

1.1 – Qu'est-ce que le C ?

- ▶ **Langage de bas niveau** (proche du matériel) mais **portable**.
- ▶ **Utilisations** : Systèmes embarqués, noyaux de systèmes d'exploitation (Linux, Windows), drivers, applications performantes.
- ▶ **Caractéristiques** :
 - ▶ **Rapidité** : Compilé en code machine.
 - ▶ **Contrôle fin** : Gestion manuelle de la mémoire, pointeurs.
 - ▶ **Bibliothèque standard riche** (stdio.h, stdlib.h, etc.).

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

3

1.2 – Premier Programme en C

```

1  #include <stdio.h>
2  int main(){
3      printf("Bonjour super beau prof");
4      return 0;
5  }
```

- ▶ **Explication** :
 - ▶ `#include <stdio.h>` : Inclut la bibliothèque standard pour l'entrée/sortie.
 - ▶ `int main()` : Point d'entrée du programme.
 - ▶ `printf()` : Affiche du texte.
 - ▶ `return 0` : Indique une exécution réussie.

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

4

2 – Bases du Langage C

- ▶ Les variables doivent faire l'objet d'une déclaration de **type** de la forme:
`type liste_des_variables ;`
- ▶ Le type d'une donnée détermine :
 - ▶ l'ensemble des valeurs admissibles,
 - ▶ le nombre d'octets à réserver en mémoire
 - ▶ l'ensemble des opérateurs qui peuvent y être appliqués

2.1 – En C : types simples

- ▶ Le langage C est faiblement typé (*i.e. flexible*).
- ▶ \Rightarrow permet d'utiliser des opérandes de différents types dans un même calcul
- ▶ Le risque :
 - ▶ Conversion de type automatique
 - ▶ Arrondis
 - ▶ Résultats incorrects, inexplicables

2.1 – En C : types simples

- ▶ Ensembles de nombres et leur représentation
 - ▶ En mathématiques : \mathbb{N} , \mathbb{Z} , \mathbb{R}
- ▶ En informatique :
 - ▶ Ordinateur ne peut traiter aisément que des nombres entiers d'une taille limitée
 - ▶ Utilisation du système binaire pour calculer et sauvegarder ces nombres
 - ▶ Valeurs correctement approchées des entiers très grands, des réels ou des rationnels à partie décimale infinie sont obtenues par des ruses de calcul et de représentation

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

7

2.1 – En C : types simples

- ▶ Obligation du programmeur :
 - ▶ Il n'est pas nécessaire qu'il connaisse les détails des méthodes de codage et de calcul
 - ▶ Il doit cependant pouvoir :
 - ▶ choisir un type numérique approprié à un problème donné
 - ▶ choisir un type approprié pour la représentation sur l'écran
 - ▶ prévoir le type résultant d'une opération entre différents types numériques;
 - ▶ prévoir et optimiser la précision des résultats intermédiaires au cours d'un calcul complexe;

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

8

2.1 – En C : types simples

► Exemple :

- Supposons que le type choisi ne propose que 5 positions décimales

$$(1.00001 \cdot 10^7 + 33) - 1 \cdot 10^7 = 100$$

$$(1.00001 \cdot 10^7 - 1 \cdot 10^7) + 33 = 133$$

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

9

2.2 – En C : types entiers

► Caractéristiques des types numériques entiers

Définition	Description	Val min	Val max	Nb. Octets
char	Caractère	-128	127	1
short	Entier court	-32768	32767	2
int	Entier standard	-2147483648	2147483647	4
long	Entier long	-2147483648	2147483647	4

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

10

2.3 – En C : types rationnels

- ▶ Souvent appelés des flottants (*virgule flottante*)
 $\text{<+|-> <mantisse> * 10^{<exposant>}}$
- ▶ **<+|->** : signe positif ou négatif du nombre
- ▶ **<mantisse>** : décimal positif avec un seul chiffre devant la virgule
- ▶ **<exposant>** : entier relatif

2.3 – En C : types rationnels

- ▶ Types C :
 - ▶ **float** sur 4 octets
 - ▶ **double** sur 8 octets
 - ▶ **long double** sur 12 octets

Exemples :
123.449997
1.234500e+02

Type	Taille (octets)	Plage de valeurs	Exemple
float	4	1.2E-38 à 3.4E+38	float pi = 3.14;
double	8	2.3E-308 à 1.7E+308	double x = 1.23;

2.2 – En C : types entiers

► Remarque :

- Le type `int` est le type de base pour les calculs avec les entiers
- Le codage du type `int` est dépend de la machine :
 - `short` = 2 octets ; **`int` = 4 octets** ; `long` = 4 octets
 - `short` = 2 octets ; **`int` = 2 octets** ; `long` = 4 octets

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

13

2.2 – En C : types entiers

► Les modificateurs `signed/unsigned`

Définition	Description	Val min	Val max	Nb. Octets
<code>unsigned char</code>	Caractère	0	255	1
<code>unsigned short</code>	Entier court	0	65535	2
<code>unsigned int</code>	Entier standard	0	4294967295	4
<code>unsigned long</code>	Entier long	0	4294967295	4

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

14

2.4 – En C : Opérateurs

- ▶ Arithmétiques : +, -, *, /, % (modulo)
- ▶ Logiques : && (ET), || (OU), ! (NON)
- ▶ Comparaison : ==, !=, <, >, <=, >=

2.5 – En C : lecture - écriture

- ▶ Ecriture `printf` : syntaxe


```
int printf(const char *format, ...);
```

 - ▶ format : Chaîne de formatage contenant des spécificateurs (%d, %s, etc.).
 - ▶ ... : Liste d'arguments correspondants aux spécificateurs.
 - ▶ Retourne : Le nombre de caractères affichés.

2.5 – En C : lecture - écriture

► Ecriture `printf` : Spécificateurs de Format Courants

Spécificateur	Type	Exemple
<code>%.2f</code>	Flottant (2 décimales)	<code>printf("%.2f", 3.14159);</code> → 3.14
<code>%%</code>	Affiche %	<code>printf("%%");</code> → %
<code>%c</code>	Caractère (char)	<code>printf("%c", 'A');</code> → A
<code>%d</code>	Entier (int)	<code>printf("%d", 42);</code> → 42
<code>%f</code>	Flottant (float)	<code>printf("%f", 3.14);</code> → 3.140000
<code>%p</code>	Pointeur	<code>printf("%p", &variable);</code> → 0x7ffd42a1b2ac
<code>%s</code>	Chaîne (char*)	<code>printf("%s", "Bonjour");</code> → Bonjour

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

17

2.5 – En C : lecture - écriture

► Ecriture `printf` : Flags de Formatage

Flag	Signification	Exemple
-	Alignement à gauche	<code>%-5d</code> → 42 (si 42)
0	Remplit avec des zéros	<code>%05d</code> → 00042
+	Affiche le signe (+ ou -)	<code>%+d</code> → +42 OU -42

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

18

Spécificateurs de Format Courants

2.5 – En C : lecture - écriture

19

► Ecriture `printf` : exemples

```
3 int nombre = 42;
4 printf("Nombre: |%5d| |%-5d| |%05d|\n", nombre, nombre, nombre);
```

Nombre: | 42| |42 | |00042|

► Précision : Nombre de décimales pour les flottants ou nombre maximal de caractères pour les chaînes.

```
printf("%.2f\n", 3.14159);
printf("%.5s\n", "Bonjour");
```

3.14
Bonjo

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

19

Spécificateurs de Format Courants

2.5 – En C : lecture - écriture

20

► Ecriture `printf` : exemple complet

```
#include <stdio.h>

int main() {
    int age = 25;
    float taille = 1.754568;
    char initiale = 'A';

    printf("Âge: %d ans, Taille: %.2f m, Initiale: %c\n", age, taille, initiale);

    return 0;
}
```

Âge: 25 ans, Taille: 1.75 m, Initiale: A

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

20

2.5 – En C : lecture - écriture

- **Ecriture putchar : syntaxe** (permet d'afficher un caractère)

```
int putchar(int c);
```

- **c** : Caractère à afficher (passé comme int pour compatibilité avec EOF).
- **Retourne** : Le caractère affiché, ou EOF en cas d'erreur.

- **Exemple** : `putchar('A'); // Affiche 'A'`

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

21

2.5 – En C : lecture - écriture

- **printf VERSUS putchar**

putchar	printf
Affiche un seul caractère .	Affiche des chaînes formatées .
Plus rapide pour les caractères individuels.	Plus flexible (nombres, flottants, etc.).
Retourne le caractère ou EOF.	Retourne le nombre de caractères affichés.

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

22

2.5 – En C : lecture - écriture

► scanf : Lecture Formatée, Syntaxe

```
int scanf(const char *format, ...);
```

- format : Chaîne de formatage avec des spécificateurs (%d, %f, etc.).
- ... : Adresses des variables où stocker les valeurs lues (utilisez &).
- Retourne : Le nombre de variables correctement lues.

2.5 – En C : lecture - écriture

► scanf :

Spécificateur	Type	Exemple
%d	Entier (int)	scanf("%d", &age);
%f	Flottant (float)	scanf("%f", &taille);
%lf	Double (double)	scanf("%lf", &pi);
%c	Caractère (char)	scanf(" %c", &car);
%s	Chaîne (char*)	scanf("%49s", nom);
%ld	Long (long)	scanf("%ld", &grand_nombre);

2.5 – En C : lecture - écriture

► scanf : Pièges Courants

► Oublier & devant les variables :

`scanf("%d", age);` // ✗ Erreur : doit être `&age`

► Lire une chaîne sans limiter la taille :

`char nom[10];`

`scanf("%s", nom);` // ✗ Risque de débordement

► // ✓ Correct :

`scanf("%9s", nom);` // Lit max 9 caractères

2.5 – En C : lecture - écriture

► scanf : exemple

```
#include <stdio.h>

int main() {
    int age;
    float taille;
    printf("Entrez votre âge et taille: ");
    scanf("%d %f", &age, &taille);
    printf("Vous êtes âgé de %d ans et votre taille est %f m\n", age, taille);
    return 0;
}
```

2.5 – En C : lecture - écriture

► getchar : Lire un Caractère, Syntaxe

```
int getchar(void);
```

► Retourne :

- Le code ASCII du caractère lu (comme int).
- EOF (-1) en cas d'erreur ou de fin de fichier (Ctrl+D sous Linux/Mac, Ctrl+Z sous Windows).

2.5 – En C : lecture - écriture

► getchar : exemple

```
int c = getchar();
printf("Vous avez entré: %c\n", c);
```

```
d
Vous avez entré: d
```

```
printf("Entrez une ligne: ");
int c;
while ((c = getchar()) != '\n' && c != EOF) {
    putchar(c);
}
putchar('\n');
```

```
Entrez une ligne: Le prof est beau
Le prof est beau
```

2.4 – En C : Opérateurs

► Comparaison des Fonctions

Fonction	Utilisation Typique	Avantages	Inconvénients
<code>printf</code>	Affichage formaté (nombres, chaînes, etc.).	Flexible, puissant.	Plus lent pour les caractères individuels.
<code>putchar</code>	Affichage caractère par caractère.	Rapide, simple.	Limité aux caractères.
<code>scanf</code>	Lecture formatée (utilisateur/fichier).	Pratique pour les données structurées.	Risque de débordement, moins flexible.
<code>getchar</code>	Lecture caractère par caractère.	Sûr, simple, permet de traiter les entrées.	Lent pour les grandes entrées.

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

29

3 – La séquence en C

- Chaque instruction se termine par **;**
- Toutes les variables utilisées doivent obligatoirement être déclarées au préalable

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

30

3.1 – Déclaratives

- ▶ Il est aussi nécessaire de préciser ce que les variables utilisées contiendront comme type de données.
- ▶ Il peut s'agir de nombres entiers, de nombres réels, de chaînes de caractères, ...
- ▶ Il faut faire précéder la description de l'algorithme par une partie dite **déclarative** où l'on regroupe les caractéristiques des variables manipulées.

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

31

Chapitre 4 : La structure de choix

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel
2025-2026

32

4.1 - La structure alternative

- La structure alternative se présente en général sous la forme :

```
si expression alors  
    première séquence d'instructions  
sinon  
    deuxième séquence d'instructions  
fsi
```

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

33

4.2. L'alternative en C

- Si se traduit en **if**
- La condition doit être entre parenthèses
- Sinon en **else**

<pre><u>si</u> expression <u>alors</u> séquence d'instructions <u>sinon</u> séquence d'instructions <u>fsi</u></pre>	<pre>if (expression) { séquence d'instructions } else { séquence d'instructions }</pre>
--	---

LS1 STS MI-PC - Ph. Hunel

2011-2012

34

4.2 - L'alternative en C

<u>si</u> expression <u>alors</u> séquence d'instructions <u>fsi</u>	if (expression) { séquence d'instructions }
<u>si</u> expression <u>alors</u> une instruction <u>sinon</u> séquence d'instructions <u>fsi</u>	if (expression) une instruction else { séquence d'instructions }

LSI STS MI-PC - Ph. Hunel

2011-2012

35

4.2 - L'alternative en C

```
#include <stdio.h>

int NA, NB, reste;
main () {
    printf("Introduisez le nombre de doigts montrés par le joueur A : ");
    scanf("%d", &NA);
    printf("Introduisez le nombre de doigts montrés par le joueur B : ");
    scanf("%d", &NB);
    reste = (NA+NB) % 2;
    if (reste==0) {
        printf("Le joueur A a gagné\n");
    } else {
        printf("Le joueur B a gagné\n");
    }
    printf("Bravo pour le gagnant\n");
}
```

LSI STS MI-PC - Ph. Hunel

2011-2012

36

4.3 - Le choix multiple

- Supposons que l'on veuille demander à l'utilisateur de choisir dans un menu une des 3 possibilités offertes.
- Le choix présenté ne se limite pas à une alternative (soit - soit).
- Mais plutôt à une expression du type « selon que... »

LS1 STS MI-PC - Ph. Hunel

2011-2012

37

4.3 - Le choix multiple

- En LDA :
entier i
lire i
selon que
 i=1 faire bloc1
 ou que i=2 faire bloc2
 ou que i=3 faire bloc3
 autrement écrire "Mauvais choix"
Fselon
► autrement est comme dans l'alternative facultative

LS1 STS MI-PC - Ph. Hunel

2011-2012

38

4.5 - Le choix multiple

- Peut toujours s'écrire avec des alternatives :

```
entier i
lire i
Si i=1 alors
    bloc1
sinon
    si i=2 alors
        bloc2
    sinon
        si i=3 alors
            bloc3
        sinon
            écrire "Mauvais choix"
    Fsi
Fsi
```

LS1 STS MI-PC - Ph. Hunel

2011-2012

39

4.4 - Le choix multiple en C

- La traduction du choix multiple en C est assez restrictive, puisque la valeur de l'expression conditionnant le choix doit être entière (**char, short, int**).
- L'instruction **switch** permet de mettre en place une structure d'exécution qui permet des choix multiples parmi des cas de même type et faisant intervenir uniquement des **valeurs constantes entières**.

LS1 STS MI-PC - Ph. Hunel

2011-2012

40

4.4 - Le choix multiple en C

```
switch ( <expression entière> ) {  
    case <constante entière>:  
        <instruction 1>  
        ...  
        <instruction N>  
        ...  
    default :  
        <instruction 1>  
        ...  
        <instruction N>  
}
```

ATTENTION : si la dernière instruction n'est pas l'instruction **break** les autres « **case** » ainsi que le « **default** » seront exécutés

LS1 STS MI-PC - Ph. Hunel

2011-2012

41

4.4 - Le choix multiple en C

```
#include <stdio.h>  
int i;  
main() {  
    printf("Entrez votre choix : ");  
    scanf("%d",&i);  
    switch(i) {  
        case 1:printf("premier choix\n");  
                break;  
        case 2:printf("deuxième choix\n");  
        case 3:printf("troisième choix\n");  
        default:printf("Autre choix que choix 1, 2 ou 3\n");  
    }  
}
```

Entrez votre choix : 1
premier choix

Entrez votre choix : 2
deuxième choix
troisième choix
Autre choix que choix 1, 2 ou 3

LS1 STS MI-PC - Ph. Hunel

2011-2012

42

4.4 - Le choix multiple en C

```
#include <stdio.h>
int i;
main() {
    printf("Entrez votre choix : ");
    scanf("%d", &i);
    switch(i) {
        case 1: printf("premier choix\n");
                break;
        case 2: printf("deuxième choix\n");
                break;
        case 3: printf("troisième choix\n");
                break;
        default: printf("Autre choix que choix 1, 2 ou 3\n");
    }
}
```

LS1 STS MI-PC - Ph. Hunel

2011-2012

43

duree : entier
montant : entier
Taux : réel
Lire montant, duree
Si duree = 2 **alors**
| **Si** montant < 10 000 **alors**
| | taux ← 5 %
| **sinon Si** montant < 20 000 **alors**
| | | taux ← 10 %
| | **sinon Si** montant < 25 000 **alors**
| | | | taux ← 15 %
| | | **sinon** taux ← 17 %
| | **FSi**
| **FSi**
| **sinon Si** montant < 100 000 **alors**
| | taux ← 8 %
| | **sinon**
| | | taux ← 4 %
| | **FSi**
| **FSi**

Exercice

LS1 - Algorithmique/programmation renforcés ; langage C - Ph.
Hunel

2025-2026

44

Exercice

- Écrire l'algorithme qui permet de calculer le maximum de deux entiers quelconques.

45

Correction

```
Titre : Maximum
Variable a , b, max : entier
Début
  Écrire ("Saisir deux entiers a et b ")
  Lire a
  Lire b
  Si (a > b) alors
    max ← a
  Sinon
    max ← b
  Finsi
  Écrire ("le maximum de ' , a , ' et de ' , b , ' est : ' , max)
Fin
```

46

Exercices

- ▶ Nombre positif ou négatif
 - ▶ Lire un nombre et afficher s'il est positif, négatif ou nul.
- ▶ Pair ou impair
 - ▶ Lire un entier et afficher s'il est pair ou impair.
- ▶ Majeur ou mineur
 - ▶ Lire l'âge d'une personne et afficher si elle est majeure (≥ 18 ans) ou mineure

LS1 - Algorithmique/programmation renforcés ; langage C - Ph. Hunel

2025-2026

47

Exercice

- ▶ Écrire l'algorithme qui permet de déterminer le salaire mensuel d'un commercial sachant que ce salaire comporte un montant fixe de 4000 € et une commission qui représente 5% du chiffre d'affaires réalisé par mois si ce chiffre est < 30000 et de 10 % dans le cas contraire .

LS1 - Algorithmique/programmation renforcés ; langage C - Ph. Hunel

2025-2026

48

Correction

... Suite de l'algorithme

Si (CA < 30000) **alors**

Com ← CA * 0.05

Sinon

Com ← CA * 0.1

FSI

Sal ← Com + 4000

Écrire ('Le salaire mensuel est de : ', Sal, '€')

FIN

Exercice

- ▶ Compliquons un peu l'énoncé:
- ▶ La commission est calculée de la manière suivante :
 - ▶ Commission = 15% du CA quand CA > 100000
 - ▶ Commission = 10% du CA quand 30000 < CA ≤ 100000
- ▶ Dans le cas contraire pas de commission
- ▶ Écrire l'algorithme qui permet de déterminer le salaire mensuel.

Correction

... Suite de l'algorithme

Si (CA > 100000) **alors**
Com \leftarrow CA * 0.15

Sinon

Si (CA > 30000) **alors**
Com \leftarrow CA * 0.1

Sinon

Com \leftarrow 0

Finsi

Finsi

Sal \leftarrow Com + 4000

Écrire ('Le salaire mensuel est de : ', Sal , '€')

FIN

LSI - Algorithmique/programmation renforcés ; langage C - Ph.
Hunel

2025-2026

51

Compilateur C

LSI - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel
2025-2026

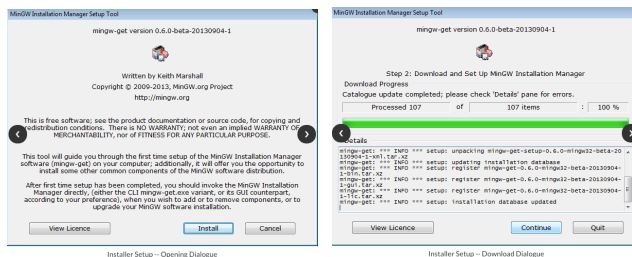
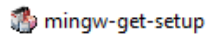
52

Installation

1. Télécharger MinGW

<https://sourceforge.net/projects/mingw/files/latest/download>

2. Exécuter le fichier



Sélectionnez juste le package de base et dans le menu Installation sélectionnez « Apply change »

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

53

Installation

1. Modifier la variable d'environnement Path en ajoutant le chemin : c:\MinGW\bin

2. Rajouter l'extension C/C++ dans votre Visual Studio code



LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

54

Chapitre 5 : La structure répétitive

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel
2025-2026

55

5.1 - La boucle "tant que"

- ▶ Fait répéter une séquence d'instructions aussi longtemps qu'une condition est **VRAI**
- ▶ En LDA :
Tant que **condition** faire
séquence d'instructions
Ftg

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

56

5.2 - La boucle " pour faire "

- ▶ Lorsque le nombre d'itération est connu
- ▶ Exemple de la table de multiplication
- ▶ En LDA :

pour **var_de_crt** ← **prem_val** à
dern_val **faire**
séquence d'instructions
fpour

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

57

5.3 – Les boucles en C

- ▶ En C, la boucle tant que se traduit par

```
while ( <expression logique> )  
    <instruction>
```

- ▶ OU

```
while ( <expression logique> ) {  
    <séquence d'instructions>  
}
```

LS1 STS MI-PC - Ph. Hunel

2011-2012

58

5.3 – Les boucles en C

```
#include <stdio.h>
int m,n,a,b,r,PGCD;
main() {
    printf("Nous allons calculer le PGCD de 2 nombres\n");
    printf("Introduisez le premier nombre : ");
    scanf("%d",&m);
    printf("Introduisez le second nombre : ");
    scanf("%d",&n);
    a=m;    b=n;
    while (b!=0) {
        r=a % b;
        a=b;    b=r;
    }
    PGCD=a;
    printf("Le PGCD de %d et %d est %d\n",m,n,PGCD);
}
```

LSI STS MI-PC - Ph. Hunel

2011-2012

59

5.3 – Les boucles en C

- En C, la boucle pour se traduit par

```
for (exp_init ; exp_cond ; exp_evol)
    <instruction>
```

- Ou

```
for (exp_init ; exp_cond ; exp_evol) {
    <séquence d'instructions>
}
```

LSI STS MI-PC - Ph. Hunel

2011-2012

60

5.3 – Les boucles en C

► **exp_init :**

- est une instruction d'initialisation ; elle est exécutée avant l'entrée dans la boucle

► **exp_cond :**

- est la condition de continuation ; elle est testée à chaque passage, y compris lors du premier ; l'instruction ou les instructions composant le corps du for sont répétées tant que le résultat de l'expression **exp_cond** est VRAI

► **exp_evol :**

- Est une instruction de rebouclage ; elle fait avancer la boucle ; elle est exécutée en fin de boucle avant le nouveau test de passage.

LS1 STS MI-PC - Ph. Hunel

2011-2012

61

5.3 – Les boucles en C

```
#include <stdio.h>
main() {
    int i,n;
    printf("Quelle table\n");
    scanf("%d",&n);
    for (i=1;i<=10;i++)
        printf("%d fois %d font %d\n",n,i,n*i);
}
```

++ : opérateur d'incrément

$i++ \Leftrightarrow i=i+1$

-- : opérateur de décrémentation

$i-- \Leftrightarrow i=i-1$

LS1 STS MI-PC - Ph. Hunel

2011-2012

62

5.3 – Les boucles en C

```
#include <stdio.h>

main() {
    int i;
    for (i=1; i<=10; i++)
        printf("%d : The teacher is the best\n", i);
    printf("-----\n");
    for (i=10; i>=1; i--)
        printf("%d : The teacher is the first best\n", i);
}
```

++ : opérateur d'incrémementation

i++ \Leftrightarrow i=i+1

-- : opérateur de décrémementation

i-- \Leftrightarrow i=i-1

LS1 STS MI-PC - Ph. Hunel

2011-2012

63

```
1 : The teacher is the best
2 : The teacher is the best
3 : The teacher is the best
4 : The teacher is the best
5 : The teacher is the best
6 : The teacher is the best
7 : The teacher is the best
8 : The teacher is the best
9 : The teacher is the best
10 : The teacher is the best
-----
10 : The teacher is the first best
9 : The teacher is the first best
8 : The teacher is the first best
7 : The teacher is the first best
6 : The teacher is the first best
5 : The teacher is the first best
4 : The teacher is the first best
3 : The teacher is the first best
2 : The teacher is the first best
1 : The teacher is the first best
```

LS1 STS MI-PC - Ph. Hunel

2011-2012

64

5.3 – Les boucles en C

- Somme des 10 premiers entiers : comparaison entre l'utilisation et de la boucle « **tant que** » de la boucle « **pour** »

```
somme = 0;
i=0;
while (i<10) {
    somme = somme + i;
    i = i + 1;
}
```

```
somme = 0;
for (i=0;i<10;i++)
    somme = somme + i;
```

5.3 – Les boucles en C

- Le langage C propose également une autre forme de la boucle tant que qui permet d'exécuter au moins une fois le corps de la boucle :

```
do
    <instruction>
while ( <expression logique> )
► OU
do {
    <séquence d'instructions>
} while ( <expression logique> )
```

5.3 – Les boucles en C

```
#include <stdio.h>

main() {
    int i, somme, N;
    somme=0;
    printf("Entrez le nombre d'élément que vous voulez sommer : ");
    scanf("%d",&N);
    i=1;
    while (i<N) {
        somme = somme+i;
        i=i+1;
    }
    printf("Somme des %d premiers entiers est : %d\n",N,somme);
}
```

LSI STS MI-PC - Ph. Hunel

2011-2012

67

5.3 – Les boucles en C

```
#include <stdio.h>

main() {
    int i, somme, N;
    somme=0;
    printf("Entrez le nombre d'élément que vous voulez sommer : ");
    scanf("%d",&N);
    i=1;
    do {
        somme = somme+i;
        i=i+1;
    } while (i<N);
    printf("Somme des %d premiers entiers est : %d\n",N,somme);
}
```

LSI STS MI-PC - Ph. Hunel

2011-2012

68

5.4 – Exercices

- Ecrire un algorithme qui demande un nombre de départ, et qui affiche ensuite les dix nombres suivants.
- Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

69

5.4 – Exercices

- Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne.
- En cas de réponse supérieure à 20, on fera apparaître un message : « *Entrez un nombre plus petit !* »,
- et inversement, « *Entrez un nombre plus grand !* » si le nombre est inférieur à 10.

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

70

5.4 – Exercices

- Ecrire un programme qui génère 2 nombres aléatoires qui ne seront pas connus de l'utilisateur.
 - Le premier nombre sera généré entre 1 et 50
 - Et le second entre 50 et 100
- Ecrire un algorithme qui demande un nombre jusqu'à ce que la réponse soit comprise entre le premier et le deuxième nombre aléatoire généré.
- En cas de réponse supérieure à premier nombre aléatoire, on fera apparaître un message : « *Entrez un nombre plus petit !* »,
- et inversement, « *Entrez un nombre plus grand !* » si le nombre est inférieur au second nombre aléatoire.
- Si l'utilisateur saisie 0, le programme s'arrête.

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

71

5.4 – Exercices

- Ecrire un algorithme et un programme Python qui affiche la figure suivante :

```
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

72

5.4 – Exercices

- Ecrivez un algorithme et un programme affichant un triangle d'étoiles. L'utilisateur entrera le nombre initial d'étoiles, et le programme affichera les lignes les unes à la suite des autres, chaque ligne perdant une étoile à chaque fois :

Nombre initial d'étoiles : 7

```
*  
**  
***  
****  
*****  
*****  
*****
```

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

73

5.4 – Exercices

- Ecrivez un algorithme et un programme affichant un triangle d'étoiles. L'utilisateur entrera le nombre initial d'étoiles, et le programme affichera les lignes les unes à la suite des autres, chaque ligne perdant une étoile à chaque fois :

Nombre initial d'étoiles : 7

```
*****  
*****  
*****  
****  
****  
***  
**  
*
```

LS1 - Algorithmique/programmation renforcés ;
langage C - Ph. Hunel

2025-2026

74

5.4 – Exercices

► Il s'agit d'afficher les différents termes de la suite définie de la manière suivante :

- Le premier terme U_1 est compris entre 4 et 10 ($4 \leq U_1 \leq 10$)
- Le terme général de la suite est :
 - $U_{n+1} = U_n/3$ si U_n est divisible par 3
 - $U_{n+1} = U_n + 1$ si le reste de la division de U_n par 3 est 1
 - $U_{n+1} = U_n - 2$ si le reste de la division de U_n par 3 est 2
- La suite s'arrête quand $U_n == 0$

Chapitre 6 : Les tableaux

6.1 – Tableaux à un indice

- Un tableau (encore appelé table ou variable indexée) est un ensemble de données, qui sont toutes de même type, désigné par un identificateur unique (le nom du tableau), et qui se distinguent les une des autres par leur numéro d'indice
- Exemple : les températures sous abri à 15h00 des jours d'une semaine seront les 7 valeurs de la variable température, qui est un tableau de 7 éléments (variables) de type réel désigné par :
 - Température[1], Température[2], ..., Température[7],

LS1 STS MI-PC - Ph. Hunel

2011-2012

77

6.2 – Tableaux à plusieurs indices

	1	2	j	n
1				
2				
3				
j				
n				

LS1 STS MI-PC - Ph. Hunel

2011-2012

78

6.3 - Les tableaux en C

► Exemple :

```
int a[13];  
char b[8][5][10];  
float d [6][15][9];
```

► Principe :

<type><identificateur>[*taille*₁][*taille*₂]...[*taille*_k];

► N'importe quelle référence à une case peut être utilisée comme une simple variable :

```
int i,j,k;  
a[i]  
b[i][j][k]
```

LS1 STS MI-PC - Ph. Hunel

2011-2012

79

6.3 - Les tableaux en C

► La *taille* correspond au nombre de cases du tableau

► Attention : les indices, permettant de localiser le contenu d'une case d'un tableau, varient entre 0 et *taille*-1

► Il est possible d'affecter un tableau à un ensemble de valeurs dès sa déclaration par :

<type><identificateur>[*taille*₁][*taille*₂]...={val₁, val₂, ...};

► Exemple :

```
int matrice[2][3]={1,2,3,4,5,6} ⇔
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

LS1 STS MI-PC - Ph. Hunel

2011-2012

80

Exemple de programme C

```
#include <stdio.h>

main() {
    int i, somme, temperature[7];
    float moyenne;
    for (i=0; i<7; i++) {
        printf("Temperature[%d]=", i);
        scanf("%d", &temperature[i]);
    }
    somme=0;
    for (i=0; i<7; i++)
        somme=somme+temperature[i];
    moyenne=somme/7;
    printf("la température moyenne de la semaine est\n", moyenne);
}
```

LS1 STS MI-PC - Ph. Hunel

2011-2012

81

6.4 – Les chaînes de caractères

- ▶ Il n'existe pas de type spécial « chaîne de caractère »
- ▶ Une chaîne de caractère est vue comme un tableau de caractère à une dimension
- ▶ Il existe cependant une notation particulière et de nombreuses fonctions spécifiques pour les chaînes de caractère

LS1 STS MI-PC - Ph. Hunel

2011-2012

82

6.4.1 – Déclaration de chaîne de caractères

- Déclaration de chaînes de caractères en LDA

chaîne NomVariable

- Déclaration de chaînes de caractères en C

char NomVariable [**<Longueur_optionnelle>**];

- Exemples

```
char NOM [20];  
char PRENOM [20];  
char PHRASE [300];
```

LS1 STS MI-PC - Ph. Hunel

2011-2012

83

6.4.1 – Déclaration de chaîne de caractères

- Espace à réserver

- •Malheureusement, le compilateur C ne contrôle pas si un octet a été réservé pour le symbole de fin de chaîne;
- •l'erreur se fera seulement remarquer lors de l'exécution du programme.

par le symbole **\0** (NUL).

- Pour un texte de **n** caractères, il faut prévoir **n+1** octets.

LS1 STS MI-PC - Ph. Hunel

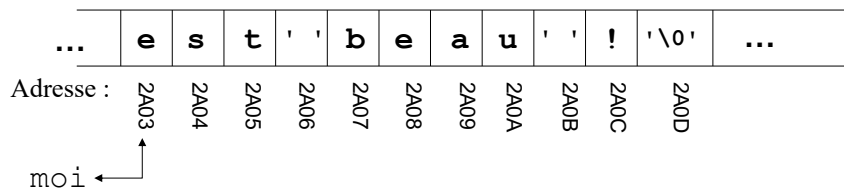
2011-2012

84

6.4.1 – Déclaration de chaîne de caractères

► Mémorisation

```
char moi[11] = "est beau !"
```



- L'adresse de la chaîne de caractères (tableau de caractères) est l'adresse du premier élément du tableau

LS1 STS MI-PC - Ph. Hunel

2011-2012

85

6.4.1 – Déclaration de chaîne de caractères

► Initialisation

► Initialisation classique d'un tableau :

```
Char moi[] = {'b','e','a','u','\0'};
```

- Facilité pour les chaînes de caractères :

```
Char moi[] = "beau";
```

- A l'initialisation par [], le système réserve automatiquement le nombre d'octets nécessaires
 - i.e. taille de la chaîne + 1
 - Pour l'exemple moi : 4 + 1 = 5 octets

LS1 STS MI-PC - Ph. Hunel

2011-2012

86

6.6.1 – Déclaration de chaîne de caractères

► Exemple

```
Char moi[] = "beau";
```

moi	:	'b'	'e'	'a'	'u'	'\0'
-----	---	-----	-----	-----	-----	------

```
Char moi[5] = "beau";
```

moi	:	'b'	'e'	'a'	'u'	'\0'
-----	---	-----	-----	-----	-----	------

```
Char moi[7] = "beau";
```

moi	:	'b'	'e'	'a'	'u'	'\0'	0	0
-----	---	-----	-----	-----	-----	------	---	---

LSI STS MI-PC - Ph. Hunel

2011-2012

87

6.6.1 – Déclaration de chaîne de caractères

► Exemple

```
Char moi[4] = "beau";
```

moi	:	'b'	'e'	'a'	'u'	☠
-----	---	-----	-----	-----	-----	---

↪ Erreur pendant l'exécution

```
Char moi[3] = "beau";
```

↪ Erreur pendant la compilation

LSI STS MI-PC - Ph. Hunel

2011-2012

88

6.4.2 – Accès aux éléments

- L'accès à un élément d'une chaîne de caractères peut se faire de la même façon que l'accès à un élément d'un tableau

```
Char moi[5];
```

Défini un tableau de 5 éléments :

```
moi[0], moi[1], moi[2], moi[3], moi[4],
```

```
Char B[5] = "beau";
```

B :

'b'	'e'	'a'	'u'	'\0'
-----	-----	-----	-----	------

B[0] B[1] B[2] B[3] B[4]

LS1 STS MI-PC - Ph. Hunel

2011-2012

89

6.4.3 – Fonctions sur de chaîne de caractères

- **strlen(<s>)** : fournit la longueur de la chaîne **sans** compter le '\0' final
- **strcpy(<s>, <t>)** : copie <t> vers <s>
- **strcat(<s>, <t>)** : ajoute <t> à la fin de <s>
- **strcmp(<s>, <t>)** : compare <s> et <t> lexicographiquement et fournit un résultat:
 - Négatif : si <s> précède <t> Zéro : si <s> est égal à <t>
 - Positif : si <s> suit <t>
- **strncpy(<s>, <t>, <n>)** : copie au plus <n> caractères de <t> vers <s>
- **strncat(<s>, <t>, <n>)** : ajoute au plus <n> caractères de <t> à la fin de <s>

LS1 STS MI-PC - Ph. Hunel

2011-2012

90