CHARLES CARNUS
LYCÉE TECHNIQUE
ENSEIGNEMENT SUPÉRIEUR

Langage C

Notions de base

Prof : **Kamal Boudjelaba**

14 août 2023

B T S S N

LTP Charles Carnus
Avenue de Saint Pierre - Bourran - 12000 RODEZ, France

IR–EC

Table des matières

1. Introduction

1.1 Bibliothèques de fonctions

La pratique du C exige l'utilisation de bibliothèques de fonctions. Ces bibliothèques sont disponibles sous forme précompilées (.lib). Afin de pouvoir les utiliser, il faut inclure des fichiers en-tête (.h) dans nos programmes. Ces fichiers contiennent les prototypes des fonctions prédéfinies dans les bibliothèques et créent un lien entre les fonctions précompilées et nos programmes.
Pour inclure les fichiers en-tête : include <fichier.h>
Pour le compilateur que nous utiliserons, différents types de fichiers seront identifiés par leurs extensions :

`.c` : fichier source
`.obj` : fichier compilé
`.exe` : fichier exécutable
`.lib` : bibliothèque de fonctions précompilées
`.h` : bibliothèque en-tête

1.2 Structure d'un programme en C

- La fonction main : Elle constitue le programme principal

```
int main()
{
    déclaration des variables
    instructions
}
```

- Les fonctions :

```
Type_du_resultat Nom_fonction (Type_param Nom_param,...)
{
    déclaration des variables locales
    instructions
}
```

- Les commentaires : Un commentaire commence toujours par les deux symboles `/*` et se termine par les deux symboles `*/`.
- Les variables : `Type_variable Nom_variable;`

1.3 Programme de base en C

Le classique "Hello word" (affiche Hello word à l'écran). Pour le faire, il faut :

— Inclure les bibliothèques

— Inclure le main

```
int main()
{
    déclaration des variables : aucune
    instruction : écrire "bonjour"
}
```

La fonction prédéfinie qui permet d'écrire à l'écran est `printf`, elle est contenue dans le fichier en-tête `stdio.h` ; sa syntaxe est : `printf("ce que l'on veut écrire");`
Voici donc le programme :

```c
#include <stdio.h>

int main()
{
  printf("bonjour\n");
  /*toute instruction se termine par un point virgule*/
}
```

Séquences d'échappement :

```
\t : tabulation (horizontale)
\v : tabulation (verticale)
\n : nouvelle ligne
\b : batch (curseur arrière)
\r : retour (retour au début de ligne, sans saut de ligne : retour chariot)
\f : saut de page
\a : attention (signal acoustique)
```

2. Les variables

En mathématiques, on distingue divers ensembles de nombres (entiers naturels, entiers relatifs, réels, complexes, ...). L'ordre de grandeur des nombres est illimité, ils peuvent être exprimés sans perte de précision. Un ordinateur utilise le système binaire pour sauvegarder et calculer les nombres, il existe pour un ordinateur deux grands systèmes de nombres : les entiers et les rationnels.

Table 1. Les entiers.

Type	Signification	Intervalle	Taille
`char`	caractère	−128 à +127	1 Octet
`short`	entier court	−32768 à +32767	2 Octets
`int`	entier standard	−32768 à +32767	2 Octets
`long`	entier long	−2147483648 à +2147483647	4 Octets
`unsigned char`	caractère	0 à +255	1 Octet
`unsigned short`	entier court	0 à +65535	2 Octets
`unsigned int`	entier standard	0 à +65535	2 Octets
`unsigned long`	entier long	0 à +4294967295	4 Octets

Table 2. Les rationnels.

Type	Mantisse[1]	Intervalle	Taille
`float`	6	3.4×10^{-38} à $3.4\times10^{+38}$	4 Octets
`double`	15	1.7×10^{-308} à $1.7\times10^{+308}$	8 Octets
`long double`	19	3.4×10^{-4932} à $1.1\times10^{+4932}$	10 Octets

[1] nombre de chiffres significatifs après la virgule

2.1 Déclaration de variables

`type`[1] `nom`[2] ;

[1] Voir tableaux 1 et 2
[2] Nom de la variable

On peut déclarer dans une instruction plusieurs variables d'un même type : `type var1, var2, var3, ...` ;
Exp : `int a, b, c` ;
On peut initialiser une variable lors de sa déclaration : `type var` = valeur ;
Exp : `float x` = 12.65 ;

2.2 Les opérateurs

Table 3. L'opérateur d'affectation.

Opérateur	Signification	Instruction	Exemples

=	Affectation	`Non_variable = expression`	`x = 3.14`
			`lettre = 'c'`
			`y = a`
			`z = a+2*b`
			`a = a-b`

Table 4. Les opérateurs arithmétiques.

Opérateur	Signification	Exemple
+	Addition	x = 4+3 ;
-	Soustraction	y = 6-2 ;
*	Multiplication	z = 4.1*5.8
/	Division	a = 5/9 ;
%[1]	Modulo[1]	b = 7%2 ;

[1] Reste de la division entière (Exp : 5%3=2)

Table 5. Les opérateurs logiques[1].

Opérateur	Signification
&&	ET
‖	OU
!	NON

[1] Les résultats des opérations logiques sont de type `int` :
La valeur 0 correspond au booléen FAUX et
la valeur 1 correspond au booléen VRAI.

Table 6. Les opérateurs de comparaison[1].

Opérateur	Signification
==	égalité
!=	inégalité (différent)
<	inférieur
<=	inférieur ou égal
>	supérieur
>=	supérieur ou égal

[1] Les résultats des opérations de comparaison sont de type `int` :
La valeur 0 correspond au booléen FAUX et
la valeur 1 correspond au booléen VRAI.

Remarque

Les opérateurs logiques considèrent toute valeur différente de 0 comme VRAI, toute valeur nulle comme FAUX.

- `32 && 2.3` → 1
- `!65.4` → 0
- `0 ||(32>5)→ 1`

Table 7. Les opérateurs d'incrémentation et de décrémentation.

$i = i+1$	$i++$ ou $++i$
$i = i-1$	$i--$ ou $--i$

(a) Pour incrémenter ou décrémenter une variable, par exemple dans une boucle (dans ce cas, pas de différence entre la notation préfixe ($++i$, $--i$) et la notation postfixe ($i++$, $i--$).

(b) Pour incrémenter ou décrémenter une variable et en même temps affecter sa valeur à une autre variable. Dans ce cas, on choisit entre la notation préfixe et postfixe :

Table 8. Affectation avec les opérateurs d'incrémentation et de décrémentation.

$x = i++$	Passe d'abord la valeur de i à x, puis incrémente i (le ++ est après i, on l'incrémente après)
$x = i--$	Passe d'abord la valeur de i à x, puis décrémente i
$x = ++i$	Incrémente d'abord i puis passe la valeur de i incrémentée à x (le ++ est avant i, on l'incrémente avant)
$x = --i$	Décrémente d'abord i puis passe la valeur de i décrémentée à x

3. Lecture et écriture des données

La bibliothèque standard `<stdio.h>` contient un ensemble de fonctions qui assurent la communication de la machine avec le monde extérieur.
Les fonctions les plus importantes sont :

Pour la lecture :

- `printf()` : écriture formatée de données
- `putchar()` : écriture d'un caractère

Pour l'écriture :

- `scanf()` : lecture formatée de données
- `getchar()` : lecture d'un caractère

3.1 Ecriture formatée de données

`printf` : cette fonction est utilisée pour transférer du texte, des valeurs de variables ou des résultats d'expression vers le fichier de sortie standard `stdout` (par défaut l'écran).

```
printf("format", expr1, expr2);
```

- format (format de représentation) : est une chaîne de caractères qui peut contenir :
  - du texte
  - des séquences d'échappement (Voir section 1.3)

— des spécificateurs de format (un spécificateur pour chaque expression)

Table 9. Les spécificateurs de format : ils comment toujours avec le symbole %.

Symbole	Impression comme	Type[1,2]
%d ou %i	Entier relatif	int
%u	Entier naturel (non signé)	int
%o	Entier exprimé en octal	int
%x	Entier exprimé en hexadécimal	int
%f	Rationnel en notation décimale	float
%e	Rationnel en notation scientifique	float
%g	Rationnel en notation décimale/scientifique	float
%lf	Rationnel en notation décimale	double
%le	Rationnel en notation scientifique	double
%lg	Rationnel en notation décimale/scientifique	double
%c	Caractère	char
%s	Chaîne de caractères	char*

[1] Entier long : on utilise %ld, %li, %lu, %lo, %lx
[2] Rationnel : long double, on utilise %Lf, %Le, %Lg

3.2 Lecture formatée de données

`scanf` : La fonction scanf reçoit ses données à partir du fichier standard stdin (le clavier).

Syntaxe : `scanf("format", adr_var1, adr_var1);`

— format (format de données)
  — La chaîne de format détermine comment les données reçues doivent être interprétées
— Les données reçues correctement sont mémorisées aux adresses indiquées par adr_var1, adr_var1 ...

Table 10. Les spécificateurs de format pour `scanf`.

Symbole	Lecture d'un(e)	Type[1,2]
%d ou %i	Entier relatif	int
%u	Entier naturel (non signé)	int
%o	Entier exprimé en octal	int
%x	Entier exprimé en hexadécimal	int
%f	Rationnel en notation décimale	float
%e	Rationnel en notation scientifique	float
%g	Rationnel en notation décimale/scientifique	float
%lf	Rationnel en notation décimale	double
%le	Rationnel en notation scientifique	double
%lg	Rationnel en notation décimale/scientifique	double
%c	Caractère	char
%s	Chaîne de caractères	char*

[1] Entier long : on utilise %ld, %li, %lu, %lo, %lx
[2] Rationnel : long double, on utilise %Lf, %Le, %Lg

Exp : `scanf("%d",&nombre);` On entre au clavier 12 $\Longrightarrow$ nombre = 12.

On peut traiter plusieurs variables avec une seule instruction `scanf` : Lors de l'entrée des données, une suite de signes d'espacement (espace, tab, interligne) est évaluée comme un seul espace.

3.3 Ecriture d'un caractère

Syntaxe : `putchar(caractere)` ;

`putchar` transfère le caractère "caractere" vers le fichier de sortie standard `stdout` (l'écran), les arguments de `putchar` sont des caractères (type `char`, i.e. des nombres entiers entre 0 et 255).

Instructions	Affichage
putchar('x') ;	x
putchar('?') ;	?
putchar(65) ;	A (Voir la table ASCII)
putchar(A) ;	Valeur de la variable A si c'est un caractère

3.4 Lecture d'un caractère

Syntaxe : `getchar()` ;

Les valeurs retournées par `getchar()` sont des caractères. Le type du résultat de `getchar` est `int`.

```
int C;
C = getchar();
```

`getchar` lit les données de la zone tampon `stdin` (clavier) et fournit les données seulement après confirmation par "Enter".

Il existe dans la bibliothèque `<conio.h>` une fonction `getch()` qui fournit immédiatement le prochain caractère entré au clavier (sans validation).

4. Structures alternatives

Les structures de contrôle définissent l'ordre dans lequel les instructions sont effectuées.
Particularité des instructions de contrôle en C : les "conditions" peuvent être des expressions qui fournissent un résultat numérique.

4.1 if-else

Structure :

```
if (condition) {
  Instruction(s) 1;
}
[ else {
  Instruction(s) 2;
} ]
```

4.2 L'opérateur condtionnel

`resultat =expr1 ? expr2:expr3;`

Si `expr1` est vraie, `resultat = expr2, sinon resultat =expr3`.

4.3 L'instruction switch

L'instruction `switch` est une instruction multidirectionnelle utilisée pour gérer les décisions. Cela fonctionne presque exactement comme la déclaration `if-else`. La différence est que l'instruction `switch` génère un code plus lisible par rapport à l'instruction `if-else`. De plus, elle s'exécute parfois plus rapidement que `if-else`.

```
switch(expression)
{
    case val1:
        instruction1;
        instruction2;
        ...
        [break;]
    case val2:
        instruction3;
        instruction4;
        ...
        [break;]
    case val3:
        instruction5;
        instruction6;
        ...
        [break;]
    default:
        instruction7;
        ...
}
```

"expression" dans l'instruction `switch` peut être toute expression valide qui donne une valeur entière. L'expression peut également être un caractère (car tous les caractères sont finalement convertis en un entier avant toute opération), mais il ne peut s'agir ni de virgule flottante (`float`, `double`) ni de chaîne.

val1, val2 ...après le mot clé "`case`" doit être de type entier (comme `int`, `long int` ...) ou de type caractère. Elle peut aussi être une expression qui donne une valeur entière. Chaque cas doit avoir une seule valeur. Les valeurs multiples dans la déclaration `case` ne sont pas autorisées. de plus, toutes les les valeurs doivent être uniques.

Exemple :

```c
#include< stdio.h>

int main(void){
    int jour;

    printf("saisir le numéro du jour : ");
    scanf("%d",&jour);

    switch(jour){
        case 1 : printf("Lundi");
        case 2 : printf("Mardi");
        case 3 : printf("Mercredi");
        case 4 : printf("Jeudi");
        case 5 : printf("Vendredi");
        case 6 : printf("Samedi");
        case 7 : printf("Dimanche");
        default: printf("jour invalide");
    }
    return 0;
}
```

5. Les boucles

Il arrive souvent dans un programme qu'une même action (instruction) soit répétée plusieurs fois, avec éventuellement quelques variantes. Il est alors fastidieux d'écrire un algorithme qui contient de nombreuses fois la même instruction. Pour gérer ces cas, on fait appel à des instructions en boucle qui ont pour effet de répéter plusieurs fois une même instruction.
Deux formes existent :

- la première, si le nombre de répétitions est connu avant l'exécution de l'instruction de répétition (On utilise la boucle for),
- la seconde s'il n'est pas connu (on utilise la boucle while).

L'exécution de la liste des instructions se nomme itération.

5.1 La boucle while

```c
while (condition)
  {
  Instructions;
  }
```

5.2 La boucle do-while

```c
do (condition)
  {
  Instructions;
  }
  while (condition);
```

5.3 La boucle for

```c
for (expr1;expr2;expr3)
  {
  Instructions;
  }
```

- `expr1` est évaluée une fois avant le passage dans la boucle, elle est utilisée pour initialiser les données de la boucle.
- `expr2` est évaluée à chaque passage de la boucle, elle est utilisée pour savoir si la boucle est répétée ou non (c'est une condition de répétition, et non d'arrêt).
- `exp3` est évaluée à la fin de chaque passage de la boucle, elle est utilisée pour réinitialiser les données de la boucle.

6. Exemples

Exemple 1

— Vérifier les résultats affichés par chacune des instructions printf.

— Mettre les résultats dans un tableau (Excel).

```c
#include <stdio.h>
main()
{
  int i = 23674;
  int j = -23674;
  long int k = (1l << 32);
  double x = 1e-8 + 1000;
  char c = 'A';
  char *chaine = "chaine de caracteres";
  printf("impression de i: \n");
  printf("%d \t %u \t %o \t %x",i,i,i,i);
  printf("\nimpression de j: \n");
  printf("%d \t %u \t %o \t %x",j,j,j,j);
  printf("\nimpression de k: \n");
  printf("%d \t  %o \t %x",k,k,k);
  printf("\n%ld \t %lu \t %lo \t %lx",k,k,k,k);
  printf("\nimpression de x: \n");
  printf("%f \t %e \t %g",x,x,x);
  printf("\n%.2f \t %.2e",x,x);
  printf("\n%.20f \t %.20e",x,x);
  printf("\nimpression de c: \n");
  printf("%c \t %d",c,c);
  printf("\nimpression de chaine: \n");
  printf("%s \t %.10s",chaine,chaine);
    printf("\n");
}
```

Exemple 2

— Tester le programme avec 3 valeurs.

— Mettre les résultats des tests dans un tableau (Excel).

```c
#include <stdio.h>
main()
{
  int i;
  printf("entrez un entier sous forme hexadecimale i = ");
  scanf("%x",&i);
  printf("i = %d\n",i);
}
```

Exemple 3

— Tester le programme avec des valeurs différentes.

— Mettre les résultats des tests dans un tableau (Excel).

```c
#include <stdio.h>
int main()
{
    int A,B ;
    printf("Entrer deux nombres entiers:\n") ; scanf("%d %d",&A,&B);
    if(A>B)
    printf("%d est plus grand que %d\n",A,B); else
    if(A<B)
    printf("%d est plus petit que %d\n",A,B); else
    printf("%d est égal à %d\n",A,B);
}
```

Exemple 4

— Tester le programme.

— Comparer les résultas avec ceux des exemples 5, 6 et 7.

```c
#include <stdio.h>
int main()
{
int i =0;
while(i<4)
    {
        printf("%d\n",i);
        i++ ;
    }
}
```

Exemple 5

— Tester le programme.

— Comparer les résultas avec ceux des exemples 4, 6 et 7.

```c
#include <stdio.h>
int main()
{
int i =0;
while(i<4)
    printf("%d\n",i++);
}
```

Exemple 6

— Tester le programme.

— Comparer les résultas avec ceux des exemples 4, 5 et 7.

```c
#include <stdio.h>
int main()
{
int i =0;
while(i<4)
    printf("%d\n",++i);
}
```

Exemple 7

— Tester le programme.

— Comparer les résultas avec ceux des exemples 4, 5 et 6.

```c
#include <stdio.h>
int main()
{
int i =4;
while(i)
    printf("%d\n",i--);
}
```

Exemple 8

— Tester le programme 2 fois.

— Prendre une capture d'écran du résultat de chaque essai.

```c
#include <stdio.h>
int main()
{
int n ;
do
    {
        printf("Entrer un nombre entre 1 et 10:\n") ; scanf("%d",&n);
    }
    while(n<1||n>10) ;
}
```

Exemple 9

— Tester le programme 2 fois.

— Prendre une capture d'écran du résultat de chaque essai.

```c
#include <stdio.h>
int main()
{
int n, div ;
printf("Entrer le nombre à diviser:\n");
scanf("%d",&n);
do
    {
        printf("Entrer le diviseur:\n");
        scanf("%d",&div);
    }
    while(!div) ;
    printf("%d/%d = %f\n",n,div,(float)n/div);
}
```

Exemple 10

— Tester le programme.
— Modifier le programme pour que le logiciel affiche jusqu'à la valeur du carré du nombre 100000.
— Conclure.

```c
#include <stdio.h>
int main()
{
int i;
for(i=1;i<=4;i++)
    {
        printf("Le carré de %d est %d\n",i,i*i);
    }
}
```

Exemple 11

— Tester le programme avec différents caractères.
— Conclure.

```c
#include <stdio.h>

int main()
{
int C ;
printf("introduire un caractère suivi de 'Enter'\n");
C = getchar();
printf("Le caractère %c a le code ASCII %d\n", C, C);
return 0;
}
```

7. Les tableaux

7.1 Les tableaux à une dimension

Déclaration : `Type(simple) NomDuTableau[Dimension];`
Exemples :

- `int Tableau1[6];` Déclaration d'un tableau nommé `Tableau1`, de type `int` et de dimension 6
- `char MonTableau[50];` Déclaration d'un tableau nommé `MonTableau`, de type `char` et de dimmension 50
- `float Table[12];` Déclaration d'un tableau nommé `Table`, de type `float` et de dimension 12

Lors de la déclaration d'un tableau, on peut initialiser ses composantes en indiquant la liste des valeurs entre accolades :

```
int B[5] = {10, 20, 30, 40, 50};
```

Rang (indice)	0	1	2	3	4
Valeur	10	20	30	40	50

— Ce tableau est de longueur 5 car il contient 5 emplacements (cases).
— Chacun des éléments du tableau est repéré par son rang (appelé indice).

Pour accéder à un élément du tableau, il suffit de préciser entre crochets l'indice de la case contenant cet élément :

```
B[0], B[1], ..., B[4]
```

- Pour accéder au troisième élément (30), on écrit `B[2];`
- `x = B[1];` La variable `x` prend la valeur du deuxième élément du tableau (20).
- `B[4] = 79;` on remplace la valeur du cinquième élément (50) par 79.

La boucle `for` se prête particulièrement bien pour accéder et afficher les éléments du tableau.

```
int main()
{
  int B[5];
  for (int i=0;i<5;i++)
    {printf("%d \n",B[i])};
  return 0;
}
```

Affectation avec des valeurs saisies au clavier :

```
int main()
{
  int B[5];
  for (int i=0;i<5;i++)
    {scanf("%d",&B[i])};
  return 0;
}
```

7.2 Les tableaux à deux dimensions

Déclaration :
`Type(simple) NomDuTableau[DimensionLigne][DimensionColonne];`
Ou
`Type Nom[Nombre_de_lignes][Nombre_de_colonnes]{{Val1, Val2, ...},{...},{...},...};`
Exemples :

- `double A[2] [5];` Déclaration d'un tableau à deux dimensions nommé A et de type `double`
- `char B[4] [2];`Déclaration d'un tableau à deux dimensions nommé B et de type `char`
- `double table[3][2]{{5.1, 3.6}, {3.14, 0.25}, {10.3, 7.5}};`

Indices	0	1
0	5.1	3.6
1	3.14	0.25
2	10.3	7.5

— Ce tableau contient 3 lignes et 2 colonnes.
— Les éléments du tableau sont repérés par leur numéro de ligne et leur numéro de colonne :
  ◦ `Y = table[1][0];` La variable y prend la valeur située à la deuxième ligne et à la première colonne, c-à-d, 3.14

Les composantes d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire.

```
short A[2][3] ={{1,3,5},
                {10,30,50}};
```

Pour accéder à un élément du tableau :

```
NomTableau[Ligne][colonne];
```

Exemple :

```
int main()
{
  int A[5][10];
  int i, j;
  for (i=0;i<5;i++)
  {
    for (j=0;j<10;j++)
      {printf("%7d \n",A[i][j]);}
  }
return 0;
}
```

8. Les chaînes de caractères

Il n'existe pas de type spécial chaîne ou string en C. Une chaîne de caractères est traitée comme un tableau à une dimension de caractères.
La déclaration d'une chaîne de caractères se fait avec l'une des méthodes suivantes :

- `char NomChaine[Longueur];`
- `char *NomChaine;`

Exemples :

```
char Nom[26];           //Déclaration sans initialisation
char Prenom[]="Abcd";   //Déclaration avec initialisation
```

Les fonctions de `stdio.h`

```
char Texte[6] = "Hello";
char CH[5];
printf("%s",Texte);    // Affiche Hello sans retour à la ligne
printf("%s",CH);       // CH contient l'adresse du premier caractère de la chaîne
puts(Texte);           // Affiche Hello avec un retour à la ligne
puts("Bonjour");       // Affiche Bonjour avec un retour à la ligne
gets(CH);              // Lit une ligne de caractères de stdin
```

Les fonctions de `string.h`

```
char CH1[] = "Hello";
char CH2[] = "Bonjour";
char *chaine = "Bonsoir";
int k = 2;
char c = 'a';
strlen(CH1);         // Fournit la longueur de CH1
```

```
strcpy(CH1,CH2);     // Copie CH1 dans CH2
strcat(CH1,CH2);     // Ajoute CH2 à la fin de CH1
strcmp(CH1,CH2);     // Compare CH1 et CH2
strcpy(CH1,"Bonsoir");
```

Les fonctions de stdlib.h

```
char CH[] = "125";
atoi(CH);   // Retourne la valeur numérique représentée par CH comme int
atol(CH);   // Retourne la valeur numérique représentée par CH comme long
atof(CH);   // Retourne la valeur numérique représentée par CH comme double
```

Les fonctions de ctype.h

```
// c représente une valeur de type int qui peut être représentée comme caractère.
isupper(c);   // Retourne une valeur différente de 0 si c'est une majuscule
islower(c);   // Retourne une valeur différente de 0 si c'est une minuscule
isspace(c);   // Retourne une valeur différente de 0 si c'est un signe d'espacement
tolower(c);   // Retourne c converti en minuscule si c'est une majuscule
toupper(c);   // Retourne c converti en majuscule si c'est une minuscule
```

8.1 Tableaux de chaîne de caractères

```
// Déclaration
char JOUR[7][9];
char *mois[12];

// Initialisation
char JOUR[7][9] = {"Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi","Dimanche"};
char *mois[12] = {"Jan","Fev","Mars","Avr","Mai","Juin","Juil","Aout","Sep","Oct","Nov","Dec"};

// Accès aux différents éléments
printf("%s",JOUR[2]); // Affiche Mercredi
printf("%s",mois[0]); // Affiche Jan

// Accès aux caractères
for (int i=0;i<7;i++)
  printf("%c",JOUR[1][0]);

// Affectation :
// Pas d'affectation directe (JOUR[2]="Jeudi";). On utilise :
strcpy(JOUR[2],"Jeudi");
```

9. Les fonctions

9.1 Définition de fonctions

```
TypeRésultatRetourné NomFonction(TypeParam1 NomParam1, TypeParam2 NomParam2, ...)
{
  Déclarations locales
  Instructions
}
```

— **TypeRésultatRetourné** : indique le type de variable renvoyée par la fonction (string, int, double ...)

— **NomFonction** : permet de donner un nom à la fonction

Exemple :

```
int somme(int a, int b)
{
  int r;
  r = a+b;
  return (r);
}
```

9.2 Déclaration de fonctions

En C, il faut déclarer chaque fonction avant de pouvoir l'utiliser. La déclaration informe le compilateur du type des paramètres et du résultat de la fonction.
La déclaration d'une fonction se fait par un prototype de la fonction qui indique uniquement le type des données transmises et reçues par la fonction.

```
TypeRésultat NomFonction(TypeParam1, TypeParam2, ...);
ou
TypeRésultat NomFonction(TypeParam1 NomParam1, TypeParam2 NomParam2, ...);
```

9.3 Les fonctions récursives

Une fonction récursive est une fonction qui pour fournir un résultat, s'appelle elle-même un certain nombre de fois.

Exemple 9.1

La formule de calcul de la factorielle d'un nombre entier est donnée par :
$n! = 1 * 2 * 3 * \cdots * (n-1) * n$
Ce calcul peut être réalisé avec une boucle *for*.

Une autre manière de réaliser ce calcul, serait de dire que :
$n! = n * (n-1)!$
c.à.d. la factorielle d'un nombre, c'est ce nombre multiplié par la factorielle du nombre précédent. Pour coder cela, il faut une fonction `Facto`, chargée de calculer la factorielle. Cette fonction effectue la multiplication du nombre passé en argument par la factorielle du nombre précédent.
Pour terminer, il manque la condition d'arrêt de ces auto-appels de la fonction `Facto`. On s'arrête quand on arrive au nombre 1, pour lequel la factorielle est par définition 1.

```
Fonction Facto (n : Entier)
Si n = 1 alors
Renvoyer 1
Sinon
Renvoyer Facto(n - 1) * n
```

Le processus récursif remplace en quelque sorte la boucle, c'est-à-dire un processus itératif.

10. Exercices

Exercice 1

Ecrire un programme qui lit un caractère au clavier et affiche le caractère ainsi que son code numérique en employant `getchar` et `printf`.

Exercice 2

Ecrire un programme qui calcule et affiche la distance DIST (type double) entre deux points A et B du plan dont les coordonnées (XA, YA) et (XB, YB) sont entrées au clavier comme entiers.

Exercice 3

Ecrire un programme qui calcule la somme des N premiers termes de la série harmonique : $1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{N}$.

Exercice 4

Ecrire un programme qui calcule les solutions réelles d'une équation du second degré $ax^2+bx+c=0$ en discutant la formule :

- Utiliser une variable d'aide D pour la valeur du discriminant b^2-4ac et décider à l'aide de D, si l'équation a une, deux ou aucune solution réelle. Utiliser des variables du type int pour a, b et c.
- Considérer aussi les cas où l'utilisateur entre des valeurs nulles pour :
  - a ;
  - a et b ;
  - a, b et c ;
- Afficher les résultats et les messages nécessaires sur l'écran.
- Modifier le programme afin de considérer le cas des solutions complexes.

Exercice 5

Ecrire un programme C qui lit trois entiers et affiche leur somme, leur produit et leur moyenne.

Exercice 6

Une date est donnée sous forme d'un nombre entier de 6 chiffres. Par exemple 060922 représente le 06 Septembre 2022. Ecrire un programme C qui accepte en donnée un tel nombre et affiche le résultat suivant :
Jour : 06
Mois : 09
Année : 2022

Exercice 7

Ecrire un programme C qui lit trois entiers A, B, et C et affiche le maximum et le minimum.

Exercice 8

Soit la suite U_n définie par : $U_0=1$ et $U_{n+1}=5U_n+3$.
Ecrire un programme qui permet de lire n et de calculer la suite U_n pour un rang n.

Exercice 9

Ecrire un programme C qui permet d'afficher si une valeur X existe dans un tableau Tab de N réels ou non (max.50).

Exercice 10

Ecrire un programme C qui multiplie une matrice carré d'entiers $M(N*N)$ par un réel X donné.

Exercice 11

Ecrire un programme C qui calcule la transposé d'une matrice.

Exercice 12

— Ecrire une fonction qui permet de permuter deux entiers.

— Ecrire une fonction qui permet de remplir un tableau.

— Ecrire une fonction qui permet d'afficher un tableau.

Exercice 13

En utilisant `switch`, écrire un programme qui demande à l'utilisateur d'entrer deux nombres réels, de choisir l'opération à réaliser ('+' pour addition, '-' pour la soustraction, ...) et d'afficher le résultat de l'opération.

Exercice 14

Tester les codes suivants et conclure (ne pas oublier de les compléter !).

Code 1 :

```
x=1;
while (x<=10) {
   if (x == 7) {
   printf("Division par zéro !");
continue;
   }
   a = 1/(x-7);
printf("%f", a);
x++;
}
```

Code 2 :

```
x=1;
while (x<=10) {
   if (x == 7) {
   printf("Division par 0");
x++;
continue;
}
   a = 1/(x-7);
printf("%f", a);
x++;
}
```

Code 3 :

```
for (x=1; x<=10; x++) {
 a = x-7;
if (a == 0) {
    printf("Division par 0");
break;
 }
 printf("%f", 1/a);
}
```