INSTRUCTIONS

1. Premier exemple
2. Structure d'un programme C
3. Compilation—exécution
4. Préprocesseur
5. Identificateurs
6. Variables
7. Fonctions
8. Fonctions E/S
9. Instructions
10. Constructeurs d'instructions
    - Conditionnelles
    - Itérations (boucles)
    - Blocs
11. Rupture de séquence
12. Branchement inconditionnel

Premier exemple

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

```
Bonjour !
```

```
#include <stdio.h>
main()
{
  char nom[30];
  scanf("%s", nom);
  printf("Bonjour %s !\n", nom);
}
```

à tous

```
Bonjour à tous !
```

Structure d'un programme C

`/* fichier som.c */`	*commentaire*
`#include <stdio.h>`	*macro-inclusion*
`#define PI 3.14159`	*macro-définition*
`float a,b;`	*définition de variables globales*
`float som(float, float);`	*déclaration de fonction*
`float dif();`	*déclaration de fonction*
`main(){`	*définition de "main"*
`    float c,d;`	*définition de variables locales*
`    a = PI; b = 1 ;`	*initialisations*
`    c = som(a,b);`	*appel de fonction*
`    d = dif();`	*appel d'une fonction de* `dif.c`
`    printf("%f %f",c,d);`	*fonction E/S*
`}`	
`float som(float x, float y){`	*définition de fonction*
`    return x+y;`	
`}`	

`/* fichier dif.c */`	*commentaire*
`extern float a,b;`	*déclaration de variables externes*
`float dif(){`	*définition de fonction*
`    return a-b;`	
`}`	

Compilation—exécution

1– LANCEMENT DE L'EXÉCUTION DU PROGRAMME

PRÉPROCESSEUR

COMPILATEUR

ASSEMBLEUR

ÉDITEUR DE LIENS

CHARGEUR

2– RÉSULTAT DE L'EXÉCUTION

```
4.14159 2.14159
```

La compilation des différentes unités peut se faire séparément.

Préprocesseur

- inclusion de fichier (`stdio.h, prog.c` respectivement) dans le fichier source

```
#include <stdio.h>
#include "prog.c"
```

- définition de constante par “substitution”

```
#define PI 3.14159
#define Cours "Cours de C"
```

  Toutes les occurrences de `PI` sont remplacées par `3.14159`, celles de `Cours` par `"Cours de C"`.

- permet la définition de macro-instructions (“macros”) avec paramètres

Identificateurs

Les éléments du langage (variables, fonctions, types, ...) sont manipulés par leur nom ou *identificateur*

1– SYNTAXE

Lettre ou '_' suivi de lettres, chiffres, ou '_' :

`[A-Za-z_][A-Za-z0-9_]*`

- les minuscules et majuscules sont différenciées
- la longueur est arbitraire
- le nombre de caractères significatifs dépend de la machine
  - au moins 31 pour les identificateurs internes
  - au moins 6 pour les identificateurs externes

2– MOTS-CLÉS

Ces identificateurs sont réservés :

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

Variables

- Opérateur d'adressage '`&`'

  Si `x` est un identificateur de variable : `&x` est l'adresse-mémoire de la variable (début de la zone mémoire associée à la variable `x`)

- Opérateur d'indirection '`*`'

  Si (la valeur de) `p` est une adresse-mémoire : `*p` est la variable qui est associée à cette adresse en mémoire

- `x` est équivalent à `*&x`

- `p` est équivalent à `&*p`

```
#include <stdio.h>
main(){
   char x, *p;
   p = &x;
   printf("Adresse de x = %p = %p = %p\n", p, &x, &*p);
   x = 'a';
   printf("Valeur de x = %c = %c = %c\n", x, *p, *&x);
}
```

```
Adresse de x = 001D17A3 = 001D17A3 = 001D17A3
Valeur de x = a = a = a
```

Fonctions

1– DÉFINITION D'UNE FONCTION

type_de_retour nom_fonction(*type1 var1, ...*){
  liste de déclarations
  liste d'instructions
}

```
int puissance(int x, int n){
    int p = 1, i;
    for (i = 0; i < n; i++)
        p = p*x;
    return p;
}
```

2– TRANSMISSION PAR VALEUR

- Les *paramètres* sont les variables figurant dans l'en-tête de la définition de la fonction.
- Les *arguments* sont les expressions qui sont substituées aux paramètres lors d'un appel.
- Lorsqu'une fonction est appelée, les valeurs des arguments servent à initialiser les paramètres de la fonction.
- La seule façon de faire modifier la valeur d'une variable par une fonction est de lui fournir l'adresse de cette variable.

```
main(){
    int x;
    printf("valeur de x ?  ");
    scanf("%d", &x);
    printf("valeur lue = %d\n", x);
}
```

RETENIR
il faut fournir au `scanf` l'*adresse*
de la variable que l'on veut lire

3– ENTRÉES/SORTIES

C ne possède pas d'instructions d'E/S. Il faut utiliser les fonctions de la bibliothèque standard.

- la bibliothèque standard est chargée automatiquement par l'éditeur de liens
- le fichier d'inclusion `<stdio.h>` contient les déclarations de fonctions liées aux E/S de la bibliothèque standard, ainsi que les macro-définitions de constantes comme `EOF`

Fonctions E/S

4 fonctions essentielles

1– SAISIE D'UN CARACTÈRE

```
int getchar(void)
/* saisit le caractère sur l'entrée
standard (stdin), EOF en fin de fichier */
```

2– AFFICHAGE D'UN CARACTÈRE

```
int putchar(char c)
/* écrit la valeur de c sur la sortie standard
(stdout) */
```

```
#include <stdio.h>

void main(void){
    /* recopie, caractère par caractère,
    l'entrée standard sur la sortie standard */
    int c;
    while ((c = getchar()) != EOF)
        putchar(c);
}
```

3– SAISIE FORMATÉE

```
int scanf(char *format, <liste d'adresses>)
```

Exemple

```
#include <stdio.h>

main(){
    char ca; short en; char ch[3]; float re;
    scanf("%c%hd%s%f", &ca, &en, ch, &re);
    ...
}
```

Résultat

entrée	format	adresse	type	mémoire
A	`%c`	`&ca`	`char`	41
257	`%hd`	`&en`	`short`	0101
"xy"	`%s`	`ch`	`char`	787900
4.2	`%f`	`&re`	`float`	?

- il faut passer les adresses à `scanf`
- formats et types doivent être compatibles

Formats de conversions

`%d`	entier forme décimale
`%i`	entier
`%o`	entier forme octale
`%u`	entier sans signe
`%x`	entier forme hexadécimale
`%c`	caractère
`%s`	chaîne de caractères
`%e,f,g`	flottant
`%p`	pointeur, adresse
`%n`	nombre de caractères lus
`%[...]`	chaîne de caractères dans ...
`%[^...]`	chaîne de caractères hors de ...

- les blancs et les tabulations sont ignorés dans le format
- les autres caractères (c'est-à-dire différents de `%`) doivent correspondre textuellement au flot d'entrée
- le caractère `*` supprime l'affectation, donc la lecture de l'argument
- `h, l, L` indiquent la largeur du format
- `scanf` retourne le nombre de conversions réalisées ou `EOF` si la fin du fichier est atteinte

```
while(!scanf("%[oOnN]", &car)
    printf("Repondre par Oui ou Non");
```

4– AFFICHAGE FORMATÉ

```
int printf(char *format, <liste d'expressions>)
```

Exemple

```
main(){
    int i; double x;
    scanf("%d%lf", &i, &x);
    printf("%3d %f x/i=%g\n", i, x, x/i);
}
```

Résultat

```
123 560.54
123 560.54000 x/i=4.55724
```

- il y a autant de spécifications de format que d'expressions
- le type de l'expression doit être compatible avec la spécification correspondante
- un caractère du format qui n'appartient pas à une spécification est recopié sur la sortie standard

Spécification de conversion

```
%[-][+][n1].[n2][h l L] <conversion>
```

- Conversions

`d,i,o,x,X`	entier
`u`	entier sans signe
`c`	caractère
`s`	chaîne de caractères
`f`	double `xxx.y1...yn`
`e,E`	double `x.y1...yn e±zz`
`p`	pointeur, adresse
`g,G`	double économique
`n`	nombre de caractères

- `n1` taille minimale du champ
- − cadrage à gauche
- + affichage du signe
- `h=short, l=long, L=long double`

Instructions

Expressions

;
expression;

Conditionnelles

`if`
`if else`
`switch case`

Itérations

`for`
`while`
`do while`

Bloc

{	liste de déclarations
	liste d'instructions
}	

Ruptures de séquence

`continue;`
`break;`
`return` *expr*;

Branchement

`goto`

Conditionnelles

`if`
`if else`
`switch case`

1– INSTRUCTIONS AVEC `if`

`if` (*expression*) *instruction*

`if` (*expression*) *instruction*
`else` *instruction*

```
#include <stdio.h>
main(){
   float x, y, resultat; char operateur;
   printf("Expression ? ");
   scanf("%f %c %f", &x, &operateur, &y);
   if(operateur == '+')
      resultat = x+y;
   else
      resultat = x-y;
   printf("%f %c %f = %f\n",
             x, operateur, y, resultat);
}
```

```
49 - 27
49 - 27 = 22
```

- Condition : construite avec
  - des expressions arithmétiques, ...
  - des comparateurs : `==` `!=` `<` `<=` `>` `>=`
  - des connecteurs : `&&` `||` `!`
- FAUX = 0, VRAI = tout entier non nul
- Dans un emboîtement de `else`, tout `else` est rattaché au `if` le plus proche

```
if (x%2)
    if (y%2)
        printf("x, y impairs \n");
    else
        printf("x impair, y pair \n");
```

```
if (x%2) {
    if (y%2)
        printf("x, y impairs \n");
} else
    printf("x pair \n");
```

2– AIGUILLAGE

```
switch (expression){
    case expression_constante_1:   instructions_1
    ...
    case expression_constante_n:   instructions_n
    default:  instructions
}
```

```
scanf("%d", &i);
switch(i){
case 1 : printf(" 1");
case 2 : printf(" 2");
case 3 : printf(" 3");
case 4 : printf(" 4");
default : printf(" coucou !\n");
}
```

2

```
 2 3 4 coucou !
```

3– UTILISATION de `break`

```
scanf("%d", &i);
switch(i){
case 1 : printf(" 1"); break;
case 2 : printf(" 2"); break;
case 3 : printf(" 3"); break;
case 4 : printf(" 4"); break;
default : printf(" coucou !\n");
}
```

2

```
 2
```

La valeur de *expression* est de type `int` et est comparée aux différentes constantes

- en cas d'égalité, l'instruction correspondante est exécutée, ainsi que toutes les instructions suivantes; possibilité de sortie avec `break`
- si l'expression diffère de toutes les constantes, les intructions par défaut sont exécutées. Si `default` n'existe pas, le programme sort du `switch`
- les valeurs des constantes sont différentes mais leur ordre est arbitraire
- `default` apparaît au plus une fois, n'importe où
- l'ensemble des `case` doit être délimité par des accolades
- plusieurs instructions peuvent se suivre derrière les "`case` *expression_constante*:" sans être entourées d'accolades

Itérations

`for`
`while`
`do while`

1– ITÉRATION `for`

`for(`*expr1*`;` *expr2*`;` *expr3*`)` *instruction*

```
...
for(i = 0; i < 10; i++)
   printf(" %d", i);
...
```

```
0 1 2 3 4 5 6 7 8 9
```

```
int puissance(int x, int n){
   int p, i;
   for (p = 1, i = 0; i < n; i++, p = p*x);
   return p;
}
```

2– ITÉRATION `while`

`while(`*expression*`)` *instruction*

Tester l'expression avant d'exécuter l'instruction (l'instruction peut donc ne pas être exécutée du tout)

```
...
while((c = getchar()) != EOF)
   ...
```

3– ITÉRATION `do while`

`do` *instruction* `while (`*expression*`);`

Tester l'expression après avoir exécuté l'instruction (l'instruction est donc exécutée au moins une fois)

```
...
do{ printf("Un entier entre 0 et 10 ? ");
    scanf("%d", &i); printf(" --> %d\n", i);
} while(i < 0 || i > 10);
...

...
do{ c = getchar();
} while (c == ' ');
...
```

Blocs

{	liste de déclarations
	liste d'instructions
}	

```
...
if(x > y) {
   int temp;
   temp = x;  x = y;  y = temp;
}
...
```

Rupture de séquence

`continue;`
`break;`
`return` *expr*;

- `continue`
  - concerne la boucle la plus proche
  - fait passer au pas suivant
- `break`
  - concerne la boucle ou l'aiguillage le plus proche
  - fait sortir de la boucle ou de l'aiguillage
- `return`
  - concerne les fonctions

```
...
while(1){
   scanf("%d", &i);
   if(i == 0) break;
   else if(i < 0) continue;
   else somme += i;
}
printf("Somme des nombres positifs = %d\n", somme);
...
```

Branchement inconditionnel

```
goto identificateur;
...
identificateur :   instruction
```

- *identificateur* est une étiquette
- la portée de l'étiquette est limitée à la fonction où elle est définie
- on ne peut faire de `goto` d'une fonction à une autre

NOTE
POUR UN USAGE AVERTI UNIQUEMENT !

EXERCICES

1. **ASCII** : écrire un programme qui affiche la table des caractères et leur code ASCII.

2. **WC** : écrire un programme qui lit son entrée au clavier et calcule :
   - le nombre de caractères lus
   - le nombre de lignes lues
   - le nombre de mots lus (utiliser `isalpha(char)` de <`ctype.h`>)

3. **Calculette** : écrire un programme de calculette, capable d'évaluer les quatre opérations `+, -, *, /`. Prévoir le cas d'une division par 0.

ASCII

```
#include <stdio.h>

main(){
   unsigned i, j;

   for(i=0; i<32; i++){
      for(j=1; j<8; j++)
         printf(" -%o-%3d-%c- ",
                i+j*32, i+j*32, i+j*32);
      printf("\n");
   }
}
```

```
-40- 32- -  -100- 64-@-  -140- 96-`-
-41- 33-!-  -101- 65-A-  -141- 97-a-
-42- 34-"-  -102- 66-B-  -142- 98-b-
-43- 35-#-  -103- 67-C-  -143- 99-c-
-44- 36-$-  -104- 68-D-  -144-100-d-
-45- 37-%-  -105- 69-E-  -145-101-e-
-46- 38-&-  -106- 70-F-  -146-102-f-
-47- 39-'-  -107- 71-G-  -147-103-g-
-50- 40-(-  -110- 72-H-  -150-104-h-
-51- 41-)-  -111- 73-I-  -151-105-i-
-52- 42-*-  -112- 74-J-  -152-106-j-
-53- 43-+-  -113- 75-K-  -153-107-k-
-54- 44-,-  -114- 76-L-  -154-108-l-
-55- 45---  -115- 77-M-  -155-109-m-
-56- 46-.-  -116- 78-N-  -156-110-n-
-57- 47-/-  -117- 79-O-  -157-111-o-
-60- 48-0-  -120- 80-P-  -160-112-p-
-61- 49-1-  -121- 81-Q-  -161-113-q-
-62- 50-2-  -122- 82-R-  -162-114-r-
-63- 51-3-  -123- 83-S-  -163-115-s-
-64- 52-4-  -124- 84-T-  -164-116-t-
-65- 53-5-  -125- 85-U-  -165-117-u-
-66- 54-6-  -126- 86-V-  -166-118-v-
-67- 55-7-  -127- 87-W-  -167-119-w-
-70- 56-8-  -130- 88-X-  -170-120-x-
-71- 57-9-  -131- 89-Y-  -171-121-y-
-72- 58-:-  -132- 90-Z-  -172-122-z-
-73- 59-;-  -133- 91-[-  -173-123-{-
-74- 60-<-  -134- 92-\-  -174-124-|-
-75- 61-=-  -135- 93-]-  -175-125-}-
-76- 62->-  -136- 94-^-  -176-126-~-
```

```
-77- 63-?-  -137- 95-_-  -177-127--
```

WC

```
#include <stdio.h>
#include <ctype.h>
main(){
   int nc = 0, nl = 0, nm = 0, en_mot = 0; char c;
   while((c=getchar()) != EOF){
      if(c == '\n'){
         nl++; en_mot = 0;
      }else{
         nc++;
         if (isalpha(c) && en_mot == 0){
            nm++; en_mot = 1;
         }else if(!isalpha(c)) en_mot = 0;
      }
   }
   printf("nc = %d, nm = %d, nl = %d\n",
           nc, nm, nl);
}
```

Ce programme s'appelle "wc"
pour word counter.
<EOF>
`nc = 45, nm = 8, nl = 2`

Calculette

```
#include <stdio.h>
main(){
   float x, y, resultat; char operateur, c;
   int ERROR = 0;
   printf("Calculette\n");
   do{
      printf("Expression ? ");
      scanf("%f %c %f %*s", &x, &operateur, &y);
      switch(operateur){
        case '+': resultat = x+y; break;
        case '-': resultat = x-y; break;
        case '*': resultat = x*y; break;
        case '/': if(y != 0) resultat = x/y;
                  else { printf("DIV par 0\n"); ERROR = 1;}
                  break;
        default :
               { printf("Oprateur inconnu\n"); ERROR = 1;}
      }
      if(!ERROR)
         printf("%f %c %f = %f\n", x,operateur,y,resultat);
      do{ printf("Encore (o/n)? "); c = getchar();
      } while(c != 'o' && c != 'n');
   } while (c != 'n');
   printf("Au revoir !\n");
}
```