

Programmation 1 :

Bases de la programmation

- 7 TD de 2 h
+ 1 TD noté (20pt) de 2h
- 6 TP de 2h avec compte-rendu évalués (total 3 pt)
à réaliser en *binôme*
+ 1 TP *individuel* noté (17pt) de 2h

→ *Sera suivi de de : Programmation 2 (consolidation des bases)*



TD : A. Sigayret, S. Sidibé



Programmation 1 :

Bases de la programmation

- **Plan**

I – Présentation générale

II – Opérations de bases

III – Structures de contrôle

IV – Fonctions

V – Tableaux

Programmation 1 :

I – Présentation générale

- **Objectifs**

- Concevoir un algorithme à partir d'un problème simple,
- Transcrire un algorithme dans un langage structuré (langage C),
- Ecrire, compiler, corriger et tester un programme,
- Maîtriser les bases du langage C.

- **Mots-clés**

Problème, algorithme, programme (code source), compilation, exécution, macro-instruction, entrée-sortie, variable, donnée élémentaire, typage, déclaration, définition, structure de contrôle, type structuré, effet de bord, variable globale, variable locale...

Programmation 1 :

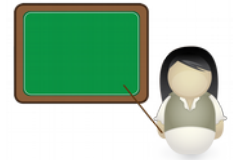
I – Présentation générale

- **Elaboration d'un programme**



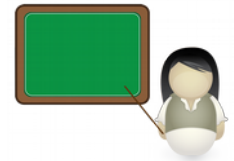
1. Formaliser le problème

- Enoncer et comprendre le problème,
- Définir les solutions attendues.



2. Formaliser un algorithme solution

- Définir les données (valeurs initiales, entrées, hypothèses), et les résultats (sorties attendus),
- Choisir les traitements.



3. Mettre en œuvre la réalisation

- Choisir la plateforme de programmation et le langage,
- ***Concevoir le programme et le vérifier,***
- Effectuer des essais (tests).



Programmation 1 :

I – Présentation générale

- **Programme** : ensemble cohérent d'instructions informatiques de traitements de donnée pour obtenir un résultat.
- **Logiciel** : ensemble de programmes réalisant une fonctionnalité.
- **Suite logicielle** : groupement de logiciels.

- **Réalisation d'un programme** :
 - Choisir un langage de programmation (langage C),
 - Ecrire le code source du programme transcrivant l'algorithme choisi,
 - Compiler ce code source (corriger les erreurs),
 - Faire tourner le programme sur des exemples bien choisis (tester).

Programmation 1 :

I – Présentation générale

- **Concrètement** : programmation en C (mode console* sous Linux)

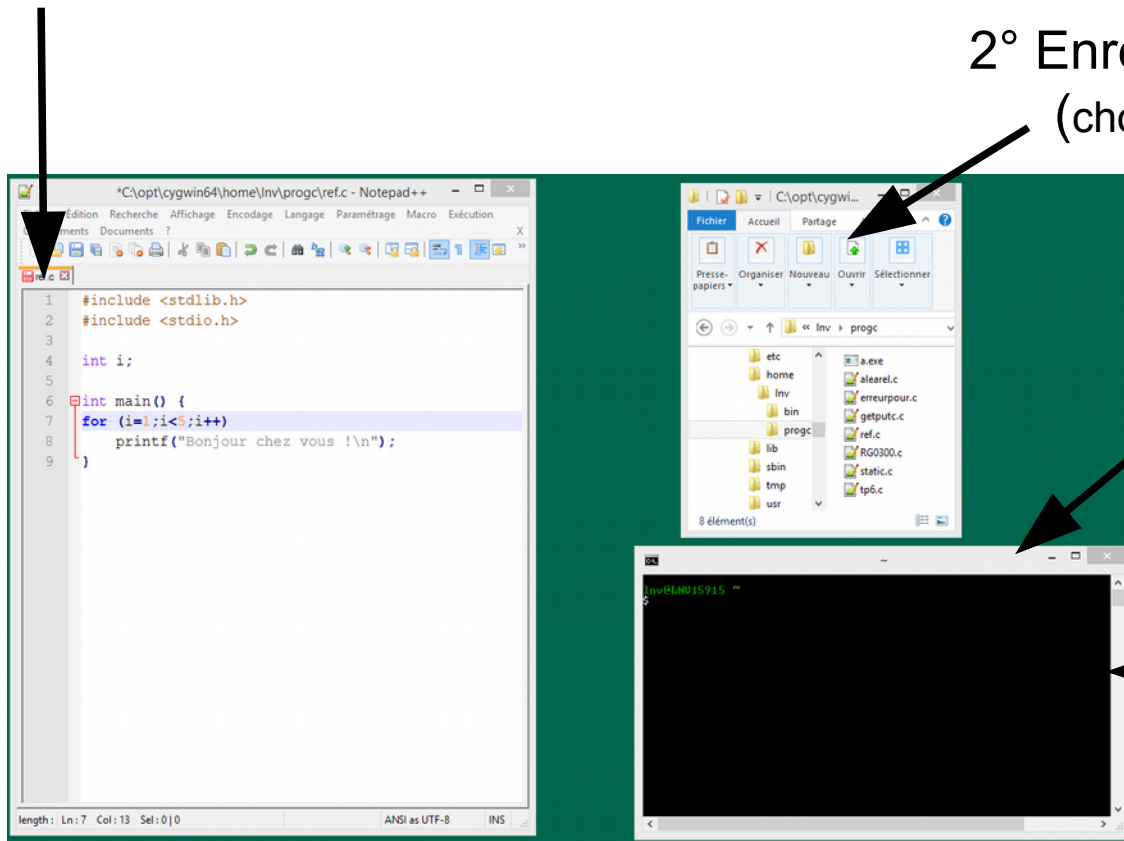
1° Dans un éditeur de texte, rédiger le code source

2° Enregistrer ce fichier *nomfichier.c*
(choisir un nom avec l'extension .c)

3° Dans un terminal*, compiler ce fichier
avec gcc

4° Exécuter ce fichier
(dans le terminal)

* console = terminal



Programmation 1 :

I – Présentation générale

- **Concrètement** : programmation en C (mode console sous Linux)

1° Edition du **code source** : choisir un éditeur de texte convivial
(avec *coloration syntaxique*)

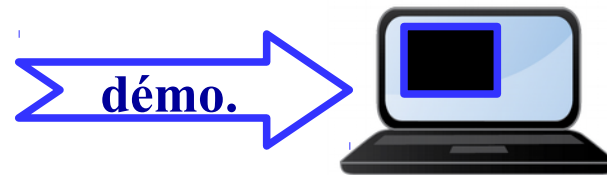
2° Enregistrement : choisir un dossier de travail

3° Compilation : `gcc nomfichier.c`
(on obtient* un fichier **exécutable** nommé `a.out`)

4° Exécution du programme : `./a.out`

Programme C : exemple

```
#include <stdio.h>
int main() {
    printf("Bonjour\n");
}
```



*N.B. `gcc nomfichier.c -o nomexec` donne un exécutable nommé *nomexec*

Programmation 1 :

I – Présentation générale

- **Du problème au programme**

1° Un problème

Problème MonProblème

Donnée : ...

Résultat : ...

2° Un algorithme résolvant ce problème

Algorithme MonAlgo

Donnée : ...

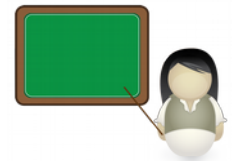
Résultat : ...

(description des traitements)

3° Un programme implémentant cet algorithme

- code source : *pour humains & machines*

- code exécutable : pour machines

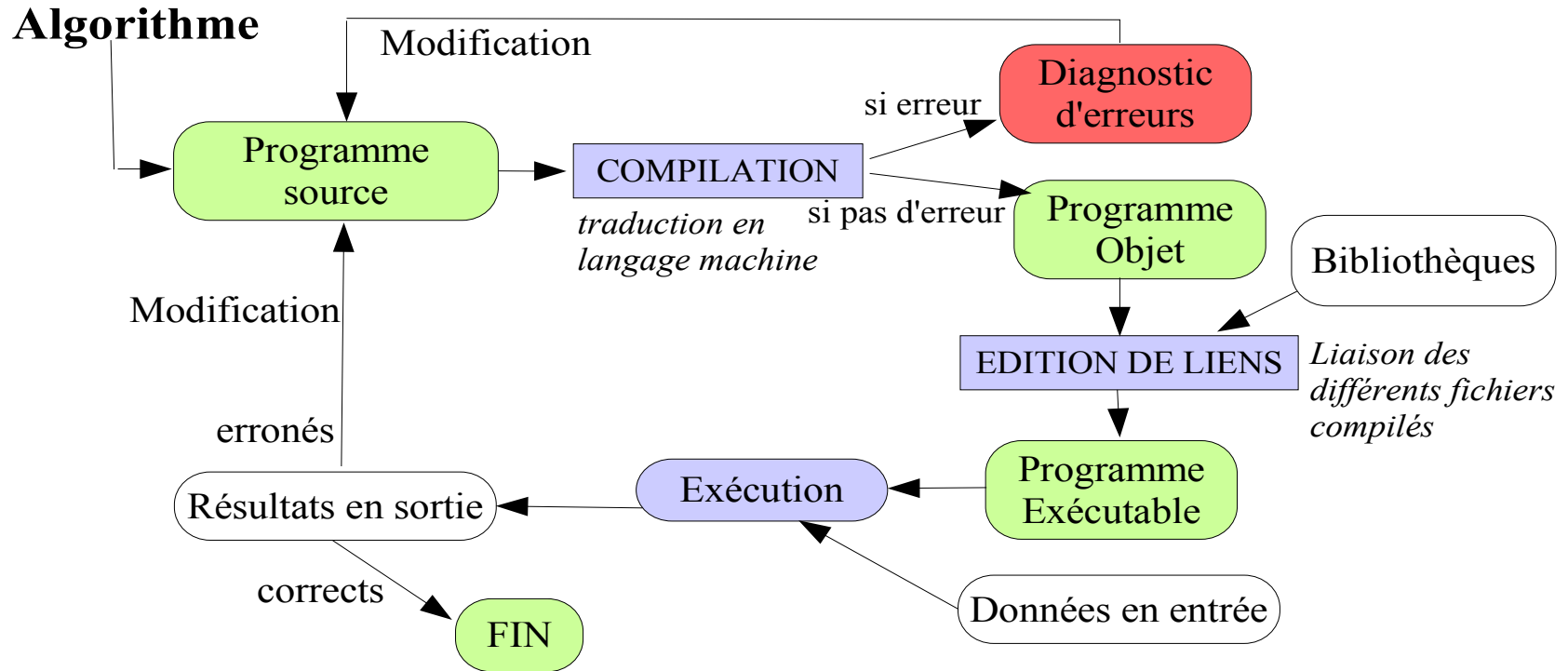


Programmation 1 :

I – Présentation générale

- Cycle de d'élaboration d'un programme

© F. Jacquet



La conception algorithmique traite les difficultés conceptuelles avant les difficultés et particularités matérielles, réduisant ainsi le cycle d'un programme

Programmation 1 :

I – Présentation générale

- **Ecrire un algorithme : structure générale**

Algorithme *NomDeLAlgorithme*

Donnée : *ce qui est fourni à l'algorithme pour qu'il fonctionne.*

Résultat : *ce que l'on obtient à la fin de l'algorithme.*

Description de l'algorithme

première instruction ;

...

Si condition alors

conséquence - 1ere instruction ;

...

conséquence – dernière instruction

sinon

alternative ... ;

...

dernière instruction.

← *barre verticale pour
chaque bloc d'instructions*

← *instructions de contrôle
prédéfinies (en gras ici)*

← *indentations : imbrications
standard des blocs*

Programmation 1 :

I – Présentation générale

- **Description de l’algorithme : choisir un niveau d’écriture**
 - **Schéma algorithmique** : description conceptuelle ± informelle

Exemple : Schéma Algorithmique MaTarteDélicieuse

Le sucre sera saupoudré sur la pâte à tarte étalée et la cannelle sur la compote de pomme (sous les pommes).

- **Algorithme (cas général)** : description détaillée et formelle

Exemple : Algorithme MaTarteDélicieuse*

Donnée : pâte à tarte**, compote, pommes, sucre, cannelle.

Résultat : une tarte délicieuse.

Étaler la pâte ; saupoudrer le sucre ; étaler la compote ; saupoudrer la cannelle ; placer les pommes coupées en tranches ; cuire au four.

* On pourrait donner plus de détails : quantités, température et durée de cuisson, etc.

N.B. on ne précise pas four à gaz ou électrique, variété de pommes, etc.

** Un algorithme de pâte à tarte est disponible par ailleurs.

Programmation 1 :

I – Présentation générale

- **Description de l'algorithme : choisir un niveau d'écriture**

– **Algorithme précisé** : on distingue des fonctions et des procédures

Exemple : Fonction Pâte à tarte

Entrée : 200g de farine, 50g de beurre, 1 œuf, 5g sel.

Sortie : une pâte à tarte prête à être étalée.

Variables d'usage : un rouleau à pâtisserie, ...

...

Retourner pâte.

Programmation 1 :

I – Présentation générale

– **Algorithme précisé** : choisir fonction ou procédure

Fonction AvecTVA

Entrée : le prix HT x .

← *mots-clés différents*

Sortie : le prix TTC correspondant.

Variables d'usage : un réel positif tva .

← *préciser les variables locales*

$tva \leftarrow 20$;

Retourner $x \cdot (1 + tva/100)$.

← *une fonction **retourne** une valeur*

Procédure AvecTVA

Donnée : le prix HT x .

Résultat : tcc est le prix TTC correspondant.

Variables d'usage : un réel positif tva .

← *préciser les variables locales*

$tva \leftarrow 20$;

$tcc \leftarrow x \cdot (1 + tva/100)$.

← *une procédure agit par **effet de bord***

Programmation 1 :

I – Présentation générale

- Traduction d'un algorithme en programme C (exemple)

Procédure AvecTVA

Donnée : le prix hors-taxe x.

Résultat : ttc est le prix TTC correspondant.

Variables d'usage : un réel positif tva.

tva ← 20 ;

Saisir h ;

ttc ← $x * (1 + tva / 100)$;

Afficher ttc.

```
// Programme PrixTTC
#include <stdio.h>

float x, ttc, tva;

int main() {
    tva=20;
    scanf("%f", &x);
    ttc=x*(1+tva/100);
    printf("TTC: %f\n", ttc);
}
```

Tester ce programme :

- copier le code source dans un fichier *ttc.c*
- dans un terminal : `gcc ttc.c`
- dans ce même terminal : `./a.out`
- saisir un prix hors-taxe



Programmation 1 :

I – Présentation générale

- **Caractéristiques d'un algorithme**

De haut niveau (→"portable")

- + Traduisible dans n'importe quel langage de programmation
- + Indépendant des détails techniques (choix de langage, de S.E.)

Précis (→non-ambigu)

- + Aucun élément ne doit porter à confusion

Concis (→homogène)

- + Au plus une page A4
(sinon, décomposer le problème en plusieurs sous-problèmes)

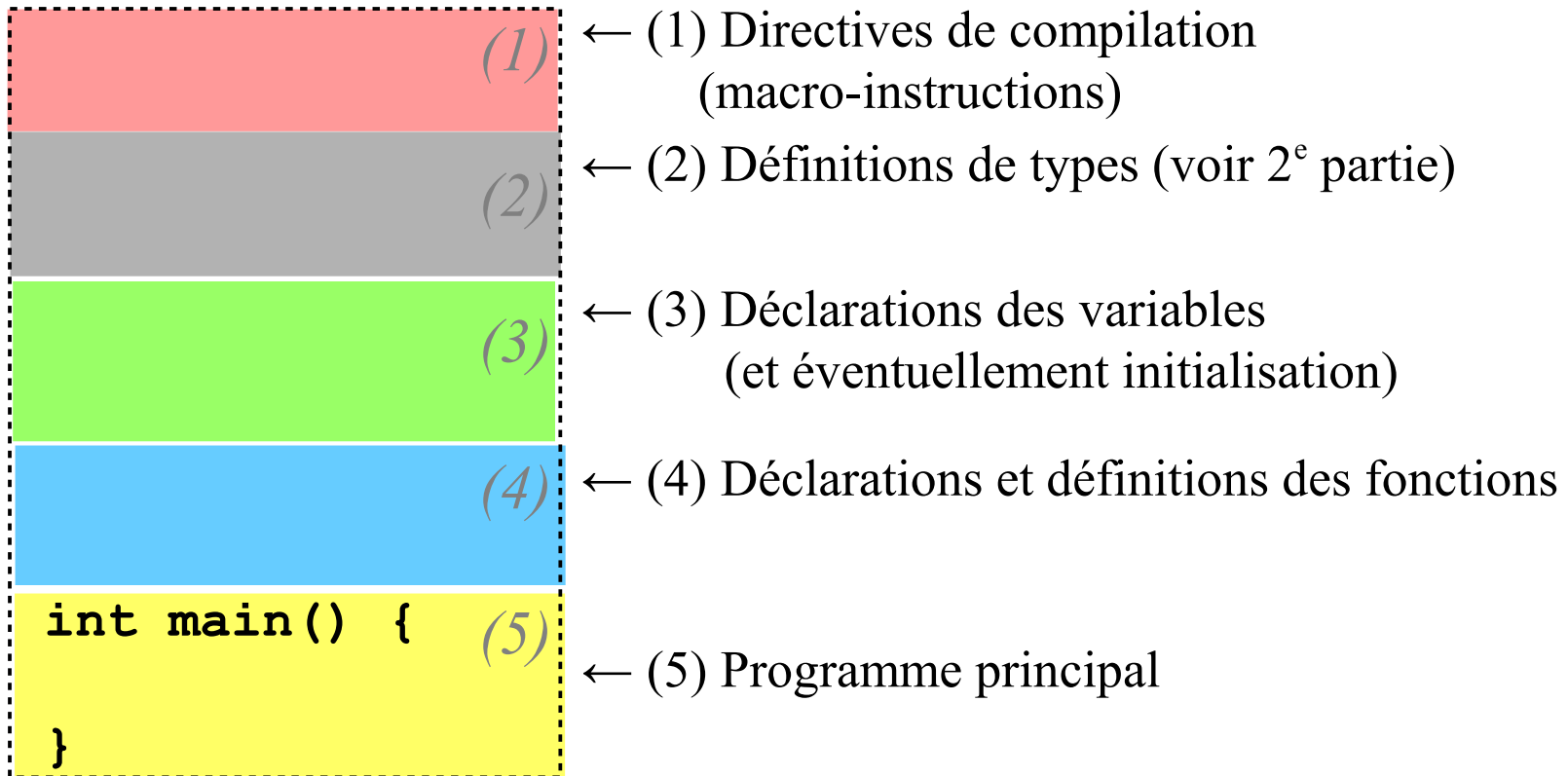
Structuré (→lisible)

- + Différentes parties facilement identifiables
- + Indentation des blocs imbriqués

Programmation 1 :

I – Présentation générale

- **Organisation générale d'un code source en C**



Programmation 1 :

I – Présentation générale

- **Pourquoi le langage C ?**

- proche de la machine,
 - très utilisé (puissance, habitude),
 - autres langages de même syntaxe (C++, Java, PHP, ...).
- ⇒ *langage à connaître*

- **Historique du langage C**

- 1972 : écrit par D. Ritchie et K. Thompson pour Unix (Bell Lab.),
 - 1978 : stabilisé par B. Kernigan,
 - 1989 : norme ANSI (C89),
 - 1990 : norme ISO/CEI (C90),
 - ... (C99, C11).
- ⇒ *langage ancien d'usage parfois compliqué*



**Apprentissage des bases de la programmation avec le langage C
comme moyen mais pas comme finalité (pas de concours de code abscons)**

Programmation 1 :

II – Opérations de base

- **Commentaire inséré**

Algorithmique :

- Sur une ligne :
commence par //
se termine avec la ligne
- Sur plusieurs lignes :
commence par /*
se termine par */

Langage C :

- Sur une ligne :
commence par //
se termine avec la ligne
- Sur plusieurs lignes :
commence par /*
se termine par */

Programmation 1 :

II – Opérations de base

- **Types de données élémentaires et leurs opérateurs**

Algorithmique :

nombre entiers naturels

nombre entiers relatifs

+ - * div mod

nombres réels

+ * - /

caractères

(concaténation : voir chaînes)

chaînes de caractères

div : division euclidienne

exp. $10 \text{ div } 3 = 3$

mod : reste de la division euclidienne

exp. $10 \text{ mod } 3 = 1$

Langage C :

unsigned int, unsigned long

int, long

+ - * / %

float, double

+ * - /

char

(opérations sur les entiers)

...

Programmation 1 :

II – Opérations de base

- **Variables et constantes : déclaration**

Algorithmique :

- pas besoin de déclaration préalable
- de type quelconque

Langage C :

- déclaration préalable indispensable (dans la zone de déclaration (3))
- variable typée

Exp. (zone 3)

// Déclaration seulement

```
int n; // n est un entier relatif
```

```
float x; // x est un réel
```

```
char c; // c est un caractère
```

```
const pi=3.1; // une constante (réelle)
```

// déclaration et initialisation

```
int k=0;
```

```
char d='@';
```

```
float y=1.13;
```

(1) directives de compilation

(2) définitions de types

(3) déclaration de variables

(4) déclaration de fonctions

(5) programme principal

Programmation 1 :

II – Opérations de base

- **Variables et constantes : affectation**

Algorithmique :

– symbole "flèche gauche" ←

Exp.

$n \leftarrow -5 ;$

$x \leftarrow -4.4 ;$

$x \leftarrow \pi x + n ;$

$c \leftarrow 'a' ;$

$c \leftarrow \text{minuscule}('A') ;$

$n \leftarrow \text{code}(c) ;$

$c \leftarrow \text{caractère}(65) ;$

⇒ Connaitre les principaux code ASCII

Langage C :

– symbole =

Exp. (dans la zone 5)

$n = -5 ;$

$x = -4,4 ;$

$x = \text{pi} * x + n ;$

$c = 'a' ;$

$c = 'A' + 32 ;$

$n = c ;$

$c = 65 ;$

N.B. En C, caractère \equiv code ASCII.

Pas de ré-affectation pour les constantes.

Programmation 1 :

II – Opérations de base

- Opérateurs propres au C

Algorithmique :

Exp.

```
n ← n+1 ; // incrémentation
```

```
n ← n-1 ; // décrémentation
```

(1) directives de compilation

(2) définitions de types

(3) déclaration de variables

(4) déclaration de fonctions

(5) programme principal

Langage C :

Incrémentation / décrémentation

Exp. (dans la zone 5)

```
n++ ;
```

```
n-- ;
```

Opérations sur les bits (voir TP) :

<< >> ~ & | ^

N.B. Ne pas utiliser +=, ..., ?, ...

Programmation 1 :

II – Opérations de base

- **Le type booléen (=logique) et ses opérateurs**

Algorithmique :

Deux valeurs logiques (booléennes) :

- Vrai (=V, =T, =True)
- Faux (=F, =False)

Des opérateurs logiques :

- non (unaire)
- et
- ou (inclusif)

Des comparateurs :

- =
- ≠ (noté parfois \diamond)
- ≤ < ≥ >

Langage C :

1 est Vrai
0 est Faux

!
&&
||

==
!=
<=

<

>=

>

Programmation 1 :

II – Opérations de base

- Le type chaîne de caractères

Algorithmique :

Délimiteur de constante : "

Affectation : ←

Opérateur : + (*concaténation*)

Comparateurs logiques

Exp.

```
S ← "bon"+"jour" ;
```

```
S ← S+" !" ;
```

```
Afficher S[1] ; // affiche 'o'
```

N.B. S[0] pour le premier caractère de la chaîne.

Langage C :

Délimiteur de constante : "

Affectation constante : =

Fonctions de <string.h>

Deux modes de déclaration :

```
char* S="bonjour" ;
```

```
char T[7]="bonsoir" ;
```

Deux modes de mémorisation :

– S : [b][o][n][j][o][u][r][\0]

– T (*taille 7*) : [b][o][n][s][o][i][r]

N.B. différence entre « chaîne à zéro terminal » et tableau de caractères

Programmation 1 :

II – Opérations de base

- **Les entrées**

Algorithmique : Saisir *nomvariable*

Langage C : fonction `scanf` + indicateur de formatage (*exemples*)

– Saisir un unsigned int :

```
scanf ("%u" , &n) ; // déclaration préalable : unsigned int n ;
```

– Saisir un int :

```
scanf ("%d" , &p) ; // déclaration préalable : int p ;
```

– Saisir un float :

```
scanf ("%f" , &x) ; // déclaration préalable : float x ;
```

– Saisir un caractère :

```
scanf ("%c" , &c) ; // déclaration préalable : char c ;
```

N.B. le nom de la variable est précédé de **&** (« *passage de la variable par adresse* »)

Programmation 1 :

II – Opérations de base

- **Les entrées**

Langage C : *pour les chaînes de caractères (exemple)*

– Saisir une chaîne de caractères :

```
scanf ("%s" , S) ; // déclaration préalable : char* S ou char S[...];
```

N.B. PAS de & pour une chaîne (*qui est déjà une adresse*)

Programmation 1 :

II – Opérations de base

- Les entrées



La saisie en C peut réserver des surprises agaçantes

Langage C : *formatage du flot de saisie*

- + Une saisie de l'utilisateur est validée par l'appui sur la touche "Entrée",
- + scanf recherche dans la saisie une valeur du type demandée qui ne contienne pas d'espaces,
- + ce qui n'est pas utilisé est conservé pour la prochaine demande de saisie.

Exemples : L'utilisateur saisira : "1000 bonjours"

```
// programme 1
```

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

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

```
→ S="1000"
```

```
→ T="bonjours"
```

```
// programme 2
```

```
scanf ("%d", n) ;
```

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

```
→ n=1000
```

```
→ S="bonjours"
```

```
// programme 3
```

```
scanf ("%d", n) ;
```

```
scanf ("%d", p) ;
```

```
→ n=1000
```

```
→ p= ????
```

Programmation 1 :

II – Opérations de base

- **Les sorties**

Algorithmique : Afficher ...

Langage C : fonction printf + indicateurs de formatage (*exemples*)

+ Affichage d'un texte constant :

```
printf("Ceci est un texte");
```

+ Affichage d'une variable :

```
printf("%d", n) ; // avec n de type int
```

+ Affichage combiné :

```
printf("il y a %d champignons", n) ;
```

```
printf("il y a %d cèpes et %d girolles", n, p) ;
```

+ Utilisation de caractères spéciaux :

```
printf("il y a %d champignons\n", n) ;
```

Programmation 1 :

II – Opérations de base

- **Les sorties**

Langage C : divers indicateurs de formatage numériques

Type	Format
int	%d
unsigned int	
- écriture décimale	%u
- écriture octale	%o
- écriture hexadécimale	%X
Float	
- écriture classique	%f
- écriture scientifique	%E

Type	Format
long	%ld
	...
double	%g ou %lf

constantes octales : commencent par **0**

constantes hexadécimales : commencent par **0x**

Programmation 1 :

II – Opérations de base

- **Les sorties**

Langage C : réserver un espace d’affichage (*exemples*)

– Réserver au moins 4 colonnes (ici pour un entier) :

+ en ajoutant des espaces à gauche : **%4d**

+ en ajoutant des zéros à gauche : **%04d**

+ en ajoutant des espaces à droite : **%-4d**

– Afficher un réel avec cinq chiffres dont trois après la virgule : **%5.3f**
(ajout d’espace à gauche si nécessaire pour avoir ≥ 5 colonnes)

– Afficher un réel avec deux chiffres après la virgule : **%.2f**

Tester avec le programme TVA



Programmation 1 :

II – Opérations de base

- *Pour maîtriser une fonctionnalité du C : faites des essais*

```
// Explorer le format d'affichage
#include <stdio.h>
int n=-123456;
float x=3.14159;
char* s="bonjour";

int main() {
    printf("3 n=%3d\n",n);
    printf("8 n=%8d\n",n);
    printf("2 x=%2f\n",x);
    printf("6.2 x=%6.2f\n",x);
    printf(".3 x=%.3f\n",x);
    printf(".0 x=%.0f\n",x);
    printf("10 x=%10f\n",x);
    printf("10 s=%10s\n",s);
}
```



**Utile : la trace de l'algorithme
ou du programme**



Résultat à l'écran :

```
3 n=-123456
8 n= -123456
2 x=3.141590
6.2 x= 3.14
.3 x=3.142
.0 x=3
10 x= 3.141590
10 s=  bonjour
```

Programmation 1 :

II – Opérations de base

- **La bibliothèque standard** (norme ISO)
 - Ensemble de fonctionnalités standard utilisables dans un programme.
 - Différents en-têtes thématiques indépendants.
 - Inclure les en-têtes utiles pour un programme avec `#include` (directives de compilation)
N.B. l'ordre des `#include` n'a pas d'importance

Exemples :

`<errno.h>` : numérotation standard des erreurs

`<limits.h>` : `INT_MIN`, `INT_MAX`, ...

`<math.h>` : fonctions mathématiques (`sin`, `cos`, `pow`, `sqrt`,...)

`<stdbool.h>` : type booléen (C99)

`<stdio.h>` : **`printf`**, **`scanf`**, `stdin`, `stdout`, `stderr`, `fprintf`, ...

`<stdlib.h>` : allocation mémoire (`malloc`, ...)

`<string.h>` : opération sur les `char*`

`<time.h>` : unités temporelles...

Programmation 1 :

II – Opérations de base

- **Dans la bibliothèque standard : <string.h>**
 - *strcpy* : copie d'une chaîne vers une autre (y compris le caractère \0)
 - *strcmp* : compare deux chaînes de caractères (retourne 0 si identiques)
 - *strlen* : donne la longueur d'une chaîne de caractères non vide



Fonctions de bas niveau utilisant le type `char*`

→ faire des essais pour bien maîtriser



Programmation 1 :

II – Opérations de base

- *Explorer les fonctions de string.h ...*

```
// Explorer les fonctions de string.h
#include <stdio.h>
#include <string.h>
char S1[15]={0};
char S2[15]={0};
char S3[15]={0};

int main() {
    printf("S1 ? ");
    scanf("%s",S1);
    printf("S2 ? ");
    scanf("%s",S2);
    strcpy(S3,S2);
    printf("strlen(S2)=%d\n",strlen(S2));
    printf("S3=%s\n",S3);
}
```

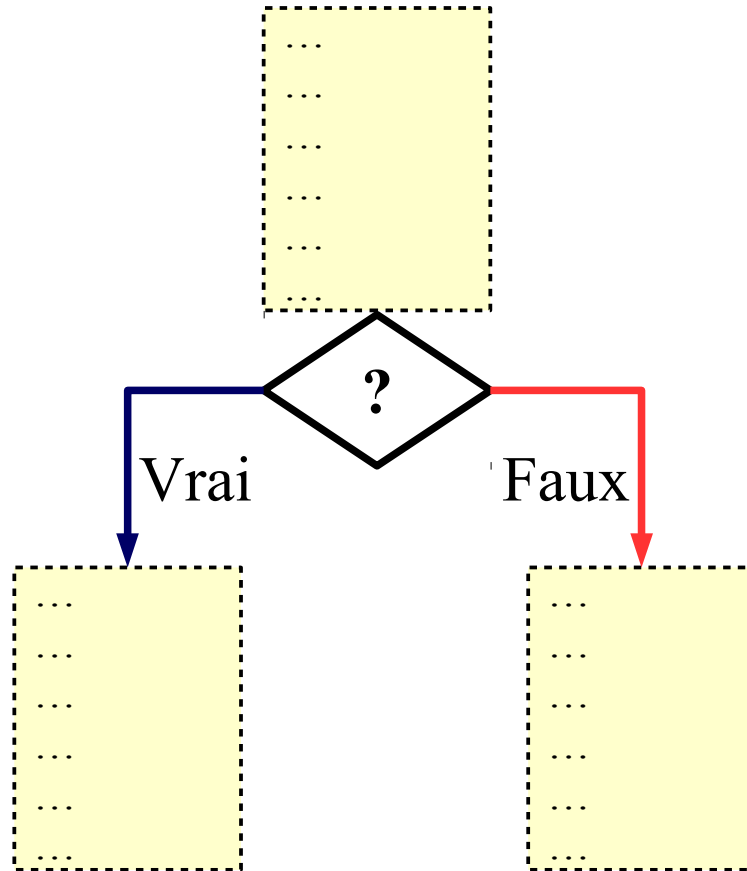
Résultat à l'écran :

```
S1 ? Bilbo
S2 ? Hobbit
strlen(S2)=6
S3=Hobbit
```

Programmation 1 :

III – Structures de contrôle

- Contrôler le déroulement du programme avec des tests



→ notion de *bloc d'instruction*

Programmation 1 :

III – Structures de contrôle

A – Les conditionnelles

A.1 – La conditionnelle simple

A.2 – La conditionnelle complète

A.3 – Imbriquer des conditionnelles Enchaîner des conditionnelles

A.4 – Le choix multiple

Programmation 1 :

III – Structures de contrôle – A.

A.1 – La conditionnelle simple

Algorithmique :

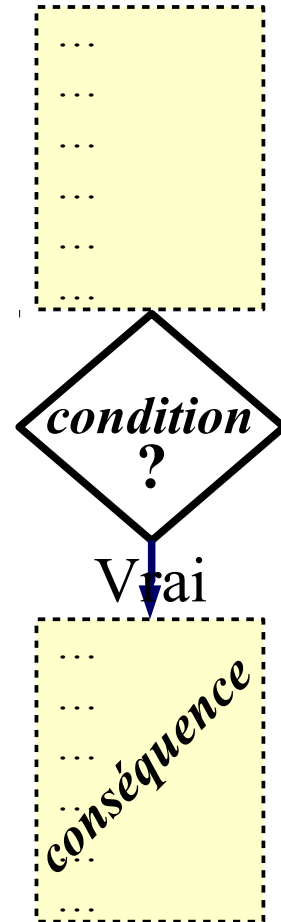
Si *condition*
alors *conséquence*

N.B. l'indentation délimite
les blocs

Langage C :

```
if (condition) {  
  conséquence  
}
```

N.B. les accolades { } sont les
délimiteurs de bloc



Programmation 1 :

III – Structures de contrôle – A.1.

- La conditionnelle simple : *exemple*

Algorithmique :

Si n est pair alors
 $n \leftarrow n \text{ div } 2;$

N.B. C'est l'indentation qui
 joue le rôle de délimiteur.

Langage C :

```
if (n%2==0) {  
  n=n/2;  
}
```

N.B. Avec une seule instruction
 les délimiteurs ne sont
 pas obligatoires :

```
if (n%2==0) n=n/2;
```

Programmation 1 :

III – Structures de contrôle – A

A.2 – La conditionnelle complète

Algorithmique :

Si *condition*
alors *conséquence*
sinon *alternative*

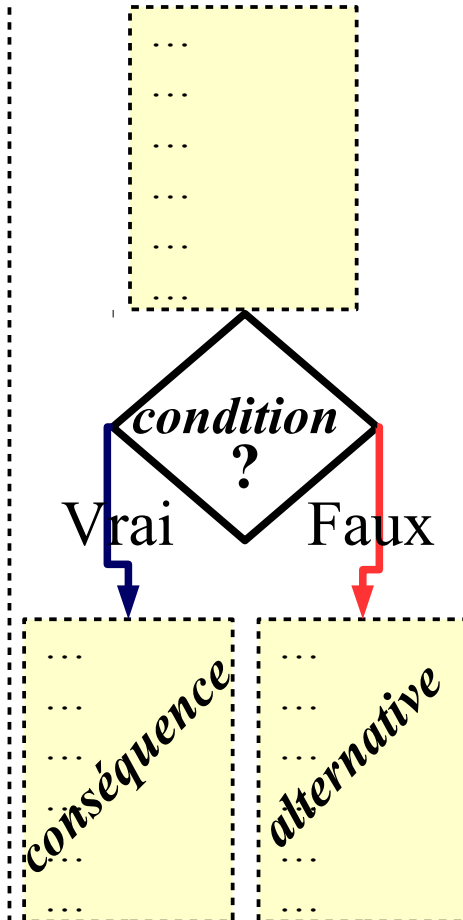
N.B. On peut aussi écrire :

Si *condition* **alors**
conséquence
sinon
alternative

Langage C :

```
if (condition) {  
    conséquence  
}  
else {  
    alternative  
}
```

N.B. Le positionnement des instructions sont pour le lecteur...



Programmation 1 :

III – Structures de contrôle – A.2.

- La conditionnelle complète : *exemples*

Algorithmique :

Si n est pair
 alors $n \leftarrow n \text{ div } 2$
 sinon $n \leftarrow 3n+1$;

Si *taux de TVA réduit*
 alors $tva \leftarrow 5$
 sinon $tva \leftarrow 20$;

Langage C :

```
if (n%2==0)
    n=n/2;
else n=3*n+1;
```

```
if (.....)
    tva=5;
else tva=20;
```

Tester avec le programme TVA



Programmation 1 :

III – Structures de contrôle – A.

A.3 – On peut imbriquer des conditionnelles

Algorithmique :

```
Si condition1  
  alors  
    Si condition2  
      alors conséquence1  
      sinon alternative1  
  sinon  
    Si condition3  
      alors conséquence2  
      sinon alternative2
```

Langage C :

```
if (condition1) {  
    if (condition2) { conséquence1 }  
    else { alternative1 }  
else {  
    if (condition3) { conséquence2 }  
    else { alternative2 }  
}
```

Programmation 1 :

III – Structures de contrôle – A.3.

- On peut enchaîner des conditionnelles

Algorithmique :

```
Si condition1  
  alors conséquence1  
sinon Si condition2  
  alors conséquence2  
...  
sinon alternative
```

N.B. Préférer un choix multiple
quand c'est possible

Langage C :

```
if (condition1)  
  { conséquence1 }  
else if (condition2)  
  { conséquence1 }  
...  
else { alternative }
```

Programmation 1 :

III – Structures de contrôle – A.

A.4 – Le choix multiple

Algorithmique :

Selon N faire

cas N1 : *bloc d'instructions*

cas N2 : *bloc d'instructions*

...

cas Ni : *bloc d'instructions*

autrement :

bloc d'instructions

N.B. Préférer un choix multiple
à un enchaînement de
conditionnelles

Langage C :

```
switch (N) {  
  case N1 : liste_d'instructions  
    break ;  
  case N2 : liste_d'instructions  
    break ;  
  ...  
  case Ni : liste_d'instructions  
    break ;  
  default : liste_d'instructions  
}
```

N.B. default est facultatif, break est impératif
N1,...,Ni sont des constantes *entières*

Programmation 1 :

III – Structures de contrôle – A.4.

- Le choix multiple : *exemple d'un menu*

Algorithmique :

Afficher *Menu*

Saisir Choix

Selon Choix **faire**

cas 0 : Quitter

cas 1 : *actions-1*

 ...

cas i : *actions-i*

// autrement : aucune action

Programmation 1 :

III – Structures de contrôle

B – Les itératives

B.1 – La boucle prédéterminée (Pour-faire)

B.2 – La boucle pré-contrôlée (Tantque-faire)

B.3 – La boucle post-contrôlée (Répéter-jusqu'à)

Programmation 1 :

III – Structures de contrôle – B.

B.1 – Itérative : la boucle prédéterminée (pour)

Algorithmique :

Pour i de 1 à n faire
bloc d'instructions

N.B. On utilisera souvent la variante :

Pour i de 0 à $n-1$ faire
bloc d'instructions

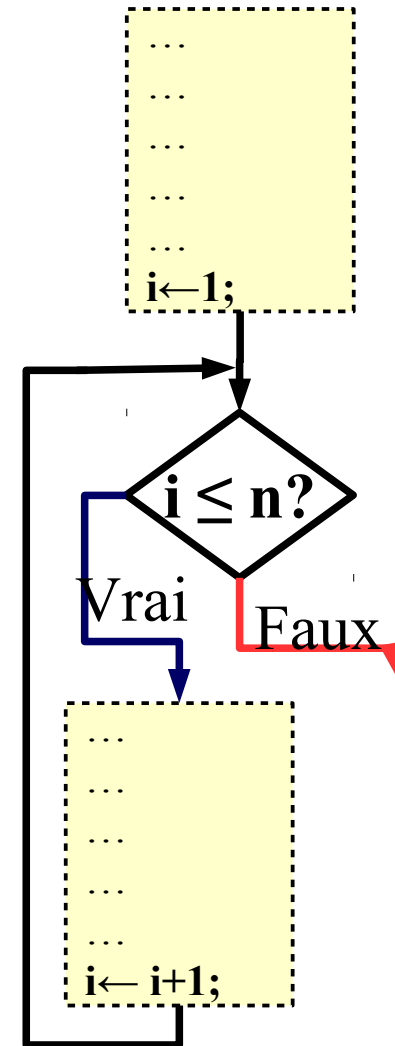
Langage C :

initialement ; continuer si ; après chaque tour

```
for (i=1; i<=n; i++) {  
    bloc d'instruction  
}
```

N.B. On utilisera souvent la variante :

```
for (i=0; i<n; i++) {  
    bloc d'instruction  
}
```



Programmation 1 :

III – Structures de contrôle – B.1.

- La boucle prédéterminée (pour) : *exemple*

Algorithmique :

Donnée : une chaîne S de n caractères.

Résultat : affichage des caractères de S pour chaque position.

Pour i de 0 à n-1 **faire**
Afficher "S[" , i, "]" = " , S[i].

N.B. Boucle très utilisée pour les parcours de tableau

Langage C :

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

```
char* S="bonjour";  
int i,n ;
```

```
int main() {  
    n=strlen(S) ;  
    for (i=0;i<n;i++)  
        printf("S[%d]=%c\n",i,S[i]) ;  
}
```

Trace :



Programmation 1 :

III – Structures de contrôle – B.

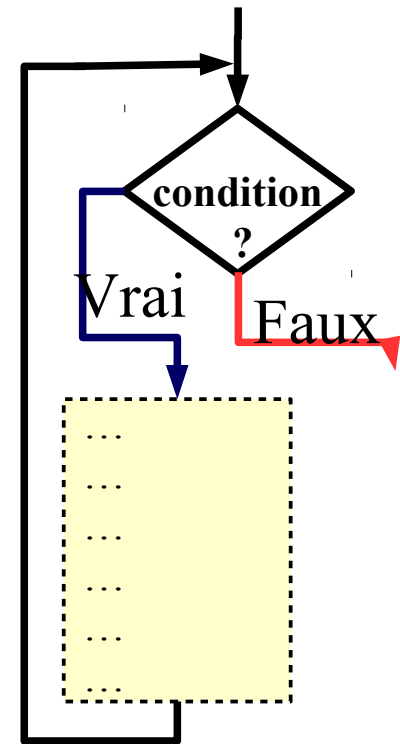
B.2 – Itérative : la boucle pré-contrôlée (avec condition de poursuite antérieure)

Algorithmique :

Tantque (*condition*) **faire**
bloc d'instructions

Langage C :

```
while (condition) {  
    bloc d'instruction  
}
```



Programmation 1 :

III – Structures de contrôle – B.2.

- La boucle pré-contrôlée (tantque) : *exemple*

Algorithmique :

Donnée : un entier naturel n .

Résultat : le nombre de chiffres de n .

$x \leftarrow 1$;

Saisir n ;

Tantque $n \geq 10$ faire

$x \leftarrow x + 1$;

$n \leftarrow n \text{ div } 10$;

Langage C :

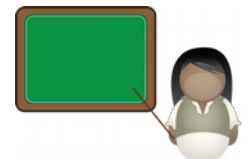
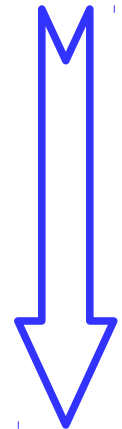
```
#include <stdio.h>
```

```
int x, n;
```

```
int main() {  
    x=1 ;  
    printf("n ? ");  
    scanf("%u", &n);  
    while (n >= 10) {  
        x++;  
        n = n / 10;  
    }  
}
```

Trace :

pour $n=8$, puis
pour $n=425$.



Programmation 1 :

III – Structures de contrôle – B.

B.3 – Itérative : la boucle post-contrôlée (avec condition [d'arrêt] postérieure)

Algorithmique :

Répéter

bloc d'instructions

jusqu'à *condition d'arrêt*

continuer si condition fausse

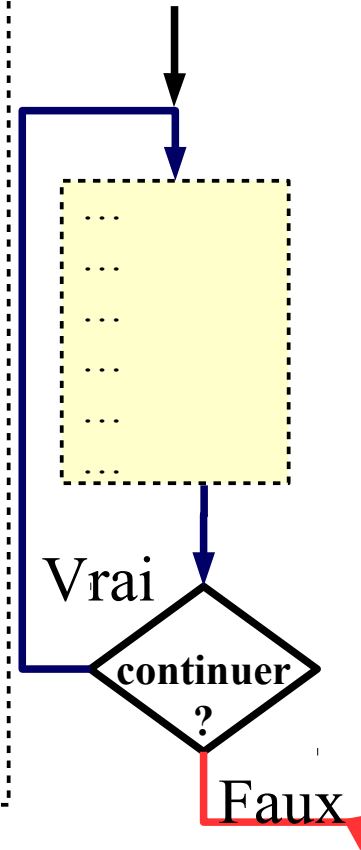
N.B. le bloc d'instruction est exécuté **au moins une fois**

Langage C :

```
do {  
    bloc d'instructions  
} while (condition de suite)
```

continuer si condition vraie

N.B. En C la condition finale est une condition de **poursuite**



Ne jamais utiliser *continue* ni *break* dans les boucles

Programmation 1 :

III – Structures de contrôle – B.3.

- La boucle post-contrôlée (répéter) : *exemple*

Algorithmique :

Répéter

Lire n ;

jusqu'à n>0 et n<10.

// il faut n dans [1..9]

Langage C :

```
do {  
    printf("n ? ");  
    scanf("%d",n);  
} while (n<=0 || n>=10);
```

Programmation 1 :

IV – Fonctions et procédures

- **Créer des "routines"**

- Nommer un bloc de programme (appelable)
- Lui transmettre (éventuellement) des paramètres
- Obtenir un résultat (procédure) et/ou un retour (fonction)
 - ← algorithme précisé
- Décomposer un problème un sous-problème
- Améliorer la lisibilité d'un programme (et donc faciliter son débogage).

Algorithmes précisés (*rappel*) :

Fonction AvecTVA

Entrée : le prix HT x.

Sortie : le prix TTC correspondant.

Variables d'usage : un réel positif tva.

tva ← 20 ;

Retourner $x*(1+tva/100)$.

Procédure AvecTVA

Donnée : le prix HT x.

Résultat : ttc est le prix TTC correspondant.

Variables d'usage : un réel positif tva.

tva ← 20 ;

ttc ← $x*(1+tva/100)$.

Programmation 1 :

IV – Fonctions et procédures

- Exemple de procédure : afficher un menu

Langage C :

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

void : en C une procédure est une fonction qui ne retourne rien

() : cette procédure n'a pas de paramètre

```
void afficherMenu() { // définition de la procédure  
    printf("0 - Quitter\n");  
    printf("1 - Action-1\n");  
    printf("2 - Action-2\n");  
    printf("3 - Action-3\n");  
}
```

```
int main() { // utilisation de la procédure  
    afficherMenu();  
    // ...  
}
```

Programmation 1 :

IV – Fonctions et procédures

- Exemple de fonction : choisir une valeur

Langage C :

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

```
int choix ;
```

*int : cette fonction retourne une valeur de type entier
() : cette fonction n'a pas de paramètre*

```
int choisir() { // définition de la fonction
```

```
int n;
```

```
do scanf("%d",&n) while (n<0 || n>3);
```

```
return n;
```

return n : cette fonction retourne la valeur de n

```
}
```

N.B. le programme principal est une fonction (sans paramètre ici)

```
int main() {
```

```
choix=choisir();
```

// utilisation de la fonction

```
// ...
```

```
}
```

Programmation 1 :

IV – Fonctions et procédures

- Exemple de structuration d'un programme :

Langage C :

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

```
int choix ;
```

```
void afficherMenu() {  
    printf("0 - Quitter\n");  
    printf("1 - Action-1\n");  
    printf("2 - Action-2\n");  
    printf("3 - Action-3\n");  
}
```

```
int choisir() {  
    int n;  
    do scanf("%d",&n) while (n<0 || n>3);  
    return n;  
}
```

```
// ...
```

```
void Action_1() {  
    /* ... */  
}
```

```
void Action_2() {  
    /* ... */  
}
```

```
void Action_3() {  
    /* ... */  
}
```

```
int main() {  
    do {  
        afficherMenu();  
        choix=choisir();  
        switch(choix) {  
            case 1 : Action_1();  
            case 2 : Action_2();  
            case 3 : Action_3();  
        }  
    } while (choix!=0);  
}
```

(1) directives de compilation

(2) définitions de types

(3) déclaration de variables

(4) déclaration de fonctions

(5) programme principal

Programmation 1 :

IV – Fonctions et procédures

- Exemple de fonction avec un paramètre :

Algorithmique :

Fonction AvecTVA

Entrée : le prix HT x.

Sortie : le prix TTC correspondant.

Variables d'usage : un réel positif tva.

tva ← 20 ;

Retourner $x * (1 + tva/100)$.

Langage C : *cette fonction retourne une valeur de type réel*

cette fonction a un paramètre réel

```
float avecTVA(float x) {  
    float tva=20;  
    return x*(1+tva/100);  
}
```

```
float ht, ttc;
```

```
int main() {  
    printf("prix HT ? ");  
    scanf("%f", ht);  
    ttc=avecTVA(ht);  
    printf("prix TTC=%.2f", ttc);  
}
```

Utilisation du retour de cette fonction

On entre la valeur de ht dans la fonction

Programmation 1 :

IV – Fonctions et procédures

- Exemple de fonction avec plusieurs paramètres :

Algorithmique :

Fonction TTC

Entrée : le prix HT x , un taux de tva t .

Sortie : le prix TTC correspondant.

Retourner $x*(1+t/100)$.

Langage C :

cette fonction retourne une valeur de type réel

cette fonction a deux paramètres réels

```
float TTC(float x, float t) {  
    return x*(1+t/100);  
}
```

```
float ht, ttc;
```

```
int main() {  
    printf("prix HT ? ");  
    scanf("%f", ht);  
    ttc=TTC(ht, 20);  
    printf("prix TTC=%.2f", ttc);  
}
```

// N.B. C distingue les majuscules et les minuscules

N.B. On peut écrire directement :

```
printf("prix TTC=%.2f", TTC(ht, 20));
```

Programmation 1 :

IV – Fonctions et procédures

- Exemple de procédure avec paramètres :

Algorithmique :

Procédure JA

Donnée : une chaîne S, un entier n.

Résultat : affichage de ...

Pour i de 0 à n faire

 Afficher i, " ans" ;

 Afficher S.

Langage C :

```
void JA(char S[], int n) {  
    int i;  
    for (i=0;i<=n;i++)  
        printf("%d ans\n",i);  
    printf("%s\n",S);  
}
```

```
char* T="joyeux anniv. !";
```

```
int main() {  
    JA(T,19);  
}
```

Programmation 1 :

IV – Fonctions et procédures

- Le passage des paramètres dans une fonction (ou procédure)

Langage C – déclaration :

```
TypeRetour NomFonction (type1 param1, ..., typek paramk) {  
  // ...  
  return valeurRetour ;  
}
```

Langage C – appel :

```
VariableRetour =NomFonction (Val1_type1, ..., Valk_typek) ;
```

A chaque appel :

- La *valeur*⁽¹⁾ de chaque *paramètre* est enregistré pour la fonction
- La *valeur de retour* est mise à disposition par l’instruction return
- Une seule occurrence de return est réalisée et met fin à la fonction

(1) pour les types de données élémentaires

Programmation 1 :

IV – Fonctions et procédures

- L'instruction de retour d'une fonction (return)

Langage C : *exemple avec if*

```
// (zone 4)
```

```
TypeRetour NomFonction (type1 param1,..., typek paramk) {  
  // ...  
  if (param1>param2) return valeurRetour1;  
  // else  
  return valeurRetour2;  
}
```

Avec des structures de contrôle, vérifier :

- Chaque situation donne une **unique** valeur de retour.

N.B. else est ici inutile car si param1>param2 la fonction s'arrête avec le return.

Programmation 1 :

IV – Fonctions et procédures

- L'instruction de retour d'une fonction

Langage C : *exemple avec while et if*

```
// (zone 4)
```

```
TypeRetour NomFonction (type1 param1, ..., typek paramk) {  
    ...  
    while (param1 > param2) {  
        ... // augmenter param2  
        if (param2 > param3) return valeurRetour1 ;  
    }  
    return valeurRetour2 ;  
}
```

Avec des structures de contrôle, vérifier :

- Chaque situation donne une **unique** valeur de retour.

Programmation 1 :

IV – Fonctions et procédures

- **La portée des variables : *exemple***

Dans la procédure JA :

- S et n sont des paramètres formels,
- i est une variable locale (« JA.i »),
- dans la fonction JA, pas d'accès à « Global.i »

Dans le programme principal :

- T, i, et n sont des variables globales (« Global.T », « Global.i », « Global.n »)
- hors de la fonction JA, pas d'accès à « JA.i » ni « JA.n »

Lors de l'appel JA(T,i) :

- JA.S ← Global.T (*paramètre effectif*)
- JA.n ← 5 (*paramètre effectif*)
- ... (JA.i est un compteur)

Langage C : *exemple*

```
void JA(char S[], int n) {
    int i;
    ...
}

int i, n ;
char* T="joyeux anniv. !";

int main() {
    i=5; JA(T, i);
    n=19; JA(T, n);
}
```

Programmation 1 :

IV – Fonctions et procédures

- **Nommage des variables : éviter les confusions**

Pour notre exemple :

Dans la procédure JA :

- noms explicites pour les paramètres

Dans le programme principal :

- utiliser des noms différents
(sauf pour les compteurs locaux h,i,j,...)

Langage C : *exemple*

```
void JA (char texte[],
         int nfois) {
    ...
}

int m,n ;
char* S="joyeux anniv. !";

int main() {
    m=5; JA(S,m);
    n=19; JA(S,n);
}
```

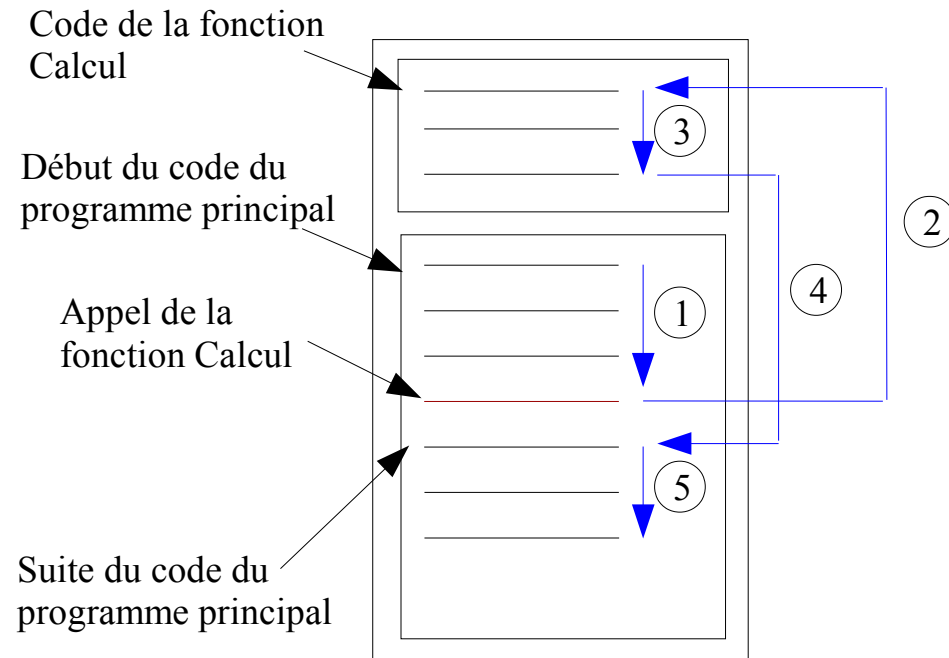
Programmation 1 :

IV – Fonctions et procédures

- **Exécution d'une fonction/procédure :**

© F. Jacquet

Lorsqu'une fonction est appelée à un point du programme, elle exécute son code. Une fois terminée, l'exécution du programme reprend au point où l'appel a été fait



Programmation 1 :

IV – Fonctions et procédures

- *Pour aller plus loin : exemple de fonction réursive*

Algorithmique :

Fonction Factorielle

Donnée : un entier n.

Résultat : n !

Si $n \leq 1$ alors retourner 1

sinon retourner $n * \text{Factorielle}(n-1)$,

Langage C :

```
long Facto(int n) {  
    if (n<=1) return 1;  
    return n*Facto(n-1);  
}
```

```
int main() {  
    printf("9!=%lu", Facto(9));  
}
```

Programmation 1 :

IV – Fonctions et procédures

- *Pour aller plus loin : le prototypage*

Langage C :

```
#include <stdio.h>
int choix;
// Prototypes=déclarations des fonctions
void afficherMenu();
int choisir();
void Action_1();
void Action_2();
void Action_3();

int main() {
    do {
        afficherMenu();
        choix=choisir();
        switch(choix) {
            case 1 : Action_1();
            case 2 : Action_2();
            case 3 : Action_3();
        }
    } while (choix!=0);
}
```

```
// Définitions séparées des déclarations
void afficherMenu() {
    printf("0 - Quitter\n");
    printf("1 - Action-1\n");
    printf("2 - Action-2\n");
    printf("3 - Action-3\n");
}
int choisir() {
    int n;
    do scanf("%d",&n) while (n<0 || n>3);
    return n;
}
void Action_1() {
    /* ... */
}
void Action_2() {
    /* ... */
}
void Action_3() {
    /* ... */
}
```

Zone 6

Programmation 1 :


V – Types tableaux

- **Tableau à une dimension**

- Grouper n (constante) variables de même type sous un nom unique,
- Accéder à chaque variable par un indice* compris entre 0 et n-1.
 - *permet un accès au tableau par une boucle*
- Permet de représenter des groupes de valeurs, vecteurs, matrices, etc.

Algorithmique :

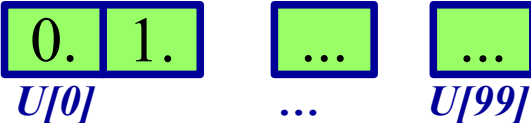
```
T[0] ← 1;
T[1] ← 1;
T[2] ← T[0] + T[1];
```



The diagram shows a horizontal array of five cells. The first cell contains '1', the second '1', the third '2', the fourth '?', and the fifth '?'. Above the first cell is the label 'T[0]', above the third is '...', and above the fifth is 'T[4]'. The cells are highlighted with a blue border and a light green fill.

Pour i de 0 à 99 faire

```
U[i] ← √i;
```



The diagram shows a horizontal array of four cells. The first cell contains '0.', the second '1.', the third '...', and the fourth '...'. Below the first cell is the label 'U[0]', below the third is '...', and below the fourth is 'U[99]'. The cells are highlighted with a blue border and a light green fill.

Langage C :

```
// Déclaration (zone 3) :
int T[5];
float U[100];

// Utilisation dans le main (zone 5) :
T[0]=1;
T[1]=1;
T[2]=T[0]+T[1];
for (i=0;i<100;i++)
    U[i]=sqrt(i);
```

* **Règle d'indilage du C** : T[i] est le i+1-ème élément

Programmation 1 :

V – Types tableaux

- **Gestion d'un tableau : danger !**

- La taille d'un tableau est fixe,
- Avant sa première affectation, une cellule peut contenir n'importe quoi,
- Il n'y a pas forcément de contrôle d'indice en C.
 - *accès aléatoire à la mémoire ou plantage (segmentation fault)*

Algorithmique :

Donnée : un tableau T de 100 entiers.

Pour i de 0 à 99 faire

U[i] ← √i;

// Erreur d'indice !

~~Afficher U[100].~~

Langage C :

// Déclaration : type nom[taille]

int T[5];

float U[100];

// Utilisation dans le main :

// T[0] n'a jamais reçu de valeur !

printf("%d", T[0]);

// U[100] n'existe pas !

~~**U[100]=1.1;**~~

Programmation 1 :

V – Types tableaux

- **Initialiser un tableau**

- L'initialisation peut être intégrée à la déclaration (cf variables simples),
- Les valeurs sont entre accolades et séparées par une virgule,
- Il existe un remplissage collectif par des zéros.

Langage C :

// Déclaration + initialisation :

// remplissage individuel

```
int T[5]={1,1,2,3,5};
```

// remplissage collectif (0 pour tous)

```
float U[100]={0};
```

// Pour les chaînes de caractères :

```
char Q[3]={'a','b','c'};
```

```
char R[6]="aeiouy";
```

```
char S[5]={0};
```



Programmation 1 :

V – Types tableaux

- **Prédéfinir la taille d'un tableau**

- Utiliser la macro-commande (= directive de compilation) **#define**,
- Donner un nom explicite à la taille du tableau.
→ *utiliser un nom pour les limites de boucles*

Langage C :

```
// Début du programme (zone 1) :
```

```
#define tailleT 5
```

```
#define tailleU 100
```

```
// Déclarations (zone 3) :
```

```
int i;
```

```
int T[tailleT];
```

```
float U[tailleU];
```

```
// Utilisation (dans le main, zone 5) :
```

```
for (i=0; i<tailleT; i++) T[i]=i;
```

```
for (i=0; i<tailleU; i++) U[i]=sqrt(i);
```

(1) directives de compilation

(2) définitions de types

(3) déclaration de variables

(4) déclaration de fonctions

(5) programme principal

Programmation 1 :

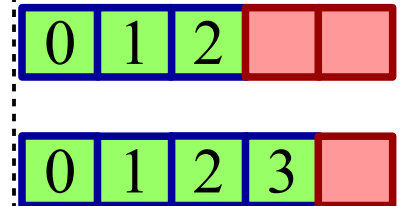
V – Types tableaux

- **Tableau de taille variable ?**

- La taille maximal n du tableau est une constante (taille réservée),
- On utilise les k premières cases, *avec impérativement $k \leq n$*
- On gère donc deux valeurs limites *taille maximale et taille utile*

```
Langage C : exemple // (zone 5)
// (zone 1)          Nutile=3;
#define Nmax 5      for (i=0;i<Nutile;i++)
// (zone 3)         T[i]=i;
int i;              if (Nutile<Nmax) {
int Nutile;         Nutile++;
int T[Nmax];        T[Nutile-1]=Nutile-1;
int Nutile;        }

```



Programmation 1 :

V – Types tableaux

- **Passer un tableau en paramètre d'une fonction**
 - On passe l'adresse du tableau (voir 2^e partie),
 - Un deuxième paramètre précise la taille

Langage C : exemple

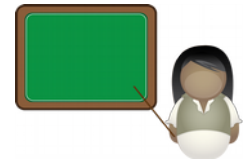
// (zone 3)

```
int maximum(int T[], int taille) {  
    int i;  
    int m=T[0];  
    for (i=1;i<taille;i++)  
        if (T[i]>m) m=T[i];  
    return m;  
}
```

Trace :

pour T :

1 -1 2 5 3 4



Programmation 1 :

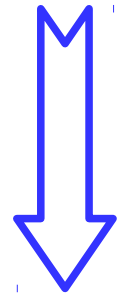
V – Types tableaux

- **Fonction modifiant un tableau (effet de bord, avec gcc sous Linux)**
 - La fonction change une/des valeurs du tableau passé en paramètre
 - Retrouvera-t-on ces modifications dans le programme principal ?

Langage C : *exemple*

```
int tableau[10]={0};
void modifier(int T[], int taille) {
    if (taille>0) T[taille-1]=5;
}
int main() {
    modifier(tableau,10);
    printf("tableau[9]=%d\n",tableau[9]);
}
```

Tester :



⇒ Mémoriser la réponse à cette question (pour ce système)

Programmation 1 :

Evaluation

TD : sur 20 pt

← un TD individuel final noté de 2 h

autorisé : une feuille A4 recto-verso écrite de votre main
interdit : tout autre document, tout outil numérique...

N.B. apporter votre carte d'étudiant.

TP : sur 20 pt

← évaluation par binôme des 6 TP de progression sur 3 pt

← un TP noté final par binôme sur 17 pt.

~~FIN~~ → A SUIVRE : Programmation 2 (consolidation des bases)