissances et de savoir-faire relatif à la résolution des problèmes complexes en adoptant le langage C.	
Thème 3 **Les Fonctions**	**<u>Leçon 5</u>** **<u>Contenu</u>** - Introduction - Déclaration	2H	- Exposé - discussion	– Vidéo Projecteur – Ecran	Laboratoire	-Mobiliser les notions de base	E. formative

	- Appel de la fonction - Passage de paramètres par valeur - **Activités d'apprentissages** Travaux pratiques	2H	-Questionnement - Animation Travail individuel ou collectif	– Tableau Blanc -Marqueurs colorés – effaceur -PC avec codeblocks		pour organiser son code ; -Se familiariser avec l'environnement de programmation C -Manipuler les fonctions	
Thème 4 **Les tableaux**	**<u>Leçon 6</u>** **<u>Contenu</u>** - Introduction - Déclaration - Utilisation - Tableau a deux dimensions - **Activités d'apprentissages** Travaux pratiques	2h 2h	- Exposé - discussion -Questionnement - Animation Travail individuel Ou collectif	– Vidéo Projecteur – Ecran – Tableau Blanc – Marqueurs colorés – effaceur - PC avec codeblocks	Laboratoire	-Manipuler les tableaux à une et a deux dimensions.	E. formative

Thème 5 **Les structures de données**	**<u>Leçon 7</u>** **<u>Contenu</u>** - Déclaration d'une structure - Manipulation - Tableau de structure - Structure de structure - **Activités d'apprentissages** Travaux pratique.	2H 2H	- Exposé - discussion -Questionnement - Animation Travail individuel Ou collectif	– Vidéo Projecteur – PC – Ecran – Tableau Blanc – Marqueurs colorés – effaceur PC avec codeblocks	Laboratoire	-Ecrire des programmes corrects partant des programmes simples à des programmes complexes. -Connaître les concepts nécessaires liées aux structures de données), -Manipuler les tableaux de structures	E. formative

Thème 6 **Les pointeurs**	**<u>Leçon 8</u>** **<u>Contenu</u>** - Stockage des variables en mémoire - Définition et déclaration d'un pointeur - Mémoire et pointeur - **Activités d'apprentissages** Travaux pratique. **<u>Leçon 9</u>** **<u>Contenu</u>** - Pointeurs et fonctions - Pointeurs et tableaux - Tableau de pointeurs	2H 2H 2H	- Exposé - discussion -Questionnement - Animation **Travail individuel ou collectif** - Exposé - discussion -Questionnement - Animation	– Vidéo Projecteur – Ecran – Tableau Blanc – Marqueurs colorés – effaceur Pc avec codeblocks Vidéo Projecteur – Ecran – Tableau Blanc – Marqueurs colorés	Laboratoire Laboratoire	-Définir et déclarer un pointeur -Gérer le stockage des variables en mémoire -Manipuler les Mémoire et pointeurs, les fonction et pointeurs -Manipuler les pointeurs et tableaux à une et a plusieurs dimensions	E. formative

	– effaceur - **Activités d'apprentissages** Travaux pratique.	2H	**Travail individuel** **ou collectif**	PC avec codeblocks			
Thème 7 : **Allocation dynamique de mémoire**	**<u>Leçon 10</u>** **<u>Contenu</u>** - Allocation dynamique pour un tableau a une dimension - Allocation dynamique pour un tableau a plusieurs dimensions - **Activités d'apprentissages** Travaux pratique.	2H 2H	- Exposé - discussion -Questionnement - Animation Travail individuel ou collectif	– Vidéo Projecteur – Ecran – Tableau Blanc – Marqueurs colorés – effaceur **PC avec codeblocks**	Laboratoire	-Allouer de l'espace mémoire pour son programme au moment de l'exécution. -Différencier les fonctions malloc(),calloc,free() et relloc. -Utiliser les tableaux dynamiques en C.	E. formative
	Examen	3H	Travail individuel	Pc avec Photoshop	Laboratoire		

Thème 1 : Introduction Général

Objectifs spécifiques

À la fin de ce thème l'apprenant, doit être capable de :

- Installez les outils nécessaires pour programmer
- Écrivez votre premier programme
- Déclarez des variables
- Faites des calculs avec des variables
- Utilisez les directives du préprocesseur
- Utiliser les fonctions d'affichage et de saisie, les opérations logiques et arithmétiques

1. Qu'est-ce que c'est le langage C ? A quoi ça sert ?

Le langage C a été développé dans les années 1970 par Dennis Ritchie chez Bell Labs. C'est un langage de programmation impératif, qui permet d'écrire des programmes efficaces et rapides. Il a été très influent dans le développement des systèmes d'exploitation et des logiciels système.

Le langage C est également connu pour sa proximité avec la machine. En effet, il permet de programmer directement en utilisant les ressources du système d'exploitation. Cela le rend très utile pour les programmes système, les pilotes de périphériques, les logiciels embarqués, etc.

Le langage C est également très populaire dans le monde de la programmation informatique. De nombreux programmes, tels que les navigateurs Web, les systèmes d'exploitation et les jeux, ont été écrits en C. C'est un langage très puissant qui est utilisé pour des applications variées.

Pour commencer à programmer en C, vous avez besoin d'un environnement de développement intégré (IDE) et d'un compilateur C. Un environnement de développement intégré (IDE) c'st un logiciel qui combine un éditeur de texte, un compilateur et un débogueur en une seule interface utilisateur. Il existe de nombreux IDE disponibles pour le langage C, dont certains sont gratuits. Voici quelques exemples populaires :

- CodeBlocks (gratuit)
- Dev-C++ (gratuit)
- Visual Studio Code (gratuit)

2. La structure d'un programme

Un programme simple se compose de plusieurs parties : Des directives de précompilation

- Une ou plusieurs fonctions dont l'une s'appelle obligatoirement main(), celle-ci constitue le programme principal et comporte 2 parties :
  - La déclaration de toutes les variables et fonctions utilisées
  - Des instructions

Les commentaires débutent par /*et finissent par */, ils peuvent s'étendre sur plusieurs lignes.

3. Les directives de précompilation

Elles commencent toutes par un #. commande signification

#include <stdio.h> permet d'utiliser les fonctions printf() et scanf()

#include <math.h> permet d'utiliser les fonctions mathématiques

#define PI =3.14159 définit la constante PI

#undef PI à partir de cet endroit, la constante PI n'est plus définie

#ifdef PI si la constante PI est définie, on compile les instructions 1,

instructions 1 ...

#else

instructions 2 ...

#endif

sinon, les instructions 2

Parmi ces directives, une seule est obligatoire pour le bon fonctionnement d'un programme : #include <stdio.h> . En effet, sans elle, on ne peut pas utiliser les fonctions utiles pour l'affichage à l'écran : printf() et la lecture des données au clavier .
scanf(). Nous verrons le fonctionnement de ces fonctions plus tard.

4. La fonction main()

Elle commence par une accolade ouvrante {et se termine par une accolade fermante}. À l'intérieur, chaque instruction se termine par un point-virgule. Toute variable doit être déclarée.

```
main(){
int i; /* declaration des variables */
  instruction_1;
 instruction_2;
.}
```

Exemple de programme simple :

```
#include <stdio.h>
/* Mon 1er programme en C */
 main(){
printf("Hello world\n");
}
```

5. La compilation

Une fois le programme écrit, on ne peut pas l'exécuter directement. Pour que l'ordinateur com- prenne ce que l'on veut lui faire faire, il faut traduire le programme en **langage machine**. Cette traduction s'appelle la **compilation**.

On compile le programme par la commande cc prog.c, où prog.c est le nom du programme. La compilation crée un fichier exécutable : a.out. On peut vouloir lui donner un nom plus explicite, et pour cela, à la compilation, on compile avec la commande cc -o prog prog.c qui va appeler le programme exécutable progau lieu de a.out.

On démarre alors le programme avec la commande ./prog.

6. Les variables et constantes

6.1. Les variables

Le C fait la différence entre les majuscules et les minuscules. Donc pour éviter les confusions, on écrit les noms des variables en minuscule et on réserve les majuscules pour les constantes symboliques définies par un #define. Les noms doivent commencer par une lettre et ne contenir aucun blanc. Le seul caractère spécial admis est le soulignement Il existe un certain nombre de noms réservés (while, if, case, ...), dont on ne doit pas se servir pour nommer les variables. De plus, on ne doit pas utiliser les noms des fonctions pour des variables.

Déclaration des variables

Pour déclarer une variable, on fait précéder son nom par son type.

Il existe 6 types de variables :

type	signification	val. min	val. max
char	caractère codé sur 1 octet (8 bits)	-2^7	$2^7 - 1$
short	entier codé sur 1 octet	-2^7	$2^7 - 1$
int	entier codé sur 4 octets	-2^{31}	$2^{31} - 1$

long	entier codé sur 8 octets	-2^{63}	$2^{63}-1$
float	réel codé sur 4 octets	$\sim -10^{38}$	$\sim 10^{38}$
double	réel codé sur 8 octets	$\sim -10^{308}$	$\sim 10^{308}$

On peut faire précéder chaque type par le préfixe unsigned, ce qui force les variables à prendre des valeurs uniquement positives. Exemples de déclarations :

Déclaration	signification
inta;	a*est entier*
intz=4;	z*est entier et vaut 4*
unsignedintx;	x*est un entier positif (non signé)*
floatzx,zy;	zx*et* zy*sont de type réel*
floatzx=15.15;	zx*est de type réel et vaut 15.15*
doublez;	z*est un réel en double précision*
charzz;	zz*est une variable caractère*
charzz='a';	zz*vaut* 'a'

Variable locale :

C'est une variable qui a une portée réduite au bloc où elle est déclarée ou définie.

Exemple :

```
main()
{
  int k, n=5:
/* bloc d' instructions */
{
int i;
for (i=0,k=0; i !=n-1; i++, k++)
printf ("%d",(i+k));
}
/*fin de  bloc d'instruction */
i++ ;
```

→ Le compilateur signale une erreur :

La variable i est locale au bloc d'instruction ou elle a été définie.

Variable globale :

Une variable globale est déclarée en dehors du sous programme.

Exemple :

```
# define  n =5;
int i, j;
/* variable globale au programme*/
main( )
{
  float k ;  ...
/* variable locale à la fonction main( )*/
 }
```

Une variable globale est visible pour tous les sous programmes qui sont déclarés après. La durée de vie d'une variable globale est celle du programme.

6.2. Les Constantes

La déclaration et l'initialisation d'une variable peuvent commencer par le qualificatif const qui indique que la valeur de cette variable ne pourra plus être modifiée.

const int **N = 30;**

Il est également possible de déclarer des constantes à l'aide de la commande du préprocesseur #define :
#define **NMAX 10**
Avant la compilation, le préprocesseur (une partie du compilateur) remplacera toutes les occurrences de NMAX par le chiffre 10.

7. Les opérateurs

Le premier opérateur à connaître est l'**affectation "="**. Exemple : {a=b+";} Il sert à mettre dans la variable de **gauche** la valeur de ce qui est à droite. Le membre de droite est d'abord évalué, et ensuite, on affecte cette valeur à la variable de gauche. Ainsi l'instruction i=i+1a un sens.

Pour les opérations dites naturelles, on utilise les opérateurs +, -, *, /, %.
%est l'opération modulo : 5%2est le reste de la division de 5 par 2. 5%2est donc égal à 1.
Le résultat d'une opération entre types différents se fait dans le type le plus haut. Les types sont classés ainsi : char < int < float < double

Par ailleurs, l'opération 'a'+1a un sens, elle a pour résultat le caractère suivant à adans le code ASCII. En C, il existe un certain nombre d'opérateurs spécifiques, qu'il faut utiliser prudemment sous peine d'erreurs.

++incrémente la variable d'une unité.

--décrémente la variable d'une unité.

Ces 2 opérateurs ne s'utilisent pas avec des réels.

Examples utilization :

i++; /* effectue i=i+1 */

i--; /* effectue i=i-1 */

Leur utilisation devient délicate quand on les utilise avec d'autres opérateurs. Exemple :

int i=1 , j;

j=i++; /* effectue d'abord j=i et ensuite i=i+1 */

/* on a alors j=1 et i=2 */

j=++i; /* effectue d'abord i=i+1 et ensuite j=i */

/* on a alors j=2 et i=2 */

Quand l'opérateur ++ est placé avant une variable, l'incrémentation est effectuée en premier. L'incrémentation est faite en dernier quand ++est placé après la variable. Le comportement est similaire pour--.

D'autres opérateurs sont définis dans le tableau qui suit :

i+=5 ; /* i=i+5 */

i-=3 ; /* i=i-3 */

i*=4 ; /* i=i*4 */

i/=2 ; /* i=i/2 */

i%=3 ; /* i=i%3 */

Pour finir, ajoutons que les opérateurs qui servent à comparer 2 variables sont :

==	égal à	!=	Différent de
<	Inférieur	<=	Inférieur ou égal
>	supérieur	>=	Supérieur ou égal
&&	'et' logique	||	'ou' logique

ATTENTION ! Ne pas confondre l'opérateur d'affectation =et l'opérateur de comparaison ==.

8. Quelques fonctions indispensables.

- **La fonction printf()**

Elle sert à afficher à l'écran la chaîne de caractère donnée en argument, c'est-à-dire entre parenthèses.

printf("Bonjour\n"); affichera Bonjour à l'écran. Certains caractères ont un comportement spécial :

\n	retour à la ligne
\b	n'imprime pas la lettre précédente
\r	n'imprime pas tout ce qui est avant
\t	tabulation horizontale
\v	tabulation verticale
\"	"
\'	'
\?	?
\!	!
\\	\

Mais printf()permet surtout d'afficher à l'écran la valeur d'une variable :

```
main(){
int n=3, m=4;
printf("%d",n); /* affiche la valeur de n au format d (decimal) */
printf("n=%d",n); /* affiche 'n=3' */
printf("n=%d, m=%d",n,m); /* affiche 'n=3, m=4' */
printf("n=%5d",n); /* affiche la valeur de n sur 5 caracteres : 'n=3'*/
}
```

Le caractère %indique le format d'écriture à lécran. Dès qu'un format est rencontré dans la chaîne de caractère entre " ", le programme affiche la valeur de l'argument correspondant.

printf("n=%d, m=%d",n,m);

ATTENTION : le compilateur n'empêche pas d'écrire un charsous le format d'un réel ⇒ af- fichage de valeurs délirantes. Et si on écrit un charavec un format décimal, on affiche la valeur du code ASCII du caractère.

%d %u	integer unsigned	entier (décimal) entier non signé (positif)
%hd	short	entier court
%d	long	entier long
%f	float	réel, notation avec le point décimal (ex. 123.15)
%e	float	réel, notation exponentielle (ex. 0.12315E+03)
%lf	double	réel en double précision, notation avec le point décimal
%le	double	réel en double précision, notation exponentielle
%c	char	caractère
%s	char	chaîne de caractères

Tableau des formats utilisables

Remarque : une chaîne de caractères est un tableau de caractères. Elle se déclare de la façon suivante charp[10];. Mais nous reviendrons sur la notion de tableau plus tard.

- **La fonction scanf()**

Dans un programme, on peut vouloir qu'une variable n'ait pas la même valeur à chaque exécution. La fonction scanf()est faite pour cela. Elle permet de lire la valeur que l'utilisateur rentre au clavier et de la stocker dans la variable donnée en argument.

Elle s'utilise ainsi :

```
main()
{
int a ;
scanf("%d",&a) ;
}
```

On retrouve les formats de lecture précisés entre ""utilisés pour printf(). Pour éviter tout risque d'erreur, on lit et on écrit une même variable avec le même format.

Le **&** est indispensable pour le bon fonctionnement de la fonction. Il indique l'adresse de la variable, mais nous reviendrons sur cette notion d'adresse quand nous aborderons les pointeurs.

- **La librairie string.h**

La librairie string.h fournit un certain nombre de fonctions très utiles pour manipuler des chaînes de caractères en C. En effet, le C ne sait faire que des affectations et des comparaisons pour 1 seul caractère.

```
char p,q;/* p et q sont des caractères */
```

```
char chaine1[10], chaine[10]; /* chaine1 et chaine2 sont des chaînes */
/* de caractères */
p = 'A';/* instruction valide */
chaine1 = "Bonjour";   /* instruction NON valide (1) */
chaine2 = "Hello";      /*instruction NON valide (2) */
if (p == q);     /*instruction valide */
if(chaine1==chaine2) /*instruction NON valide (3) */
```

Pour faire les affectations (1) et (2), et la comparaison (3), il faudrait donc procéder caractère par caractère. Ces opérations étant longues et sans intérêt, on utilise les fonctions déjà faites dans string.h. Pour les opérations d'affectation (1) et (2), il faut utiliser la fonction strcpy (string copy), et pour une comparaison (3), la fonction strcmp(string compare).

```
#include<stdio.h>
#include<string.h>
main()
{
char chaine1[10], chaine2[10]; int a;
strcpy(chaine1,"Bonjour"); /* met "Bonjour" dans chaine1 */
strcpy(chaine2,"Hello"); /* met "Hello" dans chaine2 */
a=strcmp(chaine1,chaine2);
/* a reçoit la différence chaine1 et chaine2 */
/* si chaine1 est classé alphabétiquement avant chaine2, alors a<0 */
/* si chaine1 est classé après chaine2, alors a>0 */
/* si chaine1 = chaine2, alors a = 0 */
/* Ici, "Bonjour" est alphabétiquement avant "Hello", */
/* donc chaine1 est plus petite que chaine2, et a < 0 */
}
```

9. Activités d'apprentissages

Ces Activités sur les notions de base abordent :

- La fonction main
- La bibliothèques standard, l'inclusion avec le préprocesseur
- La sortie formatée avec printf
- Les paramètres de main

Activités1 : programme minimal nop

Écrire un programme qui ne fait rien (nop signifie no opération, soit « aucune opération »).

Activité 2 : programme hello, world

Écrire un programme qui affiche hello, world.

Activité 3 : programme comptant les arguments en ligne de commande

Écrire un programme qui affiche le nombre de paramètres en ligne de commande (en anglais, les arguments

sont les données passées au programme, ou à une fonction).

Activité 4 : Écrivez un programme age.c qui demande l'âge de l'utilisateur, puis qui l'affiche.

Pour lire l'âge , vous utiliserez la fonction scanf déclarée dans stdio.h sous la forme

(void)scanf("%d", &ageLu);

Activité 5 : Écrivez un programme calcul.c qui calcule la distance entre deux points d'un plan :

- Lit les coordonnées de deux points : X1 (x1, y1) et X2 (x2, y2).
- Affiche les données lues
- Calcule et affiche le résultat à l'écran la distance d entre les deux points X1 et X2, avec la

formule : d=$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Activité 6 : Écrivez un programme taille.c qui affiche à l'écran la taille des différents types de données en octet du langage C sur votre architecture machine. Vous utiliserez l'opérateur sizeof(type).

Thème 2 : Les instructions de contrôle

Objectifs spécifiques

À la fin de ce thème l'apprenant, doit être capable de :

- Structurez votre code avec les conditions
- Utiliser les instructions de contrôles conditionnelles
- Répétez des instructions grâce aux boucles.
- Utiliser les instructions de rupture.

1. Que ce que c'est une instruction de contrôle ?

Ce sont des instructions qui permettent notamment de faire des tests et des boucles dans un programme. Leur rôle est donc essentiel. Pour chaque type d'instruction de contrôle, on trouvera à la fin de la troisième partie de ce thème les organigrammes correspondant aux exemples donnés.

2. Les instructions de controles conditionnelles

2.1. Les tests if, if-else

L'opérateur de test s'utilise comme suit :

Exemple 1 :

```
if (expression) {instruction;}
/* Si expression est vraie alors instruction est executee */
```

Exemple 2 :

```
if (expression) { instruction 1;}
 else { instruction 2;}
/* Si expression est vraie alors l'instruction 1 est executee */
/* Sinon, c'est l'instruction 2 qui est executee */
```

Exemple 3 :

```
if (expression 1) { instruction 1;
} else if (expression 2){ instruction 2;
} else if (expression 3){ instruction 3;
} else {instruction 4; }
/*Souvent, on imbrique les tests les uns dans les autres */
```

Remarque : les instructions à exécuter peuvent être des instructions simples {a=b;} ou un bloc d'instructions {a=b;c=d;...}.Comme nous l'avons déjà vu, une expression est vraie si la valeur qu'elle renvoie est non nulle.

ATTENTION ! les expressions (a=b)et (a==b)sont différentes :

if(a==b) vrai si a et b sont égaux

int b=1;

if (a=b) on met la valeur de b dans a. a vaut alors *1. l'expression est donc vraie*

2.2. Les tables de branchement : switch

Cette instruction s'utilise quand un **entier** ou un caractère prend un nombre fini de valeurs et que chaque valeur implique une instruction différente.

```
switch(i) {
case 1 : instruction 1; /* si i=1 on exécute l'instruction 1 */ break; /* et on sort du switch */
case 2 : instruction 2; /* si i=2 ... */ break;
case 10 : instruction 3; /* si i=10 ... */ break;
default : instruction 4; /* pour les autres valeurs de i */ break;
}
```

ATTENTION ! on peut ne pas mettre les break; dans les blocs d'instructions. Mais alors on ne sort pas du switch, et on exécute toutes les instructions des case suivants, jusqu'à rencontrer un break;
Si on reprend l'exemple précédent en enlevant tous les breaks, alors

? si i=1 on exécute les instructions 1, 2, 3 et 4

? si i=2 on exécute les instructions 2, 3 et 4

? si i=10 on exécute les instructions 3 et 4

? pour toutes les autres valeurs de i, on n'exécute que l'instruction 4.

3. Les Les instructions itératives

Les instructions dans le langage C se distinguent en instructions simples, conditionnelles et répétitives. Cette dernière catégorie assure la réalisation de boucle permettant l'itération d'un traitement un certain nombre de fois. On étudiera dans ce thème les boucles for, while( ) et do.. while()

3.1. La boucle for

Elle permet d'exécuter des instructions plusieurs fois sans avoir à écrire toutes les itérations. Dans ce genre de boucle, on doit savoir le nombre d'itérations avant d'6etre dans la boucle. Elle s'utilise ainsi

```
for(i=0;i<N;i++){
instructions...;
}
```

Dans cette boucle, i est le compteur. Il ne doit pas être modifié dans les instuctions, sous peine de sortie de boucle et donc d'erreur.

La $1^{e`re}$ instruction entre parenthèses est l'initialisation de la boucle.

La $2^{e`me}$ est la condition de sortie de la boucle : tant qu'elle est vraie, on continue la boucle.

La $3^{e`me}$ est l'instruction d'itération : sans elle, le compteur reste à la valeur initiale et on ne sort jamais de la boucle.

3.2. La boucle while

Contrairement à la boucke for, on n'est pas obligés ici de connaître le nombre d'itérations. Il n'y a pas de compteur.

```
while (expression) {
 instructions ...;
 }
```

L'expression est évaluée à chaque itération. Tant qu'elle est vraie, les instructions sont exécutées. Si dès le début elle est fausse, les instructions ne sont jamais exécutées.

ATTENTION ! Si rien ne vient modifier l'expression dans les instructions, on a alors fait une boucle infinie : while(1){

Instructions

} Exemple d'utilisation :

```
 main(){
 int x,R; x=1; R=10;
 while (x < R) { x = x+1;
 printf("x=%d",x);
 }}
```

3.3. La boucle do ... while

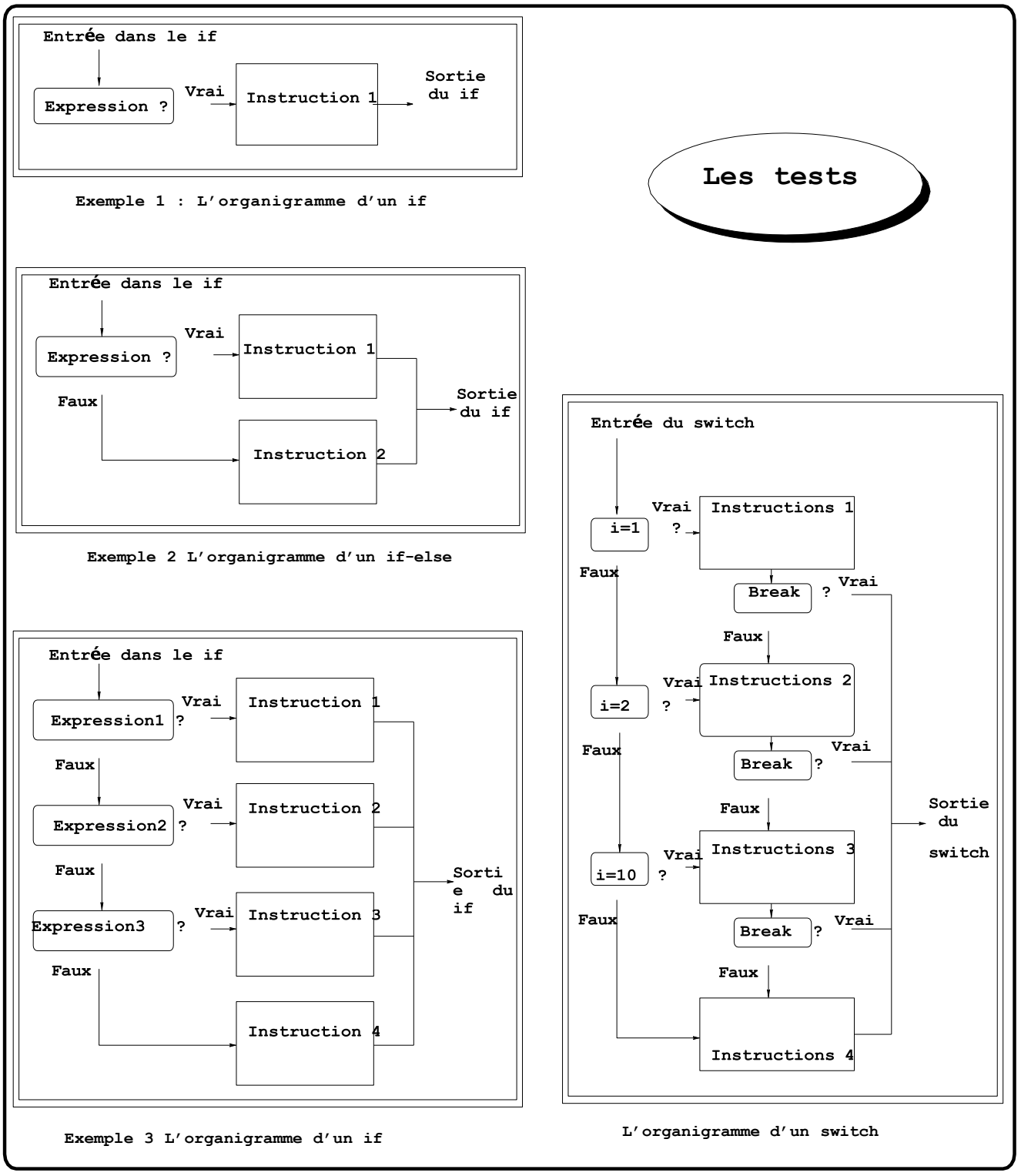
À la différence d'une boucle while, les instructions sont exécutées au moins une fois : l'expression est évaluée en fin d'itération.

```
 do {
 instructions ...;
 } while (expression)
```

Les risques des faire une boucle infinie sont les mêmes que pour une boucle while.

4. Les instructions break et continue

break fait sortir de la boucle tandis que **Continue** fait passer la boucle à l'itération suivante.



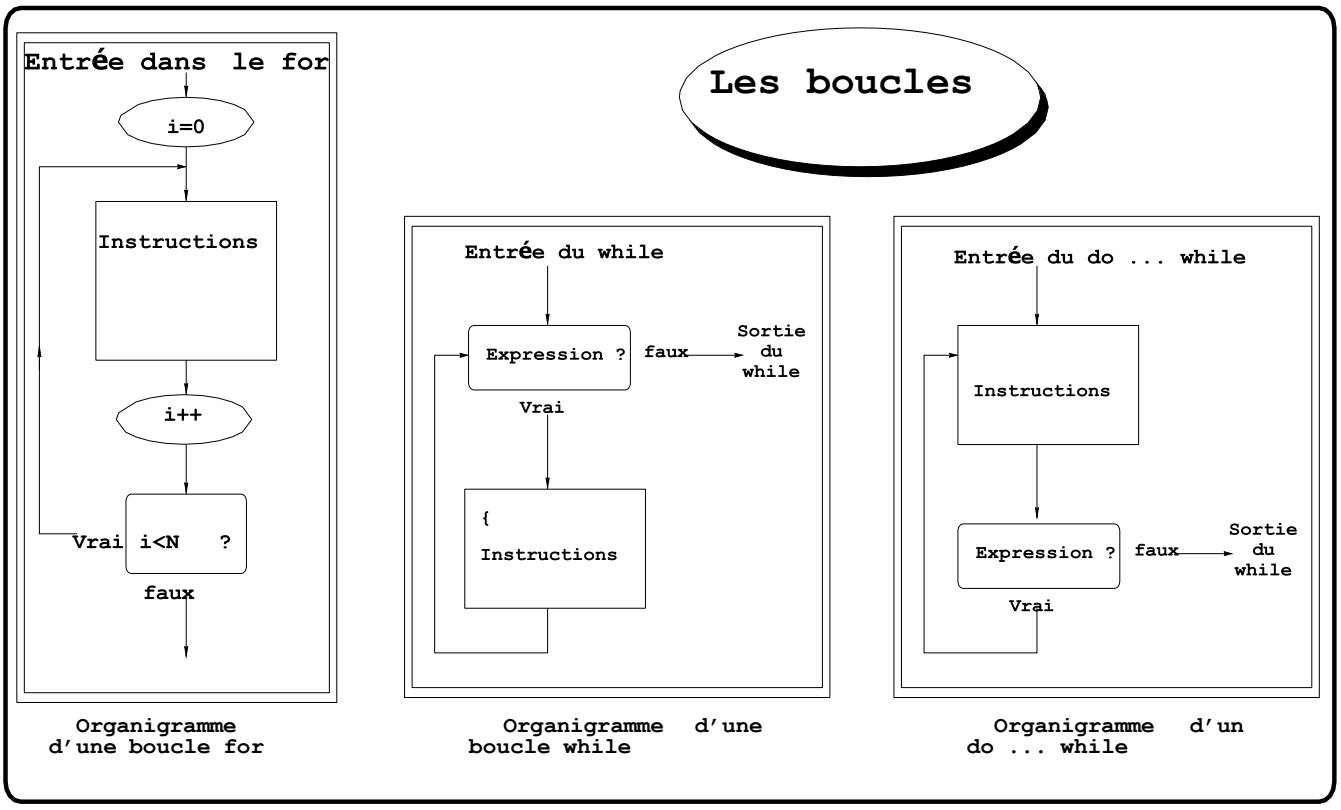


Figure – Organigramme récapitulatif des structures de contrôl

5. Activités d'apprentissages :

Pour l'ensemble des exercices, écrire l'algorithme en pseudo-code avant de coder la solution en langage C.

Activité 1 : Demander à l'utilisateur deux nombres, les mémoriser dans deux variables, multiplier leurs valeurs en affectant le résultat à une troisième variable, puis l'afficher.

Activité 2 : Même chose que l'activité 1 avec la division à la place de la multiplication. Vérifier que le 2ème nombre est différent de zéro, sinon afficher "Erreur : division par 0 ".

Activité 3 : Demander à l'utilisateur de saisir un nombre entier. Afficher si ce nombre est pair ou impair (le reste de la division entière de ce nombre par deux égaux à 0 ou non).

Pour obtenir le reste d'une division, on utilise l'opérateur "Modulo". En langage C l'opérateur Modulo est représenté par le symbole % :

int a = 14; int b = 5; int reste = a % b; // suite à cette opération, reste = 4 (14/5 = 2 et il reste 4)

Activité 4 : Demander à l'utilisateur combien font 2 fois 2 et répéter cette question aussi longtemps que la réponse est fausse. Ajouter le message "Faux, recommencez" à chaque fausse réponse, et "Bravo !" pour la bonne réponse.

Activité 5 : Amélioration de l'Activité 4 : compter le nombre d'essais et l'afficher à la fin : "Bravo ! Vous avez trouvé en x essais."

Activité 6 : Demander à l'utilisateur un nombre entier positif. Afficher tous les nombres pairs entre 0 et le nombre saisi."

Activité 7 : Demander à l'utilisateur de saisir des notes (entre 0 et 20) et lui expliquer qu'une valeur hors de cet intervalle arrêtera la saisie. Compter les notes saisies. Une fois la saisie terminée, afficher le nombre de notes saisies.

Activité 8 : Même chose que l'activité 7, mais en calculant – au fur et à mesure – la somme des notes. A la fin, calculer et afficher la moyenne, ou un message d'erreur si aucune note n'a été saisie.

Activité 9 : Prix TTC. Demander le prix unitaire HT et le nombre d'exemplaires. Calculer et afficher le prix total HT, la TVA et le prix total (TTC) à payer.

Activité 10 : Même chose que l'Activité 8, mais en mémorisant la note la plus basse et la note la plus haute. A la fin, afficher ces deux notes ainsi que la moyenne tronquée (moyenne ne tenant pas compte des valeurs extrêmes : dans notre cas la note la plus basse et la note la plus haute), ou un message d'erreur si aucune note n'a été saisie.

Activité 11 : Analyse de programme

Ces Activités d'analyses doivent être réalisés sans compilateurs, à la main à l'aide d'un crayon et d'une feuille de papier.

Soit le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
     unsigned char i=7;
  i=i/2; //"/": division entière...
  switch(i) {
    case 1 : (void)printf("Premier\n");break;
    case 2 : (void)printf("Deuxième\n");break;
    case 3 : (void)printf("Troisième\n");break;
    default : (void)printf("Non classe\n");
  }  return EXIT_SUCCESS;
}
```

Qu'est ce qui sera affiché à l'écran lors de l'exécution de ce programme ?

Même question pour le programme :

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

```
int main(void) {
  int i=18;
  i=i-(--i);
  switch(i) {
    case 1 : (void)printf("Premier\n");
    case 2 : (void)printf("Deuxième\n");
    case 3 : (void)printf("Troisième\n");
    default : (void)printf("Non classe\n");
  }  return EXIT_SUCCESS;
}
```

Thème 3 : Les fonctions

Objectifs spécifiques

À la fin de ce thème l'apprenant, doit être capable de :

- Déclarer et définir une fonction.
- Appeler une fonction dans un programme.
- Passager les paramètres par valeur.
- Manipuler retour d'une fonction. Et les types de retour.

1. Introduction

Créer une fonction est utile quand vous avez à faire le même type de calcul plusieurs fois dans le programme, mais avec des valeurs différentes. Typiquement, pour le calcul de l'intégrale d'une fonction mathématique f. Comme en mathématique, une fonction prend un ou plusieurs arguments entre parenthèses et renvoie une valeur.

2. Déclaration

On déclare une fonction ainsi :

```
type nom( type1 arg1 , type2 arg2, ... , typen argn) { */ prototype */ déclaration variables locales;
instructions; return (expression);
}
```

type est le type de la valeur renvoyée par la fonction. type1est le type du 1^{er} argument arg1

Les variables locales sont des variables qui ne sont connues qu'à l'intérieur de la fonction. expression est évaluée lors de l'instruction return (expression);, c'est la valeur que renvoie la fonction quand elle est appelée depuis main().

La $1^{e`re}$ ligne de la déclaration est appelée le **prototype** de la fonction.

Exemple de fonction :

```
float affine( float x ) { /* la fonction 'affine' prend 1 argument réel */
/* et renvoie un argument réel */
int a,b; a=3; b=5;
```

```
return (a*x+b); /* valeur renvoyee par la fonction */
}
```

On n'est pas obligés de déclarer des variables locales :

```
float norme( int x, int y ){ /* la fonction 'norme' prend 2 arguments */
/* entiers et renvoie un résultat réel */
 return (sqrt(x*x+y*y)); /* sqrt est la fonction racine carree */
}
```

On peut mettre plusieurs instructions return dans la fonction :

```
float val_absolue( float x ) { if (x < 0) {
return (-x);
} else {
return (x);
}
}
```

Une fonction peut ne pas prendre d'argument, dans ce cas-là, on met à la place de la déclaration des arguments, le mot-clé void:

```
double pi( void ) { /* pas d'arguments */
 return(3.14159);
}
```

Une fonction peut aussi ne pas renvoyer de valeur, son type sera alors void :

```
void mess_err( void ) {
printf("Vous n\'avez fait aucune erreur\n"); return; /* pas d'expression après le return */
}
```

3. Appel de la fonction

Une fonction f() peut être appelée depuis le programme principal main() ou bien depuis une autre fonction g() à la condition de rappeler le prototype de f() après la déclaration des variables de main()ou g():

```
#include <stdio.h> main(){
int x,y,r;
int plus( int x, int y );
x=5;
y=235;
```

```
r=plus(x,y); /* appel d'une fonction avec arguments */
}

int plus( int x, int y ){
void mess( void );
mess_err(); /* appel d'une fonction sans arguments */
return (x+y));
}
void mess( void ) {
printf("Vous n\'avez fait aucune erreur\n"); return;
}
```

Quand le programme rencontre l'instruction return, l'appel de la fonction est terminé. Toute instruction située après lui sera ignorée

Remarques :

- L'appel d'un sous-programme suppose qu'il est défini déjà.
- Un sous-programme est défini une et une seule fois, il est appelé autant de fois que le programme exige.
- Lors de l'appel d'un sous-programme, on distingue 2 environnements : L'environnement appelant (main) et l'environnement appelé (fonction).
- Lors de la définition d'un sous-programme, on part de paramètre formel qui figure au niveau de l'entête, lors de l'appel de fonction c'est le paramètre effectif qui est fourni.
- On doit respecter le type, l'ordre et le nombre entre les paramètres effectifs et formels.

4. Activités d'apprentissages :

Activité 1 :

1. Écrire un programme en C pour calculer la somme de deux nombres entiers en utilisant une fonction.
2. Ecrivez un programme en C pour trouver le carré d'un nombre quelconque en utilisant une fonction.

   Exemple de sortie :

   Entrez n'importe quel nombre : 2

   Le carré de 2 est: 4.00

Activité 2 : Écrire un programme en C pour échanger deux nombres à l'aide d'une fonction.

Exemple de sortie :

Entrer le 1er nombre : 3

Entrée le 2ème nombre : 5

Avant la permutation : n1 = 3, n2 = 5

Après la permutation : n1 = 5, n2 = 3

Activité 3 :

1. Écrire une fonction qui renvoie 1 si un nombre entier passé en paramètre est impair, 0 sinon. Son prototype est donc :
   int estImpair(int nb);
2. Écrire également son programme de test (main).
   Exemple de sortie :
   Entrez n'importe quel nombre : 3
   Le nombre saisi est impair.

Activité 4 :

1. Ecrivez un programme en C pour trouver la somme de la série 1!/1+2!/2+3!/3+4!/4+5!/5 en utilisant une fonction.
   Exemple de sortie :
   La somme de la série est : 34
2. Ecrivez un programme en C pour vérifier les nombres Parfait à l'aide d'une fonction.

Un nombre parfait est un nombre entier positif qui est égal à la somme de ses diviseurs positifs, à

l'exclusion du nombre lui-même

Activité 5 :

3. Ecrivez un programme en C pour afficher tous les nombres parfaits dans un intervalle donné à l'aide d'une fonction.
   Exemple de sortie:
   Entrez le min: 1
   Entrez le max: 100
   Les nombres parfaits entre 1 et 100 sont : 6 28

Thème 4 : Les tableaux

Objectifs spécifiques

À la fin de ce thème l'apprenant, doit être capable de :

- Créez des variables de type "tableaux"
- Manipulez du texte avec les chaînes de caractères
- Manipuler les tableaux à deux dimensions

1. Introduction

Dans ce thème nous allons voir tout d'abord comment déclarer un tableau, comment l'initialiser, comment faire référence à un de ses éléments. Nous terminerons par un certain nombre des activités d'apprentissages très utiles pour mettre en œuvre ces instructions.

2. Déclaration

Un tableau est un regroupement de plusieurs éléments de même type dans une seule variable. la taille d'un tableau est connue à priori lors de la déclaration Comme une variable, on déclare son type, puis son nom. On rajoute le nombre d'éléments du tableau entre crochets [] :

float tab[5]; est un tableau de 5 flottants.

ATTENTION !

Les numéros des éléments d'un tableau de n éléments vont de 0 a n − 1.

La taille du tableau doit être une constante (par opposition à variable), donc int t1[n]; où n serait une variable déjà déclarée est une mauvaise déclaration. Par contre si on a défini #define N 100 en directive, on peut déclarer int t1[N]; car N est alors une constante.

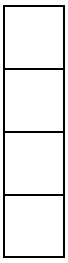
On peut initialiser un tableau lors de sa déclaration :

```
float tab[5] = { 1, 2, 3, 4, 5}; /* init. de tous les éléments de tab */
float toto[10] = {2, 4, 6}; /* equ. à toto[0]=2; toto[1]=4; toto[2]=6; */
/* les autres éléments de toto sont mis à 0. */
```

3. Utilisation

Comme shématisé ci-contre, on accède à l'élément i d'un tableau en faisant suivre le nom du tableau par le numéro i entre crochets. Un élément de tableau s'utilise comme une variable quelconque. On peut donc faire référence à un élément pour une affectation : x=tab[2], tab[3]=3, ou dans une expression : if (tab[i]< 0)

float ab[5];



Cas d'un tableau de caractères

Un tableau de caractères est en fait une **chaîne de caractères**. Son initialisation peut se faire de plusieurs façons :

```
char p1[10]='B','o','n','j','o','u','r';
char p2[10]="Bonjour"; /* init. par une chaîne litterale */
char p3[]="Bonjour"; /* p3 aura alors 8 éléments */
```

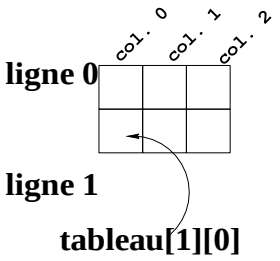
ATTENTION ! Le compilateur rajoute toujours un caractère *null* à la fin d'une chaîne de carac- tères. Il faut donc que le tableau ait au moins un élément de plus que la chaîne litterale.

4. Tableau à 2 dimensions

Un tableau à 2 dimensions est similaire à une matrice. C'est en fait un tableau de tableau à 1 dimension, il se déclare donc de la façon sui- vante :

int tableau[2][3]; /* tableau de 2 lignes et 3 colonnes */ Comme il est schématisé ci-contre, tableau[i][j] fait référence à l'élé- ment de la ligne i et de la colonne i de tableau. Tout comme un élément d'un tableau à 1 dimension, tableau[i][j] se manipule comme n'importe quelle variable.

int tableau[2][3];



5. Activités d'apprentissages :

Activité 1

1. Déclarer un tableau d'entiers de 10 éléments et l'initialiser avec les nombres 1 à 10.
2. Afficher le --tableau en séparant les valeurs par des virgules.

Activité 2

1. Déclarer un tableau d'entiers de 100 éléments et l'initialiser avec les nombres 0 à 99 (utiliser une boucle !).
2. Afficher le tableau en séparant les valeurs par des virgules (limiter à 10 valeurs par lignes).

Résultat attendu :

```
 0,  1,  2,  3,  4,  5,  6,  7,  8,  9
10, 11, 12, 13, 14, 15, 16, 17, 18, 19
20, 21, 22, 23, 24, 25, 26, 27, 28, 29
30, 31, 32, 33, 34, 35, 36, 37, 38, 39
40, 41, 42, 43, 44, 45, 46, 47, 48, 49
50, 51, 52, 53, 54, 55, 56, 57, 58, 59
60, 61, 62, 63, 64, 65, 66, 67, 68, 69
70, 71, 72, 73, 74, 75, 76, 77, 78, 79
80, 81, 82, 83, 84, 85, 86, 87, 88, 89
90, 91, 92, 93, 94, 95, 96, 97, 98, 99
```

Activité 3

Soient deux tableaux de nombres réels tSource et tDestination de 10 éléments chacun.

1. Écrire un programme permettant de recopier, dans tDestination, tous les éléments positifs de tSource, en complétant éventuellement tDestination par des zéros (initialiser tSource avec des valeurs au moment de sa déclaration).
2. Afficher les deux tableaux.

Activité 4

1. Écrire un programme qui demande 10 nombres entiers à l'utilisateur, les range dans un tableau avant d'en rechercher le plus grand et le plus petit.
2. Afficher le tableau, ainsi que le nombre le plus petit et le plus grand.

Activité 5

Demander à l'utilisateur de saisir des notes (entre 0 et 20) et lui expliquer qu'une valeur hors de cet intervalle arrêtera la saisie.

1. Saisir les notes et les mémoriser dans un tableau
2. Compter les notes saisies et afficher leur nombre
3. Calculer et afficher la moyenne
4. Comparer chaque note à la moyenne et ajouter, dans l'affichage précédent "égal", "inférieur" ou "supérieur à la moyenne"
5. Compter et afficher combien il y a de notes supérieures à la moyenne
6. Dans le tableau de notes, chercher la note la plus petite. Afficher cette note et sa position dans le tableau
7. Même chose pour la note la plus grande.

Le programme affichera un message d'erreur si le nombre de note saisi est 0.

Activité 6

Nombre de caractères dans une chaîne

1. Demander à l'utilisateur de saisir son nom
2. Compter les caractères et afficher leur nombre

Rappel : un tableau de caractère (chaine de caractères), se termine par le caractère spécial : '\0'.

Activité 7

Localisation d'une lettre

1. Initialiser un tableau avec le texte suivant : "Quel sinistre mot !"
2. Trouver et afficher à quelle position se trouve la lettre 'm'
3. Même chose pour la première lettre 's'
4. Mettre la lettre 'm' à la place de 's' et vice versa
5. Afficher le texte ainsi modifié
6. Demander à l'utilisateur de saisir une lettre quelconque
7. Chercher cette lettre dans le texte et afficher sa position ou un message d'erreur si elle est absente

Activité 8

On imagine une visite médicale en deux parties où les patients se présentent d'abord tous pour la mesure de leur taille, puis repassent, dans le même ordre, au pesage.

1. Enregistrer dans un tableau la taille t en mètres de tous les patients qui se présentent (arrêt par la saisie d'un nombre <= 0).
2. Après la saisie, afficher le nombre total de patients.
3. Pour chaque patient précédemment mesuré, enregistrer la masse m en kilogrammes dans un second tableau.
4. Calculer et afficher la taille moyenne et le poids moyen des patients. Pour chaque patient, calculer l'indice de masse corporelle : IMC = m/t^2 et afficher :

- pas assez" si IMC < 18.5,
- trop" si IMC > 25,
- ou "normal" sinon.

Activité 9 : Nombre de caractères dans une chaîne

1. Créer une fonction qui renvoie le nombre de caractères d'une chaîne de caractères passée en paramètre. Son prototype sera :
   int longueurChaine(char texte[]);
2. Dans le "main()", demander à l'utilisateur de saisir son nom
3. Toujours dans le "main()", utiliser la fonction "longueurChaine" pour compter les caractères

et afficher leur nombre

Rappel : un tableau de caractère (chaine de caractères), se termine par le caractère spécial : '\0'.

Activité 10

1. Saisir un mot et l'afficher dans l'ordre inversé
2. Créer une fonction qui inverse une chaîne de caractères passée en paramètre (cette fonction pourra elle-même utiliser la fonction "longueurChaine" créée précédemment).
3. Utiliser cette fonction dans le "main()" de manière à inverser et afficher une chaîne de caractères saisie par l'utilisateur.

Activité 11

1. Écrire une fonction qui remplace les voyelles (minuscules et majuscules) par des espaces dans une chaîne passée en paramètre.

Activité 12

1. Écrire une fonction qui cherche combien de fois un caractère est présent dans une chaîne de caractères. Le caractère à chercher et la chaîne seront passés en paramètres.

Activité 13

1. Ecrivez un programme en C pour trouver le plus grand élément d'un tableau à l'aide de la fonction.
   Exemple de sortie :

   Entrer le nombre d'éléments à stocker dans le tableau : 5
   Saisir les 5 éléments dans le tableau :
   tab[0] : 5
   tab[1] : 2
   tab[2] : 1
   tab[3] : 9
   tab[4] : 0
   Le plus grand élément du tableau est: 9

Thème 5 : Les Structures

Objectifs spécifiques

À la fin de ce thème l'apprenant, doit être capable de :

- Connaitre la syntaxe
- Déclarer les structures de données
- Accéder aux champs d'une structure

Les structures permettent de rassembler des valeurs de type différent. Par exemple, pour une adresse, on a besoin du numéro (int) et du nom de la rue (char).

1. Déclaration

On déclare une structure ainsi :

```
struct adresse {
int numero;
char rue[50];
};
```

Chaque élément déclaré à l'intérieur de la structure est appelé un **champ**. Le nom donné à la structure est appelé **étiquette de structure**. Ici, on a en fait déclaré un type de structure, pas une variable.On déclare les variables associées à une structure de cette manière :

```
struct adresse chez_pierre , chez_julie;
```

Car si la structure d'une adresse est toujours la même (numéro et nom de la rue), chez_pierreet chez_julie, qui sont des structadressedifférentes, n'habitent pas au même endroit.

On peut initialiser une structure lors de sa déclaration :

```
struct adresse chez_pierre={ 15 , "rue_Dugommier" };
```

2. Manipulation

On accède aux données contenues dans les champs d'une structure et faisant suivre le nom de la strucure par un point "." et le nom du champ voulu :

```
chez_julie.numero=19;
```

strcpy(chez_julie.rue,"avenue Pasteur");

Si 2 structures ont le même type, on peut effectuer :

chez_pierre=chez_julie; /* Pierre a emménagé chez Julie! */

Mais on ne peut pas comparer 2 structures (avec ==ou !=).

3. Tableau de structure

On déclare un tableau de structure de la même façon qu'un tableau de variables simples : le nombre d'éléments est précisé entre crochets. struct adresse pers[100];

Cette déclaration nécessite que la structure adresse ait déjà été déclarée avant. pers est alors un tableau dont chaque élément est une structure de type adresse. Et pers[i].rue fait reférence au champ "rue" de la ie`me personne de la structure pers .

4. Structure de structure

On peut utiliser une structure comme champ d'une autre structure. Dans la lignée des exemples précédents, on peut définir une structure adresse qui pourra être utilisée dans la structure repertoire. Elle peut également être utilisée dans une autre structure.

```
struct adresse {
int numero;
char rue[50];
};
struct repertoire {
char nom[20];
 char prenom[20];
struct adresse maison;
}; /* déclaration d'un champ structure */
/* de type 'adresse' appelé 'maison' */
struct repertoire monrepertoire[100];
strcpy(monrepertoire[0].nom,"Cordier");
strcpy(monrepertoire[0].prenom,"Julie");
monrepertoire[0].maison.numero = 19;
strcpy(monrepertoire[0].maison.rue,"avenue_Pasteur");
strcpy(monrepertoire[1].nom,"Durand");
strcpy(monrepertoire[1].prenom,"Pierre");
```

monrepertoire[1].maison.numero = 15;

strcpy(monrepertoire[1].maison.rue,"rue_Dugommier");

Lorsqu'un tableau fait partie des champs d'une structure, on peut accéder aux valeurs de ce tableau par : char initiale;

initiale=monrepertoire[1].prenom[0]; /* initiale de Pierre */

5. Activités d'apprentissages :

Activité 1

Créer une structure date qui contiendra les champs suivants :

- day, un entier entre 0 et 31 ;
- month, un entier entre 0 et 12 ;
- year, un entier entre 0 et 3000.

Déclarer une structure de type date et affecter à chacun des champs la date de naissance de Linus Torvalds (fondateur du noyau Linux) avant de l'afficher :

Activité 2

Créez une structure point qui comporte deux champs :

- x, l'ordonnée d'un point
- y, l'abscisse d'un point

Créez un tableau de points qui contient les coordonnées des points d'un cercle unitaire avec un pas de 20 degrés. On rappelle que les coordonnées du cercle unitaire peuvent être calculées par les formules suivantes

$\binom{x}{y}=\binom{\cos\alpha}{\sin\alpha}$

Activité 3

En C, à quoi sert une structure ?

1. à regrouper des fonctions au sein d'une entité
2. à regrouper des variables au sein d'un entité
3. à regrouper des variables de différents types
4. à décomposer un gros projet en plusieurs fichiers

En C, comment appelle-t-on les éléments qui composent une structure ?

1. les éléments
2. les champs
3. les types
4. les lignes

Comment accède-t-on aux champs d'une structure ?

1. en précisant le nom du champ entre parenthèses
2. en précisant le nom du champ entre crochets
3. en ajoutant le nom du champ séparé par un point
4. en ajoutant le nom du champ séparé par une flèche (->)

Quelles syntaxes permettent de déclarer une variable de type article ?

```
struct article {
  char designation[50];
  float prix[50];
  int quantité;
};
```

1. struct article
2. struct confiture
3. struct article confiture
4. struct confiture

Thème 6 : Les pointeurs

Objectifs spécifiques

À la fin de ce thème l'apprenant, doit être capable de :

- Utiliser, initialiser, déclarer les pointeurs en c
- Gérer le stockage des variables en mémoire
- Manipuler les pointeurs avec les fonctions et les tableaux

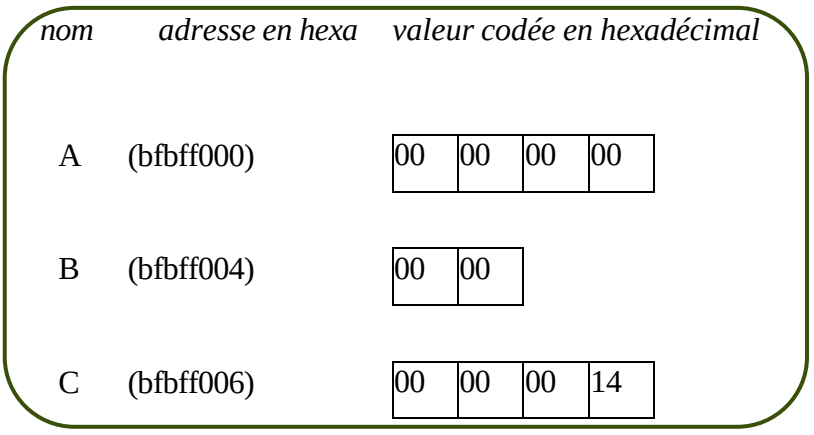
1. Stockage des variables en mémoire

Lors de la compilation d'un programme, l'ordinateur réserve dans sa mémoire une place pour chaque variable déclarée. C'est à cet endroit que la valeur de la variable est stockée. Il associe alors au nom de la variable l'adresse de stockage. Ainsi, pendant le déroulement du programme, quand il rencontre un nom de variable, il va chercher à l'adresse correspondante la valeur en mémoire.

Pour les déclarations de variables suivantes :

```
int a=0; /*'a'est  un  entier codé sur 4 octets*/
short b=0; /*'b'est un entier codé sur 2 octets*/
int c=14; /*'c' est  un entier codé sur 4 octets*/
```

ceci est stocké en mémoire :



Ici, on suppose que l'espace mémoire servant à stocker les données commence à l'adresse (bfbff000). Chaque case représente 1 octet.

Explication du shéma :

A est codé sur 4 octets et son adresse est (bfbff000), donc l'adresse de B sera (bfbff000)+(4)=(bfbff004).
B est codé sur 8 octets et son adresse est (20004), donc l'adresse de C sera (bfbff004)+(2)=(bfbff006).

2. Définition et déclaration d'un pointeur

Un pointeur est une variable qui a pour valeur l'adresse d'une autre variable : celle sur laquelle elle pointe ! Un pointeur est toujours associé à un type de variable et un seul. Au moment de la déclaration, on détermine le type de variable pointé par le pointeur, en écrivant le type concerné, puis le nom du pointeur avec une *devant :

```
int *pta; /* la variable pta est un pointeur sur un entier*/
int a; /* la variable a est un entier*/
```

Opérateur d'adresse : &

Pour affecter l'adresse de la variable aau pointeur pta, on écrit l'instruction :

```
pta=&a;
```

Cet opérateur signifie donc **adresse de**.

Opérateur d'indirection : *

Cet opérateur, mis en préfixe d'un nom de pointeur signifie **valeur de la variable pointée** ou, plus simplement, valeur pointée.

```
int a=1;
int *ptint; /* declaration d'un pointeur sur un entier */ ptint=&a; /* ptint pointe sur a */
*ptint=12; /* la variable pointée par ptint reçoit 12*/ printf("a=%d \n",a); /* affiche "a=12" */
```

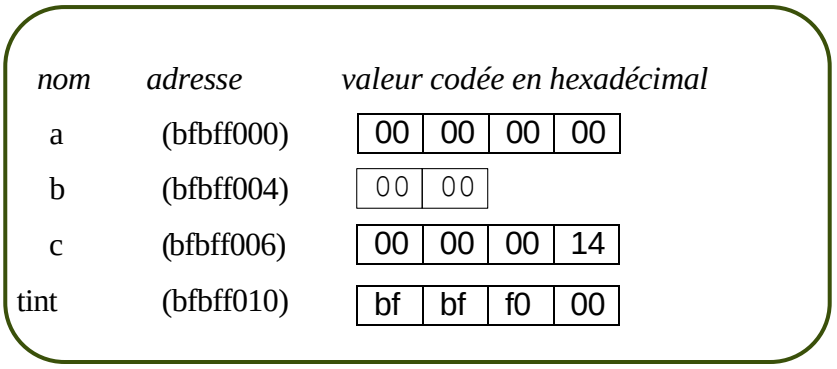
En fait, manipuler ptintrevient à manipuler a.

3. Mémoire et pointeurs

On reprend l'exemple de la 1ère partie en ajoutant un pointeur d'entier :

```
int a=0; short b=0; int
c=14;
int*ptint; ptint=&a;
```

Etat de la mémoire :



Explication : Les cases mémoire des variables a, b et c contiennent leur valeur. Celles de la variable ptint contiennent l'adresse de la valeur pointée. En effet, la valeur stockée est (bfbff000), ce qui est bien l'adresse de a.

Exemple

Voici un petit exemple d'illustration :

```
#include <stdio.h>
main(){
float *px;
float x=3.5;
px=&x;
printf ("adresse de x : 0x%lx \n",&x);
printf ("valeur de px : 0x%lx \n",px); printf ("valeur de x : %3.1f \n",x);
printf ("valeur pointee par px : %3.1f \n",*px);
return 0;
}
```

Le programme précédent affichera ceci à l'écran :

```
$ adresse de x : 0xbfbffa3c
$ valeur de px : 0xbfbffa3c
$ valeur de x : 3.5
$ valeur pointee par px : 3.5
```

4. Les pointeurs et fonctions

Une variable globale est une variable connue dans toutes les fonctions d'un programme (main comme les autres).

Une variable locale n'est connue qu'à l'intérieur d'une fonction (main ou une autre).

Les variables locales d'une fonction sont regroupées dans une partie de la mémoire, et celles d'une autre fonction, dans un autre endroit. Ainsi, il peut exister un float a; dans une fonction et un int adans une autre sans qu'il y ait de conflit.

Une conscéquence de cette propriété est que la fonction suivante ne marchera pas :

```
void   permute (int a , int b){
int buf;
buf=a;
a=b;
b   =   buf;
return;
}
```

car lors de l'appel de cette fonction depuis main, les valeurs des arguments vont être copiés dans les variables de permute et ce sont ces variables locales qui vont être modifiées, pas celles de main. Ainsi, dans main, un appel du type permute(i,j) laissera i et j inchangés. On dit que le C passe ses arguments **par valeur**.

Pour que permute fonctionne, il faut que ses arguments soient les adresses des variables a et b et utiliser des pointeurs.

```
void      permute (int *a , int *b){
int buf;
buf = *a;
*a = *b;
*b = buf;
return;
   }
```

Lors de l'appel de la fonction, les pointeurs locaux vont recevoir l'adresse des variables aet b.Donc travailler sur ces pointeurs revient à travailler sur les variables aet bde la fonction main.

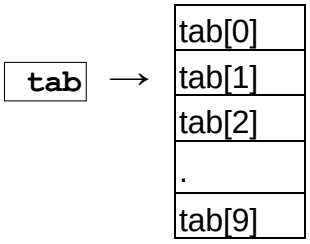
L'appel d'une telle fonction se fait ainsi : permute(&a,&b)

De façon générale, on utilise des pointeurs avec les fonctions quand on veut qu'une fonction modifie des variables du programme principal.

5. Pointeurs et tableaux à 1 dimension

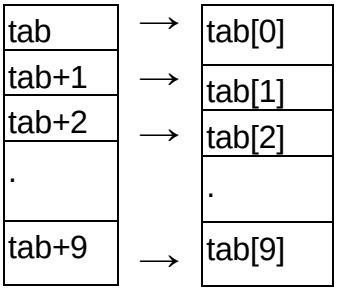
Vous avez déja manipulé des pointeurs quand vous avez manipulé les tableaux. En fait, le nom seul du

tableau est une constante qui contient l'adresse du premier élément du tableau comme le shématise la figure suivante :



Ainsi, tabest égal à &tab[0]et donc *tabest égal à tab[0].

L'élément tab[i]est équivalent à *(tab+i). On a donc les correspondances suivantes :



Au niveau de la mémoire, pour les déclarations suivantes,

```
int tab[3] = { 0 , 3 , 7 };
int *pta;
int *ptb;
pta = tab;
ptb = pta+2;
```

voici l'état de la mémoire :

nom	*adresse*	*valeur codée en hexadécimal*			
tab	(bfbff000)	00	00	00	00
	(bfbff004)	00	00	00	03
	(bfbff008)	00	00	00	07
pta	(bfbff010)	bf	bf	f0	00
ptb	(bfbff014)	bf	bf	f0	08

Explications : L'ordinateur a réservé en mémoire 3 fois 4 octets pour le tableau de 3 entiers tab. Le pointeur pta contient l'adresse du 1er élément de tab : (bfbff000). L'opération ptb=pta+2 n'ajoute pas 2 à la valeur de pta, mais ajoute 2 fois le nombre d'octets correspondant à un int,

puisque ptb est un pointeur d'entiers. Donc ptb vaut (bfbff000)+(2*4)=(bfbff008). Et ptb contient alors l'adresse de tab[2].

Ainsi, si on déclare

int tab[10]; int pt;

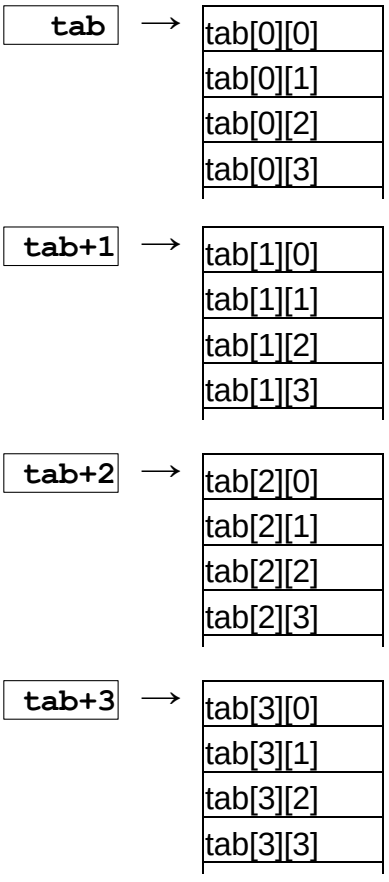
pt = tab;

on aura les équivalences suivantes :

*tab	⟷	tab[0]	⟷	*pt	⟷	pt[0]	
*(tab+1)	⟷	tab[1]	⟷	*(pt+1)	⟷	pt[1]	
et de façon plus générale, pour i de 0 à 9 :							
*(tab+i)	⟷	tab[i]	⟷	*(pt+i)	⟷	pt[i]	

6. Pointeurs et tableaux à plusieurs dimensions

Un tableau à plusieurs dimensions est un tableau dont les éléments sont eux-mêmes des tableaux. Ainsi, le tableau défini par int tab[4][5]; contient 4 tableaux de 5 entiers chacuns. tab donne l'adresse du 1er sous-tableau, tab+1celle du 2$^{e`me}$ sous-tableau et ainsi de suite :



Ici, l'opération tab+2 n'ajoute pas 2 à la valeur de tab mais ajoute 2 fois le nombre d'octets

correspondant à un tableau de 5 entiers, à savoir 5*4=20octets.

7. Tableaux de pointeurs

La déclaration d'un tableau de pointeurs se fait comme pour un tableau de variables quelconques : le type puis le nom du tableau avec

- le nombre d'éléments entre crochets derrière le nom
  et une *devant.

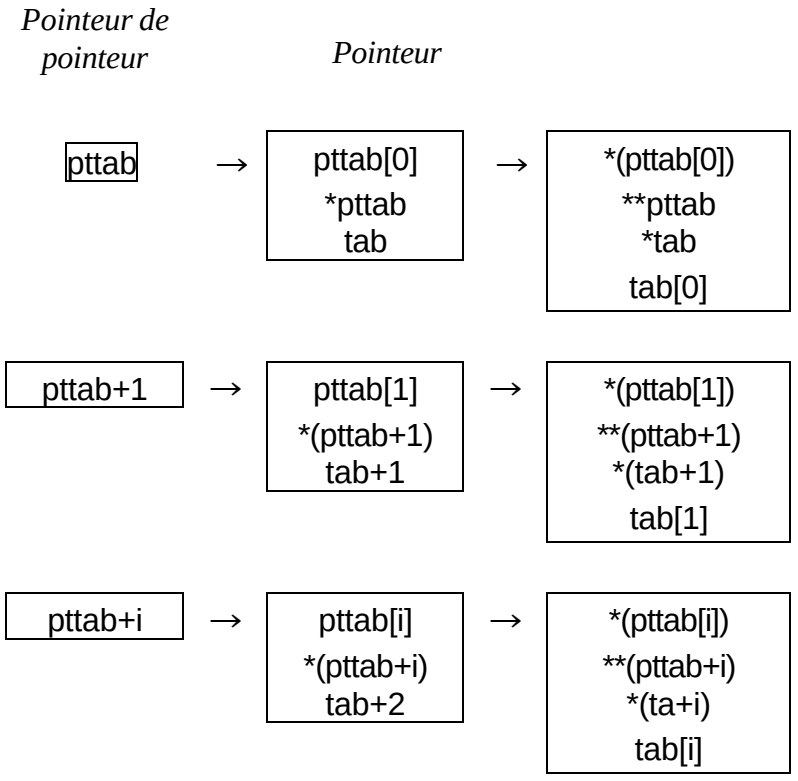
int *pttab[4]; /* pttab est un tableau de 4 pointeurs d'entiers */

1er exemple d'utilisation d'un tableau de pointeurs

int tab[4]; /* tab est un tableau de 4 entiers */

int *pttab[4] = { tab , tab+1 , tab+2 , tab+3 };

Les valeurs du tableau pttab sont des adresses de données. On dit alors que pttab est un **pointeur de pointeurs** car (pttab) pointe sur l'adresse de son 1^{er} élément, qui est lui-même une adresse. Voici les relations qu'il y a entre pttab et tab



Note : Les expressions se situant dans une même case sont équivalentes.

$2^{e`me}$ exemple :

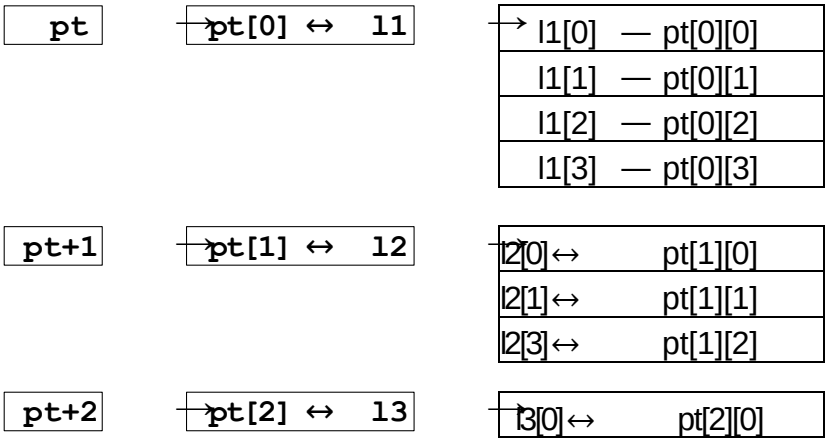
int l1[4] = { 1 , 2 , 3 , 4 };

int l2[3] = { 5 , 6 , 7 };

int l3[1] = { 8 };

int *pt[3] = { l1 , l2 , l3 };

Les éléments de tabsont les adresses de tableaux ne comportant pas le même nombre d'éléments. ptest en fait un tableau dont les 3 lignes n'ont pas la même longueur.



8. Activités d'apprentissages :

Activité 1

1. Déclarer un entier i et un pointeur p vers un entier
2. Initialiser l'entier à une valeur arbitraire et faire pointer p vers i
3. Imprimer la valeur de i
4. Modifier l'entier pointé par p (en utilisant p, pas i)
5. Imprimer la valeur de i

Activité 2

1. Déclarer et initialiser statiquement un tableau d'entiers t avec des valeurs dont certaines Seront nulles.
2. Ecrire une procédure main qui parcoure le tableau t et qui imprime les index des éléments nuls du tableau, sans utiliser aucune variable de type entier.

Activité 3

On va coder un algorithme de cryptage très simple : on choisit un décalage (par exemple 5), et un a sera remplacé par un f, un b par un g, un c par un h, etc... On ne cryptera que les lettres majuscules et minuscules sans toucher ni 0 la ponctuation ni à la mise en page (caractères blancs). On supposera que les codes des lettres se suivent de a à z et de A à Z.

1. Déclarer un tableau de caractères mess initialisé avec le message en clair.
2. Ecrire une procédure crypt de cryptage d'un caractère qui sera passé par adresse.
3. Ecrire le main qui activera crypt sur l'ensemble du message et imprimera le résultat.

Thème 7 : Allocation dynamique de mémoire

Objectifs spécifiques

À la fin de ce thème l'apprenant, doit être capable de :

- Réserver de l'espace mémoire pour son programme au moment de l'exécution.
- Utiliser les fonctions **malloc(),calloc,free()** et **relloc**.
- Utiliser les tableaux dynamiques en C.

1. Introduction

Jusqu'à maintenant, lors de la déclaration d'un tableau, il fallait préciser les dimensions, soit de façon explicite :

```
int tab[3][2];
```

soit de façon implicite :

```
int tab[][] = { {0 , 1 } , {2 , 3 } , {4 , 5 } };
```

Dans les 2 cas, on a déclaré un tableau de 3 fois 2 entiers.

Mais si l'on veut que le tableau change de taille d'une exécution à une autre, cela nous oblige à modifier le programme et à le recompiler à chaque fois, ou bien à déclarer un tableau de 1000 fois 1000 entiers et n'utiliser que les *n* premières cases mais ce serait du gâchis.

Pour éviter cela, on fait appel à l'**allocation dynamique de mémoire** : au lieu de réserver de la place lors de la compilation, on la réserve pendant l'exécution du programme, de façon interative.

La fonction malloc()

Pour l'utiliser il faut inclure la bibliothèque <stdlib.h>en début de programme.

malloc(N) renvoie l'adresse d'un bloc de mémoire de Noctets libres (ou la valeur 0 s'il n'y a pas assez de mémoire).

Exemple :

```
int *p;
p = malloc(800); /* fournit l'adresse d'un bloc de 800 octets libres */
```

/* et l'affecte au pointeur p */

L'opérateur sizeof()

D'une machine à une autre, la taille réservée pour un **int**, un **float** change. Si nous voulons réserver de la mémoire pour des données d'un type dont la grandeur varie d'une machine á l'autre, nous avons besoin de la taille effective d'une donnée de ce type.

L'opérateur sizeof nous fournit ce renseignement.

Sizeof nom_variable fournit la taille de la variable *nom_variable*

sizeof nom_constante ournit la taille de la constante *nom_constante*

sizeof(type) fournit la taille pour un objet du type *type*

Ainsi, les instructions suivantes :

```
int a[10]; char b[5][10];
printf("taille de a : %d\n",sizeof a);
printf("taille de b : %d\n",sizeof b); printf("taille de 4.25 : %d\n",sizeof 4.25);
printf("taille de Bonjour! : %d\n",sizeof "Bonjour!"); printf("taille d'un float : %d\n",sizeof(float)); printf("taille d'un double : %d\n",sizeof(double));
```

Produiront à l'exécution sur un PC :

```
taille de a : 40
taille de b : 40
taille de 4.25 : 8
taille de Bonjour! : 9
taille d'un float : 4 t
aille d'un double : 8
```

La fonction Free

La fonction free permet de libérer de la mémoire précédemment allouée.

```
void free (void* ptr);
```

Exemple, lorsque un pointeur n'est plus utilisé, il est possible de désallouer (libérer) la mémoire :

```
// Alloue dynamiquement un tableau 200 entiers
int * p;
p =(int*)malloc(200 * sizeof(int));
```

```
// Libère la mémoire utilisée par le pointeur
free (p);
```

Une erreur classique lorsqu'on utilise l'allocation dynamique est d'allouer de la mémoire, mais d'oublier de la libérer. Le programme reserve de plus en plus de bloc mémoire, mais ne les libère pas (ou que partiellement). Le programme finit par utiliser toute la mémoire disponible et au mieux, ralentit les autres applications ; au pire, fait planter la machine.

Voici un exemple de code qu'il faut absolument éviter :

```
char* p;
while (1) {
  // Alloue 10Mo
  p = (char*)malloc(1e7);
  // Pas d'appel à free(p);
}
```

La fonction calloc()

La fonction C calloc() signifie allocation contiguë. Cette fonction est utilisée pour allouer plusieurs blocs de mémoire. Il s'agit d'une fonction d'allocation dynamique de mémoire qui est utilisée pour allouer de la mémoire à complexe structures de données telles que des tableaux et des structures.

La fonction Malloc() est utilisée pour allouer un seul bloc d'espace mémoire tandis que la fonction calloc() en C est utilisée pour allouer plusieurs blocs d'espace mémoire. Chaque bloc alloué par la fonction calloc() est de la même taille.

Syntaxe de la fonction calloc() :

```
ptr = (cast_type *) calloc (n, size);
```

- L'instruction ci-dessus est utilisée pour allouer n blocs de mémoire de même taille.
- Une fois l'espace mémoire alloué, tous les octets sont initialisés à zéro.
- Le pointeur qui se trouve actuellement sur le premier octet de l'espace mémoire alloué est renvoyé.

Chaque fois qu'il y a une erreur lors de l'allocation de l'espace mémoire, telle qu'un manque de mémoire, un pointeur nul est renvoyé.

Exemple de calloc() :

Le programme ci-dessous calcule la somme d'une séquence arithmétique.

```
#include <stdio.h>
   int main() {
      int i, * ptr, sum = 0;
      ptr = calloc(10, sizeof(int));
      if (ptr == NULL) {
         printf("Erreur ! la mémoire n'est pas allouée .");
         exit(0);
      }
      printf("Calculer la somme séquentielle des 10 premiers termes \ n ");
      for (i = 0; i < 10; ++i) { * (ptr + i) = i;
         sum += * (ptr + i);
      }
      printf("Sum = %d", sum);
      free(ptr);
      return 0;
   }
```

La fonction Realloc

La fonction **realloc** permet de modifier (augmenter ou diminuer) la taille d'un bloc précédemment alloué. Voici un exemple :

```
float tab*;
// Alloue un tableau de 100 flottants
tab = (float*)malloc(100 * sizeof(float));
// Modifie la tableau pour 500 cellules
tab = (float*)realloc(tab, 500 * sizeof(float));
```

Notons que la fonction realloc() peut modifier l'adresse du pointeur si nécessaire. Dans ce cas, les données initiales seront copiées dans le nouveau bloc. Si le nouveau bloc est plus grand, l'excédent contiendra des valeurs arbitraires.

2. Allocation dynamique pour un tableau à 1 dimension

On veut réserver de la place pour un tableau de nentiers, où nest lu au clavier :

```
int n; int *tab;
printf("taille du tableau :\n");
```

```
scanf("%d", &n);
tab = (int *)malloc(n*sizeof(int));
```

Les (int *) devant le malloc s'appelle un cast. Un cast est un opérateur qui convertit ce qui suit selon le type précisé entre parenthèses.

Exemple :

```
int n1 , n2; float x = 4.5;
n1 = ((int) x)*10;
n2 = (int)(x*10);
```

Pour n1, on convertit d'abord x en entier et on multiplie le résultat par 10 : n1 = 40 Pour n2, on convertit en entier x*10: n2 = 45

La fonction malloc renvoie juste l'adresse de début d'un bloc de n fois sizeof(int). Elle ren- verra donc la même chose pour un tableau de 400 char que pour 100 int : l'adresse d'un bloc de 400 octets. C'est pourquoi le **cast** est nécessaire : pour préciser le type de données sur leqsuelles tab va pointer.

tab contient alors l'adresse de début d'un bloc de n entiers et on accède à la $i^{e`me}$ valeur du tableau par tab[i].

Jusqu'à maintenant, on a vu des pointeurs qui contenaient l'adresse d'une variable en mémoire. Ici, on a l'exemple d'un pointeur qui contient l'adresse d'un bloc contenant des données. Celles-ci ne sont accessibles que via un pointeur.

3. Allocation dynamique pour un tableau à plusieurs dimensions

On veut réserver de la place pour un tableau de n fois m entiers, où n et m sont lus au clavier : On a vu que pour manipuler des tableaux à plusieurs dimensions, il fallait utiliser des tableaux de pointeurs (*i.e.* des pointeurs de pointeurs).

```
int i,j,n,m;
float **tab; /* (1) */
 scanf("%d%d",n,m);
tab = (float **)malloc(n*sizeof(float *)); /* (2) */
for (i=0; i<n; i++){
tab[i] = (float *)malloc(m*sizeof(float)); /* (3) */
}for (i=0; i<n; i++){
for (j=0; j<m; j++){
```

```
tab[i][j] = 10*i+j; /* (4) */
}}
```

Explications :

(1) Un tableau de pointeurs étant un pointeur de pointeurs, on peut déclarer au choix un tableau de pointeurs : int *tab[3] ou un pointeur de pointeur : **tab. Dans le cas de l'allocation dynamique de mémoire, comme on ne connait pas la taille du tableau dont on aura besoin, on déclare donc un pointeur de pointeur.

(2) tab étant en fait un tableau de pointeurs de flottants, on réserve un bloc pouvant contenir n pointeurs de float. tab contient alors l'adresse de ce bloc.

(3) Les tab[i] sont des sous-tableaux de tab. On réserve alors pour chacun d'eux de la place pour m flottants. Au total, on a bien réservé de la place pour n fois m flottants.

(4) On manipule les éléments de tab comme ceux d'un tableau "normal".

4. Activités d'apprentissages :

Activité 1

À quoi sert l'allocation dynamique ?

1. à avoir un revenu minimum quand on est sans emploi
2. à écrire sur le disque dur
3. à allouer / libérer de la mémoire au fil de l'eau
4. à modifier la taille d'un bloc mémoire précédemment alloué

Que fait le code suivant ?

buffer = (int*)malloc(1000);

1. il alloue 1000 octets
2. il alloue 1000 entiers
3. il alloue 1000 pointeurs
4. ça dépend de la taille du type int

Qu'est-ce qu'une fuite mémoire ?

1. des octets de la mémoire physique devenus inutilisables

2. un bug

3. des blocs mémoires alloués mais jamais libérés

4. des appels à la fonction malloc() sans libération avec free()

Avec la fonction realloc() ...

1. la mémoire allouée pour un tableau peut être diminuée

2. la mémoire allouée pour un tableau peut être augmentée

3. le contenu de la mémoire sera initialisé à 0

4. le contenu de la mémoire contiendra uniquement des valeurs arbitraires

Que va-t-il advenir du pointeur ptr ?

5. ptr = (char*)realloc(ptr, 100);

6. l'adresse pointée par ptr restera toujours la même

7. l'adresse pointée par ptr sera nécessairement modifiée

8. l'adresse pointée par ptr pourrait être zéro

9. on ne peut pas savoir quelle sera la nouvelle adresse

Activité 2 : Ecrire un programme qui

1. Déclare un pointeur d'entier *ptr

2. Demande à l'utilisateur la taille du tableau

3. Alloue dynamiquement la mémoire pour le tableau

4. Vérifie que l'allocation s'est bien déroulée

5. Remplie le tableau de nombre aléatoires compris entre 0 et 20

6. Affiche le tableau

7. Libère la mémoire du tableau

Activité 3

Cette Activité doit être réalisé sans compilateurs, à la main à l'aide d'un crayon et d'une feuille de papier

1. Dérouler le programme ci-dessous

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

```
int main() {
    int i, * ptr, prod = 1;
    ptr = calloc(10, sizeof(int));
    if (ptr == NULL) {
        printf("Erreur !");
        exit(0);
    }
    printf(" le resultat est =\ n ");
    for (i = 0; i < 10; ++i) { * (ptr + i) = i;
        prod = prod *(* (ptr + i));
    }
    printf("Sum = %d", sum);
    free(ptr);
    return 0;
}
```

Conclusion Générale

Ce document est le noté de cours d'environ 45 heures du langage de programmation en C, destiné aux élèves de l'Ecole d'Enseignement Technique et de la Formation Professionnelle Commerciale (EETFPC) spécialité réseau informatique. Par conséquent, les points abordés sont toujours expliqués par des exemples et le minimum nécessaire pour comprendre les notions présentées est précise. Le cours est divisé en sept thèmes. Le premier décrit le langage de programmation C et présente quelques notions de base, le deuxième aborde les structures de contrôles. Le troisième traite les fonctions, le quatrième est consacré aux tableaux, le cinquième à la représentation des structures de données en C, dans le sixième aux pointeurs et dernier thème présente l'allocation dynamique de mémoire.

Bibliographie

https://fr.wikibooks.org/wiki/Exercices_en_langage_C/Notions_de_base

https://www.pandacodeur.com/pages/tutoriels/langage-c/langage-c-introduction.html

https://lucidar.me/fr/c-class/lesson-11-03-dynamic-memory-allocation/

https://www.technologuepro.com/cours-informatique/cours-25-programmation-1-langage-c/

https://www.electro-info.ovh/exercices-de-programmation-langage-c-les-tableaux