



Module: Algorithmique Et Programmation C I



ECOLE SUPÉRIEURE DE LA STATISTIQUE
ET DE L'ANALYSE DE L'INFORMATION

AÏCHA EL GOLLI

aicha.elgolli@essai.ucar.tn

Novembre-décembre 2022

ORGANISATION DU COURS

- Module de 21h : **7 semaines** x3h :
 - 1h30 cours (Mme El Golli)+ 1h30 TP (Mme Mansour) P2
 - Travail et cours sur classroom
- Contenu des chapitres:
 1. *Chapitre I : Introduction à l'Algorithmique et programmation C*
 2. Chapitre II: Les procédures et les fonctions
 3. Chapitre III: Les tableaux et les chaînes de caractères
- Modalités d'évaluation :
- Note de CC: note de TP (évaluations au niveau des TP) (35%)- Bonus/malus sur l'implication (présence / pertinence des interactions) en cours
- Examen final sur les concepts vus en cours/TPs (65%) ==>fin P2

RÉFÉRENCES BIBLIOGRAPHIQUES

J. Courtin, Initiation à l'algorithmique et aux structures de données, Edition Dunod, Juillet 1995.

L. Ammeraal, Algorithmes et structures de données en langage C, Edition InterEditions, Juillet 1996.

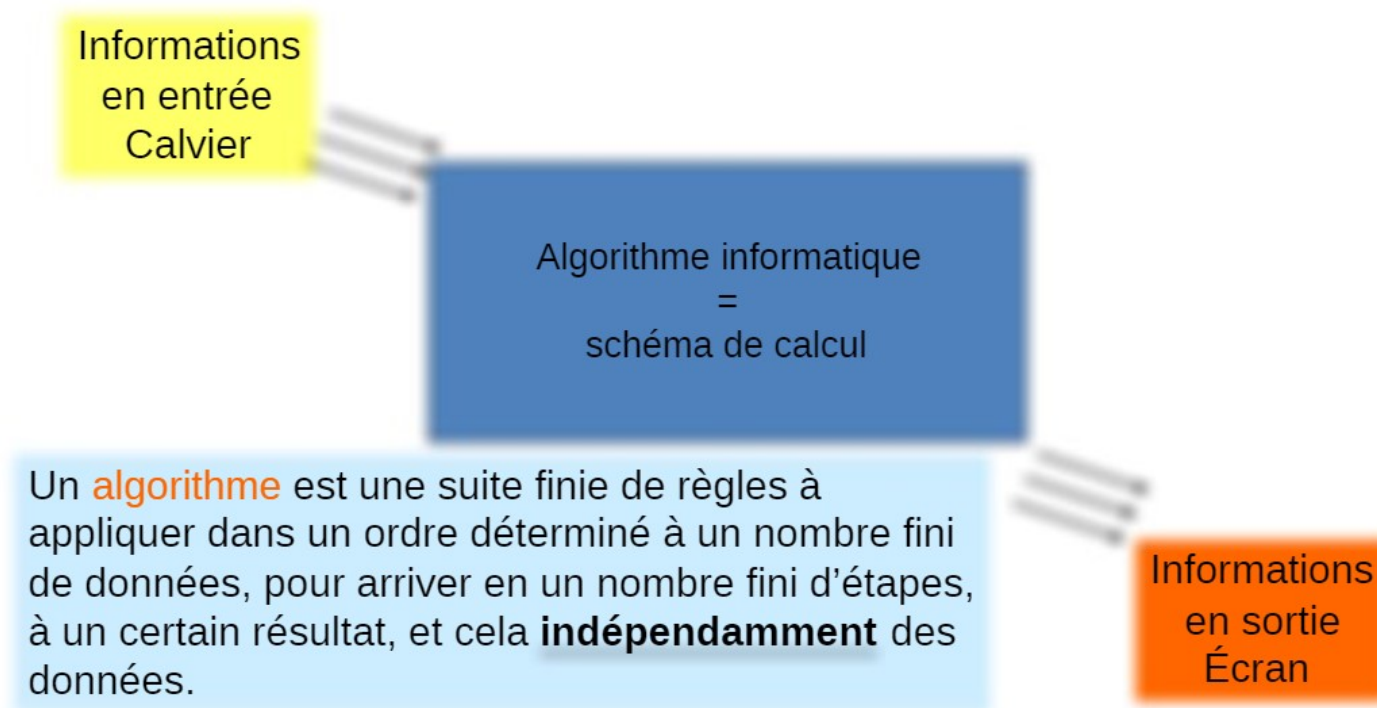
B. Kernighan, D. Ritchie, Le langage C (2^{ème} édition) , Masson, 1994.

Thomas H. Cormen, Charles E. Leireson, Ronald L Rivest et Clifford Stein, « Introduction à l'algorithmique », cours et exercices 2^{ème} cycle Ecoles d'ingénieurs », Edition Dunod, 2^{ème} édition, Paris 2002.



CHAPITRE I : INTRODUCTION À L'ALGORITHMIQUE ET PROGRAMMATION C

ALGORITHMES ET PROGRAMMES



ALGORITHMES ET PROGRAMMES

Programme :

- codage d'un algorithme afin que l'ordinateur puisse exécuter les actions décrites
- doit être écrit dans un langage compréhensible par l'ordinateur → langage de programmation: dans notre cours est le **C**

Un programme est donc une suite ordonnée d'instructions élémentaires codifiées dans un langage de programmation.

LANGAGE DE PROGRAMMATION

permet au programmeur d'écrire son programme suivant une grammaire qui peut être, soit celle du langage machine même, soit une grammaire facilement interprétable par la machine ou pouvant être traduite en langage machine au moyen d'un outil logiciel dit **compilateur** du langage. Il existe, en fait, trois catégories de langages :

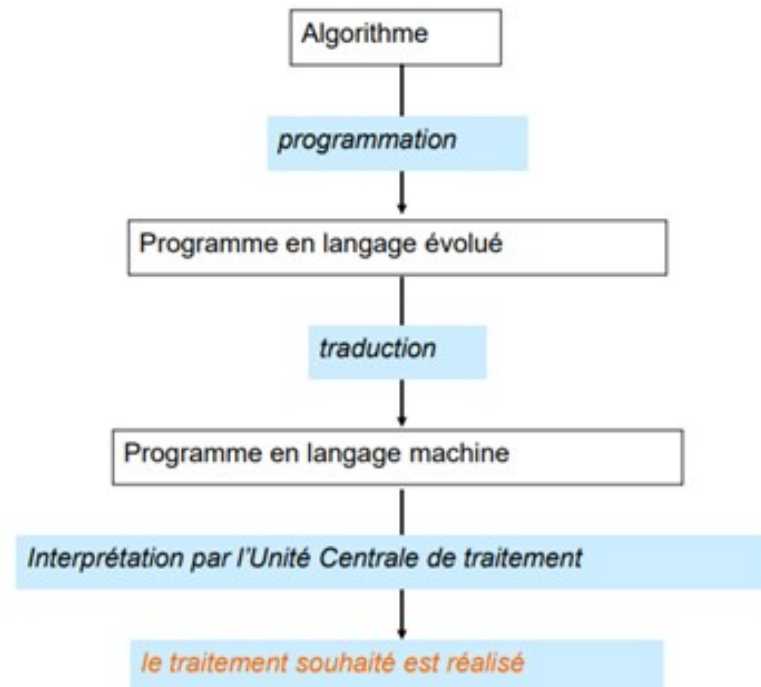
- **Le langage binaire** : il s'agit du langage machine exprimé par des chiffres (**0 ou 1**). Ce langage produit, en effet, des programmes *automatiquement consommables (compréhensibles) par la machine mais qui sont illisibles et non portables*.
- **Les langages de bas niveau (comme l'assembleur)** : ces langages produisent des programmes *facilement interprétables* par la machine mais *d'utilisation lourde* pour les programmeurs.
- **Les langages évolués/langage de haut niveau** : ils sont, d'utilisation, souples et produisent des programmes clairs et *lisibles* mais ils *doivent encore être compilés* (traduits en langage machine par un compilateur du langage) pour générer des programmes exécutables. Nous en citons: *Pascal, C, C++, Visual Basic, Java...* La famille des *langages évolués* peut être subdivisée en deux grandes catégories :
 - Les langages de scripts
 - Les langages compilés (langages interprétés)

un **langage compilé** est un langage où toutes les instructions sont traduites en **code binaire** avant d'être exécutées par un compilateur. Un **langage interprété** est un langage décodé et exécuté instruction par instruction lors de l'exécution du programme.

LE PROCESSUS DE PROGRAMMATION

Importance des algorithmes : Pour mener à bien un traitement sur un ordinateur il faut :

1. Concevoir un algorithme qui décrit comment le traitement doit être fait
2. Exprimer l'algorithme sous la forme d'un programme dans un langage de programmation adéquat
3. Faire en sorte que l'ordinateur exécute le programme : compilation



CYCLE DE VIE D'UN LOGICIEL (1)

7 étapes:

1. Spécification d'un cahier de charges
2. Conception d'un algorithme de résolution
3. Analyse du programme sous forme de spécifications
 - ✓ Spécifications externes
 - ✓ Spécifications internes
4. Codage dans le **langage choisi**

CYCLE DE VIE D'UN LOGICIEL (1)

5. Test en mise au point
6. Recette
 - ✓ Logiciel
 - ✓ Manuel d'utilisation
 - ✓ Jeu de données utilisées
7. L'évolution ultérieure du programme

ECRIRE UN ALGORITHME

Exemple : calcul de la surface d'un champ

Idée

- (i) acquérir la longueur et la largeur du champ ;
- (ii) calculer la surface ;
- (iii) afficher la surface.

Algorithme Surface

longueur, largeur, surface: entier;

DEBUT

Ecrire("longueur?");

lire(longueur);

Ecrire("largeur?");

Lire(largeur);

surface \leftarrow longueur * largeur;

Écrire(surface);

FIN

LE LANGAGE C

LE LANGAGE C

- Le langage C est né en 1972 dans les laboratoires de la Bell Telephone (AT&T) des travaux de Brian Kernighan et Dennis Ritchie.
- Il a été conçu à l'origine pour l'écriture du système d'exploitation Unix et s'est vite imposé comme le langage de programmation sous Unix.
- En 1983, l'ANSI (American National Standards Institute) décida de normaliser le langage ; ce travail s'acheva en 1989 par la définition de la norme ANSI C (C89).
- Celle-ci fut reprise telle quelle par l'ISO (International Standards Organization: Comme l'ANSI, mais au niveau mondial) en 1990 (C99)

POURQUOI LE C?

Le langage C est à l'origine de nombreux logiciels, mais surtout des plus grands systèmes d'exploitation.

Il a l'avantage d'être un langage minimaliste, et donc très proche de la machine. Il est alors plus simple à contrôler.

Il permet principalement des échanges plus rapides avec la machine et c'est pourquoi **le langage C se trouve toujours beaucoup dans les systèmes embarqués et en robotique**.

Il est intéressant d'**apprendre le langage C**, surtout quand on débute en programmation, car c'est un langage parent de beaucoup d'autres.

Le langage C est à la base de nombreux langages de programmation. Le connaître permet de réussir à apprivoiser plus facilement de nombreux environnements similaires.

LES CARACTÉRISTIQUES DU LANGAGE C

langage structuré, conçu pour traiter les tâches d'un programme en les mettant dans des blocs

programmes efficaces, il possède les mêmes possibilités de contrôle de la machine assembleur et il génère un code compact et rapide

Déclaratif, tout objet C doit être déclaré avant d'être utilisé

Format libre, la mise en page des divers composants d'un programme est totalement libre. cette possibilité doit être exploitée pour rendre les programmes lisibles

Modulaire, une application pourra être découpée en modules pouvant être compilés séparément. Un ensemble de programmes déjà opérationnels pourra être réuni dans une librairie, aptitude qui permet au langage C de se développer de lui-même

Souplesse et permissivité, peu de vérifications et d'interdits, hormis la syntaxe.

LA CHAÎNE DE PRODUCTION (1)

- ✓ On écrit un programme en C dans un fichier reconnu par le nom de la norme: **nom_du_fichier.c**
- ✓ Compilation (traduire en langage plus proche de la machine pour obtenir un exécutable). A la fin de la compilation : fichier « objet » .o
- ✓ Sous Unix la compilation du fichier qui s'appelle nom.c s'obtient à l'aide de l'instruction **gcc**

`gcc -o nom.c`

Cette instruction produit un fichier objet: **nom.o**

- ✓ Quand il y a des erreurs le fichier nom.o n'aboutit pas.

LA CHAÎNE DE PRODUCTION (2)

- ✓ En C on a des bibliothèques pour gérer les entrées/sorties
- ✓ 3 phases:
 1. Compilation
 2. Edition des liens
 3. Exécution
- ✓ La commande **gcc nom.c** effectue la compilation puis l'édition de liens.
produit un fichier exécutable nommé par défaut **a.out**
- ✓ Pour ranger l'exécutable dans un fichier autre que a.out, on écrit:
gcc -o nom nom.c
- ✓ Pour exécuter le programme on écrit: nom_du_fichier

LA CHAÎNE DE PRODUCTION (3)

Tout ceci constitue une chaîne de production en C:

- ✓ Ecrire le programme dans un fichier
- ✓ Compilation
- ✓ L'édition des liens
- ✓ Exécution



SCHÉMA GÉNÉRAL SIMPLIFIÉ D'UN ALGORITHME/PROGRAMME C

Schéma général Algorithme

ALGORITHME nom algo

/* déclaration des constantes, de types, de variable*/

Début

/*corps de l'algorithme*/

Fin

Schéma général d'un programme C

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

```
int main(){
```

```
/* Les  
instructions de votre programme */
```

```
return 0;
```

```
}
```

ALGORITHME

ALGORITHME nom algo : permet de nommer un algorithme ;

Déclarations : la partie déclarative doit se trouver en tête de l'algorithme afin de faciliter la réalisation de ce dernier et sa compréhension ;

Début et Fin : encadre le corps de l'algorithme

On déclare les variables utilisées dans un algorithme ainsi que les modules (les procédures et fonctions) utilisés dans l'algorithme.

Les déclarations de variables servent à indiquer à l'ordinateur les **identificateurs** de variables utilisés dans l'algorithme, ainsi que leur **type**.

PROGRAMME C

/ commentaire ignoré par un compilateur */* ou

// généralement réservé pour une ligne de code que l'on veut désactiver temporairement

main :

- fonction principale de tout programme C.
- marque le début et la fin de l'exécution du programme.
- le corps de **main** est entre 2 accolades {}. La partie entre les accolades est appelée un bloc.
- Tout programme écrit en C contient exactement une fonction «main».

PROGRAMME C

Les Instructions :

- Une instruction se termine toujours par un **point-virgule**.
- Un groupe d'instructions (un bloc) se trouve entre accolades.
- Règles qui améliorent la lisibilité :
 - Une instruction par ligne.
 - Décaler les instructions par rapport aux accolades.
 - Aligner les accolades verticalement.

Directives de compilation : commandes au compilateur précédées par le signe # (dièse).

Un programme commence par: # include <stdio.h> - une directive de pré-compilation

- **#include** demande au compilateur de lire et de compiler un fichier.

STDIO.H (standard input output. header) = fichier « en-tête » = bibliothèque C avec des fonctions utiles (ex: **printf**)

- **#define** définit une constante symbolique (ex #define PI 3.14). Le compilateur remplace chaque occurrence de PI par 3.14

TRADUCTION EN C

Algorithme Surface

longueur, largeur,
surface: entier;

DEBUT

Ecrire("longueur:");

lire(longueur);

Ecrire("largeur:");

Lire(largeur);

Surface <-- longueur *largeur;

Écrire("Surface= ", surface);

FIN

```
#include <stdio.h> //décrit les fonctions de lecture et affichage
```

```
main(){ //marque le début du programme
```

```
int longueur, largeur, surface;//déclarations
```

```
printf("Longueur:");
```

```
scanf("%d", &longueur);
```

```
printf("Largeur: ");
```

```
scanf("%d", &largeur);
```

```
surface=longueur*largeur;
```

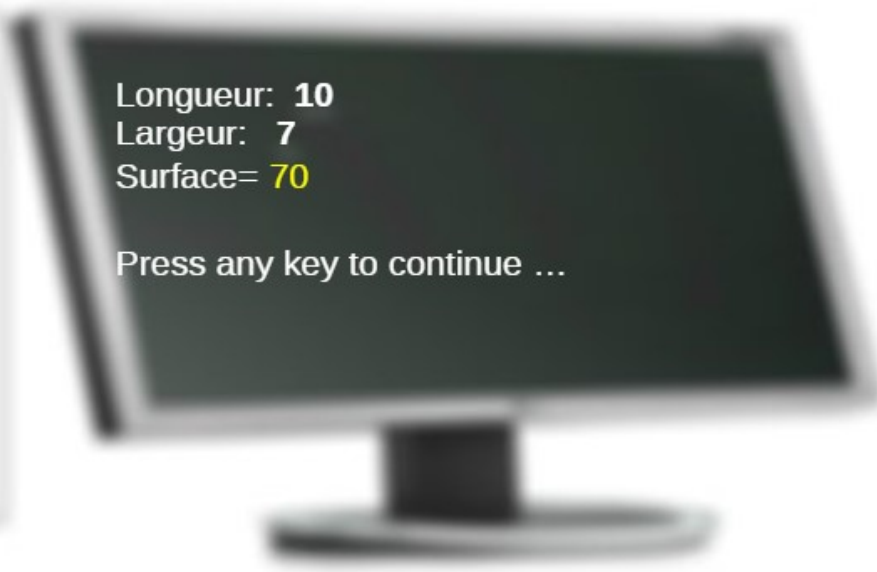
```
printf("Surface = %d \n", surface);
```

```
}
```

EXÉCUTION

```
#include <stdio.h>
//décrit les fonctions de lecture et affichage
#include<conio.h> //pour utiliser getch();
main(){ //marque le début du programme
int longueur, largeur,surface;//déclarations

    printf("Longueur: ");
    scanf("%d", &longueur);
    printf("Largeur : ");
    scanf("%d", &largeur);
    surface=longueur*largeur;
    printf("Surface = %d \n", surface);
    getch();
} //marque la fin du programme principal
```



Longueur: 10
Largeur: 7
Surface= 70

Press any key to continue ...

LES VARIABLES ET CONSTANTES

LES VARIABLES, CONSTANTES

Par définition un algorithme :

- ✓ traite une ou des variables entrantes : **les données**
- ✓ restitue une ou des variables sortantes : **les résultats**

Un algorithme qui ne satisfait pas à ces deux conditions n'a pas de justification et par conséquent n'existe pas.

Les données traitées par l'algorithme sont de deux types : variable ou constante

Chaque variable et constante doit être déclarée avant d'être utilisé : **nom + type**

Cette déclaration consiste en une réservation en mémoire de la place nécessaire et suffisante à l'hébergement de la valeur.

EXEMPLE : INSTRUCTION D'AFFECTATION + AFFICHAGE DU CONTENU D'UNE VARIABLE

ALGORITHME affichage

a : entier ;

Début

a ← 3 ;

afficher (a) ;

Fin

```
#include <stdio.h>
void main()
{
    int a ;
    a=3 ;
    printf(" %d ",a) ;
}
```

EXEMPLE : CALCUL ET AFFICHAGE D'UNE SOMME

ALGORITHME somme

a, b, c : entier ;

Début

a ← 3 ;

b ← 1 ;

c ← a + b ;

Afficher (c) ;

Fin

```
#include <stdio.h>
void main()
{
    int a,b,c ;
    a=3 ;
    b=1;
    c=a+b;
    printf(" %d ",c) ;
}
```

VARIABLE

Toute variable utilisée doit être déclarée au début dans la partie déclarative

Une variable va être caractérisé par :

- un identificateur (son nom) : pour le désigner cet identificateur doit être parlant : $q = \text{quotient} \dots$
- Un type (nature de l'objet : entier, caractère...) simple ou structuré. Un type détermine en particulier les valeurs possibles de l'objet et les opérations primitives applicables à l'objet.
- Une valeur (contenu de l'objet) unique. Cette valeur peut varier au cours de l'algorithme ou d'une exécution à l'autre : ces objets sont des variables.

SYNTAXE DÉCLARATION VARIABLE ET LES DIFFÉRENTS TYPES

<i>algorithmique</i>	C
nomvar1 [, nomvar2,..., nomvarn] : type ;	type nomvar1 [, nomvar2,..., nomvarn] ;

algorithmique	C
entier	int(4octets), long(8octets), short(2octets)
réel	float(4octets), double(8octets)
car	char (1 octet)
booléen	int (1=VRAI, 0=FAUX)
Chaîne	char nomvar [longueur de chaîne] (tableau et pointeurs)

IDENTIFICATEURS

- les noms des variables doivent **commencer** par **une lettre** et leur tailles ne dépassent pas les **32 caractères**.
 - Un identifiant doit être formé de :
 - lettres minuscules ou majuscules (de **a à z**, ou de **A à Z**),
 - **chiffres**, et
 - **caractères soulignés** (le symbole "_", appelé underscore en anglais).
 - Il ne doit en outre **pas commencer** par un **chiffre**, ni **contenir d'espaces**, de **lettres accentuées** ou tout autre **caractère spécial**.
- Ainsi **a**, **A3_B2**, **toto** sont des identificateurs valables, alors que **35**, **5A**, **A+b** ne le sont pas.
- En algorithmique il est complètement indifférent que les identificateurs soient écrits en majuscules ou en minuscules de même pour les mots réservés tels que afficher ou entrer ... ainsi TOTO, toto, Toto représentent le même identificateur.
 - En langage C les majuscules et minuscules sont différenciées.

LES MOTS CLÉS

Un certain nombre de mots, appelés mots-clefs, sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs. L'ANSI C compte 32 mots clefs :

auto	const	double	float	int	short	struct
unsigned	break	continue	else	for	long	signed
switch	void	case	default	enum	goto	register
sizeof	typedef	volatile	char	do	extern	if
return	static	union	while			

que l'on peut ranger en catégories:

– les spécificateurs de stockage

auto register static extern typedef

– les spécificateurs de type

char double enum float int long short signed struct union unsigned void

– les qualificatifs de type

const volatile

– les instructions de contrôle

break case continue default do else for goto if switch while

– divers

return sizeof

LES CONSTANTES

Une constante est une variable dont la valeur est inchangeable lors de l'exécution d'un programme. En langage C, les constantes sont définies grâce à la directive du préprocesseur *#define*, qui permet de remplacer toutes les occurrences du mot qui le suit par la valeur immédiatement derrière elle. Par exemple la directive :

```
#define _Pi 3.1415927 // (pas de ;)
```

Toutefois, avec cette méthode les constantes ne sont pas typées, il faut donc utiliser la directive *#define* avec parcimonie...

Il est ainsi préférable d'utiliser le mot clef *const*, qui permet de déclarer des constantes typées :

```
const int dix = 10;
```

```
const char v = 'v';
```

LES CONSTANTES

Caractère unique entre apostrophes

'A', 'x', 'Z', '?', ' '

Equivalent numérique (code ASCII : American Standard Code for Information Interchange) sur 7 bits donc $2^7 = 128$ caractères.

Exemple:

Caractère :	0...9	A...Z	a...z
Code ASCII :	48...57	65...90	97...122

Séquence d'échappement :

Caractères non affichables formés d'un *backslash* (\) et d'un ou plusieurs caractères.

Exemple :

\n = retour à la ligne.

\t = tabulation.

\a = alarme (un bip).

\0 = caractère nul (code ASCII 000);

Attention \0 est différent de 0 (ASCII 48)

\' = l'apostrophe.

\\ = le *backslash*.

\\" = les guillemets.

LES ENTRÉES SORTIES



ENTRÉES ET SORTIES

L'échange d'information entre l'ordinateur et les périphériques tels que le clavier et l'écran est une étape importante que tout programme utilise. En C, les bibliothèques de fonctions comprises dans les fichiers d'en-tête (header), sont incluses dans le fichier source avec la directive: **#include**.

#include <stdio.h>

stdio.h correspond au fichier d'en-tête Standard Input-Output et le ".h" le définit comme étant un header.

printf (...) : fonction **de sortie**. - écrit à l'écran les données fournies par l'ordinateur.

scanf (...) : fonction **d'entrée** (dans le programme) des données saisies à l'écran (lues).

getchar(...) : lit un caractère en entrée.

putchar (...) : affiche un caractère à l'écran.

gets (...) : lit une chaîne en entrée.

puts (...) : affiche une chaîne à l'écran.

SORTIE SUR ÉCRAN AVEC PRINTF

SYNTAXE

Le langage algorithmique utilise le mot réservé « **afficher** » pour l'affichage d'un message ou d'un résultat avec 3 formes possibles :

- Afficher (" texte ") ; → affiche la chaîne de caractère texte
- Afficher (var) ; → affiche le contenu de var
- Afficher (" la somme de ", a, " et ", b, "est" , a+b) ;

En C la syntaxe est la suivante :

```
printf (<chaîne_de_format>, <arg1>, <arg2>, ..., <argn>)
```

La chaîne_de_format comprend les paramètres d'affichage.

Paramètres d'affichage : symbole % + spécificateur de format (caractère indiquant le type de la donnée)
arg1, arg2,..., argn sont les données à afficher.

EXEMPLES DE SPÉCIFICATEURS DE FORMAT

`%c` : affiche un caractère unique

`%d` ou `%i` : un entier signé sous forme décimale

`%f` : affiche une valeur réelle avec un point décimal.

`%e` ou `%E` : affiche une valeur réelle avec un exposant.

`%x` ou `%X` : affiche un entier hexadécimal.

`%u` : affiche un entier en notation décimale non signée.

`%s` : affiche une chaîne de caractères (string).

`%g` ou `%G` : affiche une valeur réelle avec affichage de type e ou f selon la valeur.

LARGEUR MINIMALE DE CHAMPS

%4d : 4 digits "au moins" réservés pour l'entier.

%.2f : précision de 2 rangs décimaux.

Les arguments de ***printf*** sont :

Des constantes.

Des variables.

Des expressions.

Des appels de fonctions.

EXEMPLE

```
#include <stdio.h>
#include <math.h> //pour sqrt ()
void main ()
{
float i = 2.0, j = 3.0;
printf ("%f %f %f %f", i, j, i+j, sqrt(i+j));
}
```

Ici $i+j$ est une expression et $\text{sqrt}(i+j)$ est un appel de fonction.

Le programme affiche : 2.000000 3.000000 5.000000 2.236068

%f affiche donc par défaut avec une précision de 6 chiffres significatifs.

EXEMPLES

```
#include <stdio.h>
void main ()
{printf ("Le produit de %d par %d est %d.",6 ,7, 6*7);}
```

Affiche : Le produit de 6 par 7 est 42.

```
#include <stdio.h>
void main()
{
int i = 12345;
float x = 345.678;
printf("%3d %5d %8d \n", i, i, i);
printf("%3f %10f %.1f ", x, x, x);}
```

Affiche : 12345 12345 ___12345
 345.678 ___345.678 345.6

AUTRES FONCTIONS DE SORTIE

puts : écrit une chaîne de caractères suivie d'un saut de ligne.

puts ("bonjour") est équivalent à ***printf*** ("bonjour \n")

putchar : écrit un seul caractère (sans \n).

putchar (a) est équivalent à ***printf*** ("%c", a)

ENTRÉE SUR CLAVIER AVEC SCANF

Le langage algorithmique utilise le mot réservé « **entrer** »/ « **lire** » pour la lecture au clavier.

En langage C **scanf** est une fonction définie dans `stdio.h`

Syntaxe

```
scanf (<chaîne de format>, <adresse 1>,<adresse 2>,...);
```

Même chaîne de format que **printf** : `%<lettre>`

Après la chaîne de format **scanf** n'accepte que des **adresses**.

EXEMPLE

```
#include <stdio.h>
void main()
{
    float a, b, quotient;
    printf("Entrez 2 nombres:");
    scanf("%f %f", &a, &b);
    quotient = a / b;
    printf("Le quotient est %f \n", quotient);}
```

L'opérateur d'adresse "&" passe les adresses de a et de b à **scanf**. On écrit à l'écran deux nombres séparés par un espace blanc.

Si virgule entre les spécificateurs : **scanf** ("%d ,%d", &a, &b); => on écrit deux nombres séparés par une virgule.

Entre le pourcentage (%) et la lettre (d, f, s, x, e) on peut inclure d'autres spécifications:

Exemple : la largeur du champ décimal %4d " 3496 | 21 (seuls les 4 premiers digits sont lus)

ENTRÉE D'UNE CHAÎNE AVEC *SCANF*

ALGORITHME bonjour2

nom : chaîne ;

Début

afficher ("Quel est votre nom ? ");

entrer (nom) ;

afficher ("Hello" , nom, " !");

Fin

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

```
main()
```

```
{
```

```
char nom[30] ;
```

```
printf ("Quel est votre nom ? ");
```

```
scanf ("%s", nom) ;
```

```
printf ("Hello %s !\n", nom) ;
```

```
}
```

"nom" est un tableau de caractères.

La valeur de "nom" est l'adresse même du tableau. On n'utilise donc pas l'opérateur "&" devant "nom".

SOLUTION AVEC *GETS*

Le problème est que si l'on tape un nom complet (Exemple : Anne Dupont) seul "Anne" s'affiche car "scanf" s'arrête au 1^{er} espace blanc.

La fonction **gets** est l'abréviation de GET String, elle permet de récupérer une chaîne de caractères saisie.

Exemple :

```
#include <stdio.h>
void main()
{
    char nom[10];
    printf ("Quel votre nom ? ");
    gets (nom);
    printf ("Hello %s!\n", nom);
}
```

La fonction **gets** lit tout jusqu'à validation avec la touche Entrée. La fonction **getch()** lit un caractère unique . Elle renvoie le caractère lu.

Exemple :

```
char c;
c = getch (); // affectation à la variable c du caractère lu
```

EXERCICE

Traduire cet algorithme suivant en langage C :

Algorithme EntreeSortie

a, b : caractère ;

c : entier ;

d : réel ;

Début

a ← '4' ;

b ← 'c' ;

c ← 3 ;

d ← 3,14 ;

Afficher (" le caractère a vaut : ", a) ;

Afficher (" le caractère b vaut : ", b) ;

Afficher (" l'entier c vaut : ", c) ;

Afficher (" le réel d vaut : ", d) ;

Fin ;

LES OPÉRATEURS ET EXPRESSIONS

EXPRESSIONS

une expression est un ensemble d'opérande et d'opérateurs qui possède une valeur dans un type donné :

- un opérande isolé est déjà une expression par lui-même
- toute expression en Algorithmique/langage C possède une valeur
- le type d'une expression dépend du type de ses opérandes

Exp : 1) $2+3$;
2) $x+y$;
 $(x+y)/z$;
3) $\text{float } z=3.14$; $\Leftrightarrow z:\text{reel} ; z \leftarrow 3,14$;
 $\text{long } x=2, y$; $\Leftrightarrow x, y:\text{entier} ; x \leftarrow 2$;
 $y=x+z$; $\Leftrightarrow y \leftarrow x+z$;

LES OPÉRATEURS

Les opérateurs :
arithmétiques, relationnels
et logiques

Opérateurs arithmétiques

Opérateurs Algorithmique	Opérateurs C	Signification	Exemple
+	+	Addition	$X+Y$
-	-	Soustraction	$X-Y$
*	*	Multiplication	$X*Y$
div	/	Division	X/Y
mod	%	Reste de la division	$X\%Y$
-	-	Négation	$-X$

OPÉRATEURS ARITHMÉTIQUES

`reste_de_la_division = 20%3` affecte la valeur 2 à la variable "reste_de_la_division".

Le type du résultat dépend du type des opérandes, par exemple, si on fait une division entre deux variables de type `int` (entier), le résultat sera un entier. En revanche, si l'un des deux opérandes est un réel, le résultat sera un réel.

```
int a = 10, b = 11;
```

```
float c = 11, res1, res2;
```

```
res1 = b / a;
```

```
res2 = c / a;
```

```
printf("res1 = %f \n res2 = %f\n", res1, res2);
```

```
/* affichera :
```

```
res1 = 1.000000
```

```
res2 = 1.100000 */
```

LES OPÉRATEURS

Les opérateurs :
arithmétiques, relationnels
et logiques

Opérateurs relationnels

Opérateurs Algorithmique	Opérateurs C	Signification	Exemple
= ou =	==	égalité	X==Y
<	<	inférieur	X<Y
<=	<=	Inférieur ou égal	X<=Y
>	>	supérieur	X>Y
>=	>=	Supérieur ou égal	X>= Y
!= ou < >	!=	Différent	X !=Y

OPÉRATEURS RELATIONNELS

Le résultat de l'évaluation d'une expression comportant un opérateur relationnel renvoie vrai ou faux, malheureusement en C le type booléen n'existe pas. En effet, le langage C interprète la valeur vrai et faux de la manière suivante :

- une expression est logiquement fausse (False) si valeur est 0 ;
- une expression est logiquement vrai (True) si sa valeur est différente de 0.

```
/* Programme Vrai.c */
#include <stdio.h>
void main ()
{
    printf("C évalue %d comme vrai\n", 2==2); /* 2==2 renvoie 1 */
    printf("C évalue %d comme faux\n", 2==4); /* 2==4 renvoie 0 */
    if (-33)
        printf("C évalue %d comme vrai\n", -33);
}
```

Résultat à l'écran ??

LES OPÉRATEURS

**Les opérateurs :
arithmétiques, relationnels
et logiques**

Opérateurs logiques : (Not, and, or)

Opérateurs Algorithmique	Opérateurs C	Signification	Exemple
ET	&&	Et logique	X&&Y
OU	(touche 6 : Alt Gr+6)	Ou logique	X Y
!	!	Négation logique	! X

Remarque : toute expression logique en C comportant des opérateurs logiques doit être comprise entre des parenthèses.

En C l'évaluation des expressions logiques est effectuée de gauche à droite. En effet, dans le et logique si le 1^{er} opérande est évalué à faux le compilateur ne vérifie pas la valeur de vérité du 2^{ème} opérande, par contre dans le ou logique, si le 1^{er} opérande est évalué à vrai alors le compilateur ne vérifie pas la valeur de vérité du 2^{ème} opérande.

LES OPÉRATEURS

**Opérateurs d'affectation,
incrémentaux et d'affectation
composée**

Opérateurs arithmétiques

Opérateurs Algorithmique	Opérateurs C	Signification	Exemple
\leftarrow	=	Affectation	$X=Y$
++	++	Incrémentation de 1	$X++ \Leftrightarrow X=X+1$
--	--	Décrémentation de 1	$X-- \Leftrightarrow X=X-1$
+=	+=	Affectation avec addition	$X+=Y \Leftrightarrow X=X+Y$
-=	-=	Affectation avec soustraction	$X-=Y \Leftrightarrow X=X-Y$
*=	*=	Affectation avec multiplication	$X*=Y$
/=	/=	Affectation avec division	$X/=Y$
%=	%=	Affectation avec division	$X\%=Y$

EXEMPLE

```
int i = 1, j = 1 ;  
i = j + 1 ;  
j = i ++ ;          /* j = i ; i++ ; */  
i = ++ j ;          /* j ++ ; i = j ; */  
j = i -- ;          /* j = i ; i -- ; */
```

Remarque : dans une instruction d'affectation comportant un opérateur (++) ou (- -) l'ordre est important pour le savoir il suffit d'avoir la position de ces opérateurs par rapport à l'opérande :

1-si l'opérateur vient avant l'opérande et après = d'affectation c'est l'incrément ou la décrémentation qui est faite en 1^{er} lieu en second lieu il y aura l'affectation

2-par contre si les opérateurs (- - ou ++) viennent après l'opérande c'est l'affectation qui est faite en 1^{er} lieu et l'incrément ou décrémentation qui est faite en second lieu.

OPÉRATEUR CONDITIONNEL (? :)

Expression conditionnelle à évaluer: si vraie <action1> si non<action2> équivaut à l'instruction conditionnelle IF - ELSE :

```
val = ( val >= 0 ) ? val : -val ;
```

```
Si ( val >= 0 ) Alors  
    val=val  
Sinon  
    val=-val  
FinSI
```

```
if (val >= 0)  
    val = val;  
Else  
    val = -val;
```

PRIORITÉ DES OPÉRATEURS

--> Dans chaque classe de priorité, les opérateurs ont la même priorité. Si nous avons une suite d'opérateurs binaires de la même classe, l'évaluation se fait en passant de la gauche vers la droite dans l'expression.

<-- Pour les opérateurs unaires (!, ++, --) et pour les opérateurs d'affectation (=, +=, -=, *=, /=, %=), l'évaluation se fait de droite à gauche dans l'expression.

Priorité	Opérateurs	Description	Associativité
15	() [] -> .	opérateurs d'adressage	->
14	++ --	incrément/décrément	<.
	~	complément à un (bit à bit)	
	!	non unaire	
	& *	adresse et valeur (pointeurs)	
	(type)	conversion de type (cast)	
	+-	plus/moins unaire (signe)	
13	* / %	opérations arithmétiques	->
12	+ -	""	->
11	<< >>	décalage bit à bit	->
10	< <= > >=	opérateur relationnels	->
9	== !=	""	->
8	&	et bit à bit	->
7	^	ou exclusif bit à bit	->
6		ou bit à bit	->
5	&&	et logique	->
4		ou logique	->
3	?:	conditionnel	<.
2	= += -= *= /= %= >>= <<= &= ^= =	assignations	<.
1	,	séparateur	->

EXERCICES

$a+b*c$	$a+(b*c)$
$a*b+c\%d$	$(a*b)+(c\%d)$
$-c\%d$	$(-c)\%d$
$-a+c\%d$	$(-a)+(c\%d)$
$-a/-b+c$	$((-a)/(-b))+c$

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

```
void main()
```

```
{
```

```
int a,b,c;
```

```
printf ("entrez les valeurs de a, b, c");
```

```
scanf ("%d %d %d", &a, &b, &c );
```

```
c += (a>0 && a<=10) ? ++a : a/b;
```

```
printf ("%d",c);}
```

Executez avec les valeurs entrées : 4 6 8
et 12 6 8

LES STRUCTURES DE CONTRÔLE

En algorithmique / C il y a :

- Structures conditionnelles ;
- Structures itératives

STRUCTURES CONDITIONNELLES

3 structures conditionnelles qui sont :

- Le if ;
- Le if ... else
- Le switch

1) Algorithmique

Si (condition) **alors**

Suite d'instructions

Finsi ;

Exemple : si (trouve) alors

Pos ← i ;

Finsi ;

2) Algorithmique

Si (condition) **alors**

Suite d'instructions

Sinon

Suite d'instructions

Finsi ;

Exemple : si (trouve) alors

Pos ← i ;

Sinon

i ← i + 1

Finsi ;

Traduction en C

if (condition)

[{ \ *si on a plus qu'une instruction*\

Suite d'instructions

}]

if(trouve)

pos = i ;

Traduction en C

if (condition)

[{

Suite d'instructions

}]

else

[{

Suite d'instructions

}]

if(trouve)

pos = i ;

else

i++ ;

STRUCTURES CONDITIONNELLES

3) Algorithmique

Selon /Cas (variable/expression)

Choix1 : instructions ;

Choix2 : instructions ;

Choix3 : instructions ;

...

Sinon/autrement

Instructions

FinSelon/ FinCas

Traduction en C

switch (variable/expression)

{

case choix1 :{suite d'instructions ;

break ;}

case choix2 :{suite d'instructions ;

break ;}

case choix3 :{suite d'instructions ;

break ;}

...

default :

Instructions

}

Break = instruction d'arrêt

Exercice : écrire un programme C qui selon la valeur saisie de 1 à 7 affiche le jour de la semaine correspondant.

EXPR1 ? EXPR2 : EXPR3

Une expression conditionnelle a trois opérandes:

expr1 ? expr2 : expr3

Elle peut être utilisée n'importe où dans une expression. Si *expr1* est vrai, l'expression conditionnelle vaut *expr2* sinon *expr3*.

Exemple

Ce programme affiche le maximum de deux nombres saisis au clavier.

/ Programme affichant le maximum de deux nombres entiers */*

#include <stdio.h>

void main()

{

int nb_1, nb_2;

printf("Donnez un nombre entier > ");

scanf("%d",&nb_1);

printf("Donnez un autre nombre entier > ");

scanf("%d",&nb_2);

printf("Le maximum de %d et %d est %d\n", nb_1, nb_2, (nb_1 > nb_2) ? nb_1 : nb_2);

STRUCTURES ITÉRATIVES

Les structures itératives sont : **for, while, do while**

1) schéma : for (pour)

POUR $i \leftarrow i_1$ à i_2 {PAS p } FAIRE

 Instructions ;

finPour ;

en C:

for (variable = i_1 ; variable < i_2 ; variable +=PAS)

[{

 Instructions;

}]

exemple: on désire calculer la somme de la série de 1 à 5

```
int somme=0, i ;
```

```
for(i = 1 ; i <= 5 ; i+=1)
```

```
    somme+= i ;
```

BOUCLE FOR (C/PYTHON)

Langage C

```
for(i=3; i<8 ; i++)  
printf(" %d, ",i);
```

```
for(i=15; i>10 ; i--)  
printf("%d, ",i);
```

Langage Python

```
for i in range(3,8):  
    print(i, end = ",")  
Affichage à l'exécution :  
3,4,5,6,7,
```

```
for i in range(15,10,-1) :  
    print(i, end = ",")  
génère la suite 15,14,13,12,11,
```

STRUCTURES ITÉRATIVES

2) schéma: while (tant que)

tantque (condition) faire

suite d'instructions

finfaire ;

en C :

while (condition)

[{

suite d'instructions

}]

exemple :

```
int i =1, somme =0 ;
while(i <= 5)
{
    somme +=i ;
    i ++;
}
```

STRUCTURES ITÉRATIVES

Remarques :

1) D'une manière générale nous pouvons dire que:

```
for (expression_1 ; expression_2 ; expression_3) instruction
```

Équivalent à :

```
expression_1 ;  
while (expression_2)  
{  
    Instruction  
    expression_3 ;  
}
```

2) chacune des 3 expressions (de la boucle for) est facultative. Ainsi, ces constructions sont équivalentes à l'instruction for de notre premier exemple de programme :

```
i=1 ;  
for( ; i<=5 ; i++) { somme+=i ;}
```

```
i=1;  
for( ; i<=5; ){ somme+=i;  
                i++;}
```

3) lorsque l'expression_2 est absente, elle est considérée comme vraie.

STRUCTURES ITÉRATIVES

3) schéma: do... while (repete jusqu'à)

Répéter

Séquence d'instructions ;

Jusqu'à (condition)

En C:

Do

[{

Séquence d'instructions ;

[}] while (condition) ;

int i=1, somme = 0 ;

Répéter

Somme ← somme + i ;

i ← i+1 ;

jusqu'à (i>5)

do{

somme += i ;

i ++ ;

}while(i <= 5) ;

le schéma itératif do...while n'est pas un vrai schéma répéter jusqu'à. La traduction d'un schéma algorithmique répéter jusqu'à se fera en C par do...while (condition) tout en inversant la condition algorithmique :