



Université Lille Nord de France

Pôle de Recherche
et d'Enseignement Supérieur

Université
de Valenciennes
et du Hainaut-Cambrésis

ALGORITHMIQUE ET PROGRAMMATION C NIVEAU 2

UVHC – ISTV – Licence 2

Rabie Ben Atitallah

rabie.benatitallah@univ-valenciennes.fr

<http://www.lifl.fr/~benatita/pages/Teaching>

1

PROGRAMME

Chap 1 Le langage C

Chap 2 Programmation et Algorithmique

Chap 3 Pointeurs et structures

Chap 4 Récursivité

Chap 5 Fichiers et algorithmes

Chap 6 Notions de complexité des algorithmes

Chap 7 Algorithmique non numérique

Chap 8 Algorithmique numérique

CHAP I LE LANGAGE C

C ... UNIX ... Editer, Compiler, Activer

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

```
main ( )
```

```
{ int c;
```

```
  c = 5;
```

```
  while ( c > 0 )
```

```
    { printf (" c = %d", c);
```

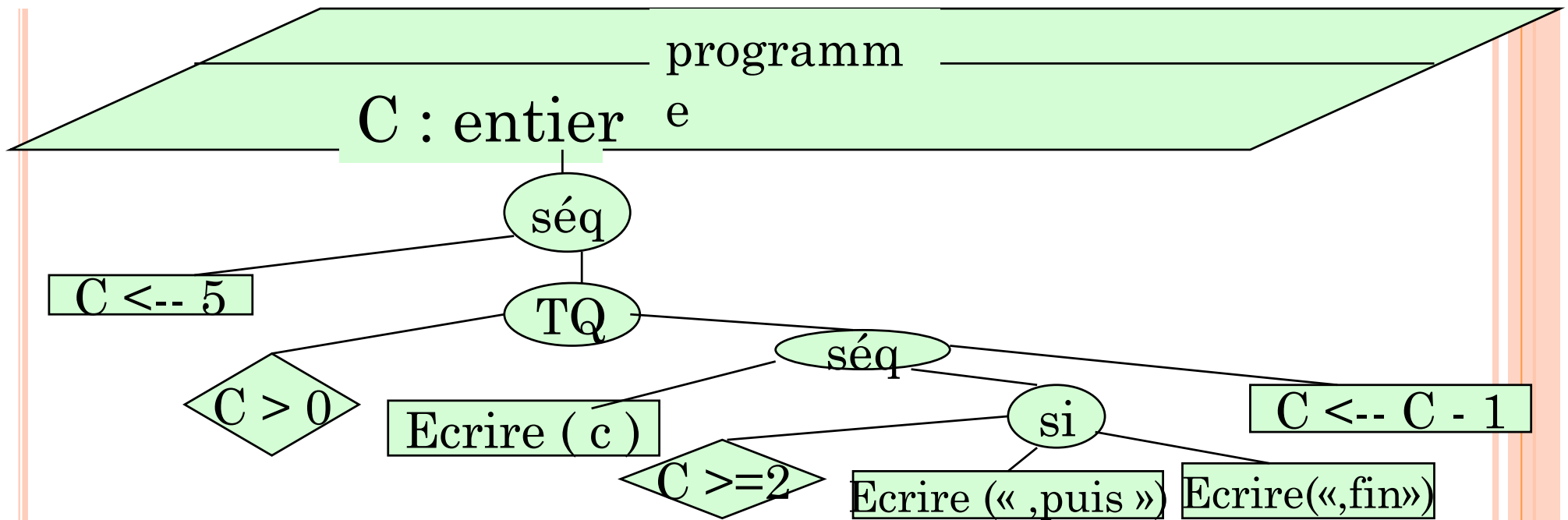
```
      if ( c >= 2 ) printf (", puis ");
```

```
        else printf (", fin \n");
```

```
      c = c - 1;
```

```
    }
```

```
}
```



main ()

{ int c = 5; while (c)

{ printf (" c = %d", c);

if (c >= 2) printf (" , puis ");

else printf (" , fin \n");

c = c - 1;

}

}

Chaque instruction se termine par un **point-virgule**.

Liberté de l'**indentation**.

Le programme est écrit en **minuscule**.

```
main ()      /* entête fonction main sans argument */  
{           /* accolade bloc de fonction */  
  float rayon, surface; /* déclaration des variables */  
  printf (" rayon = ? ");      /* affichage */  
  scanf ("%f", & rayon); /* lecture de la réponse */  
  surface = 3.1459 * rayon * rayon; /* affectation */  
  printf (" surface = %f\n", surface); /* affichage */  
}
```

I) SOMMAIRE SUR LE LANGAGE C

1) Jeu de caractères

- 26 lettres, chiffres, caractères spéciaux: - { } < > *...
- séquences d'échappement: (retour arrière '\b' ,
à la ligne '\n')

2) Mots clé en minuscule

auto extern sizeof break float static
case for struct char goto
switch const
if typedef continue int union
default
long unsigned do register void

3) DÉCLARATION DE VARIABLES

< type > < liste d'identificateurs de variables > ;

< type > < identificateur = constante > ;

int : entier

unsigned : entier non signé

char : caractère

float : réel

short : entier court **double**:réel double précision

long : entier long

signed entier signé

/*Déclarations =*/ ***signed long int a, x; signed long a,x;
long a, x;***

int i, j =0 ;

float a = 1.5, b , c= 10;

char fin = '.';

double x, y = a;

4) DÉCLARATIONS DE FONCTIONS

forme primitive ou ancienne et encore valable :

< type > nom_de_fonction (arg1, arg2, ...)

< type > arg1; < type > arg2;

*{/*déclaration de var locales; instr de la fonction*/*

*return ; /*valeur à retourner */}*

forme nouvelle et normalisée , valable sur tous les compilateurs:

<type> nom_fonction (<type> arg1, <type> arg2, ...)

*{ /*déclaration de var locales;*

*instructions */*

return ; / valeur */*

5) LES INSTRUCTIONS E/S AVEC FORMAT

printf

scanf

printf (controle, arg1, arg2, ...);

scanf (controle, arg1, arg2, ...)

idem

int i = 2, j = 25; float x = 1.5, y ;

double z; char a, b;

**printf ("entrer la valeur %d suivie du
caractère", i);**

scanf ("%f %c", &y, &a) ;

**printf ("j= %d et x+y= %f caract %c", j, x+y,
a);**

6) LES OPÉRATEURS

opérateurs arithmétiques, unaires, relationnels et logiques, et de manipulations de digits binaires. Enfin les opérateurs d'affectation: simples, multiples, composées ou élargies, incrémentation :

affectation $c = 3;$ $c = c-1;$

affectation composée $a += b;$ $/* a = a+b */$

$a -= c;$ $a *= b;$ $a /= c;$

incrémentations $c--;$ $/* c = c-1 */$ $a++;$ $++a;$

assignation multiple $a = b = c = 1;$

opérateur à 3 opérandes $d = a ? b : c;$

$/* si a \neq 0 alors d := b \quad sinon d := c */$

7) INSTRUCTIONS

if (exp. log.) instruction; ou {bloc} else instr; ou {

if (b>0) c++;

if (a > 0) --b; else {c = c+1; b++; a -= 1 ;}

while (expr. log.) instr; ou {bloc } /* tant que cond
faire...*/

while (k > 0) { printf ("%d \n", k); k--;}

do instr; ou{bloc} while (expr. log); /*faire ...tant que

cond*/(init; condition; incrémentation) instruction ; ou

{ bloc }*do { printf ("%d \n", k); k--; } while (k > 0) ;*

for (i = 1; i <= 10; i = i+1) c++;

for (i = 1; i < 10; i++, c++);/ 3 instr = */*

i = 1 ; for (; i < 10; c++, i++);

for (n = 0; n < 15; n = n + 0.5);

II TYPES, CONSTANTES, VARIABLES

1 Les types de base

ENTIER : { *[signed / unsigned]* } { *[short / long]* } *int*

REEL : *float* (4 octets) *double* (8 octets)

CARACTERES : *char* *unsigned char* (0 à 255) en fait
sous ensemble des entiers

VOID (= vide) type des sous-prog sans résultat proc

2 Type des valeurs booléennes

Tout entier = valeur booléenne. Ainsi toute valeur numérique entière peut être considérée comme une valeur booléenne ou logique

FAUX est codé par l'entier 0, **VRAI** par 1 ou un entier
de 0

if (b) c++; while (k) { printf ("%d \n", k);

3 Les types énumérés

Codage des valeurs réalisé par des identif (= const)

```
enum [ identif de type ] { ident de const [ = init ] [, ident-cst[ =  
init ] .}
```

```
enum jour {lun, mar, merc, jeudi, vend, sam, dim};
```

```
enum boolean { FALSE, TRUE };
```

```
enum jour j ;
```

4 Les définitions de type

```
typedef type déclarateur / identificateur
```

```
typedef int Entier ;
```

```
typedef enum jour Type_jour ;
```

```
typedef enum { FAUX, VRAI }
```

```
Booleen ;
```

5 Les constantes

Identif de Constantes littérales (le préprocesseur)

```
#define MAXC 20
```

```
#define MAXT 30
```

```
#define PLACE (MAXC*MAXT)
```

6 Déclaration de variables

Identif de variable, avec type et classe d'allocation

```
[ classe ] type déclarateur-unit [ ,decl ]...;
```

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

```
enum jour j;
```

```
Booleen trouve, pas_la = vrai;
```

EXEMPLE DE DÉCLARATIONS

```
#include <stdio.h>  
#define MAX 100  
#define MESSAGE printf(" Le calcul avance\n ")  
typedef int Entier;  
typedef enum {FAUX, VRAI} Booleen;  
main (  
{ Entier i, j = MAX;  
  Booleen trouve = FAUX;  
  MESSAGE; /* ..... */ }
```

III LES EXPRESSIONS

1 Expression arithmétique

opérateurs sur entier ou caractères(+, -, *, /, %)

opérateurs sur réels (+, -, *, /)

2 Expressions logiques

opérateurs sur opérandes entiers ou booléens: !, &&, ||

l'évaluation n'est complète que si indispensable.

3 Expressions relationnelles

opérateurs: == != < <= > >=

4 Expressions de manipulation de bits

<< >> & | ^

~

Décal G Déc Dr ET binaire OU inclusif OU excl

Complément

5 Opérateurs d'affectation

identif = expression; identif **op** = expression où
op est : + - * / % >> << & ^ |

incrémentation et décrémentation:

i++ on prend i, on l'utilise puis on incrémente i

++i on prend i, on incrémente i puis on l'utilise

6 Les conversions implicites ou automatiques

Si les types des opérandes sont différents, "le plus fort" l'emporte :

7 Les conversions explicites (forceur)

Un forceur est une expression qui transforme une autre expression en une valeur d'un type donné.

10 PRIORITÉ DES OPÉRATEURS

() [] -> . ASSOCIATIVITE
! ~ ++ -- - (-type) * & sizeof
* / %
+ -
>> <<
< > <= >=
== !=
& (2 opérandes)
&
&&
||
? :
= += -=
,

IV LES ENTRÉES ET SORTIES

E/S en C par une bibliothèque `stdio.h` de fonctions
`#include<stdio.h>`

1 Lecture-Ecriture d'un caractère

`getchar ()` : délivre le caractère saisi au clavier

`putchar (x)`: ajoute le caractère `x` sur la sortie `stdout`

`char c; c = getchar (); putchar ('\n');`

2 Lecture-Ecriture d'une ligne

`gets()`: délivre la ligne tapée sur l'entrée standard
`stdin`

`puts (ch)`: affiche la chaîne `ch` sur la sortie `stdout` .

3 E/S avec format

`printf (format, e1, e2,...);scanf (format, e1, e2,...);`

4 Mise en tampon des entrées et sorties

V LES INSTRUCTIONS

Instruction d'affectation=toute expression terminée par ";"

Instruction composée ou bloc ou séquence { }

Instruction

TQ

```
{ int c;  
c = getchar ();  
while ( c != EOF )  
    { putchar (c);  
      c = getchar ();  
    }  
}
```

```
{ int c;  
  while ( ( c = getchar  
    ( ) ) != EOF )  
    { putchar (c);  
      }  
}
```

Instruction répéter (faire tant que)

```
{ int reponse;  
printf ("votre choix");  
do { printf (" (0-4) ?");  
scanf ("%d", &reponse);  
}  
while (reponse<0 || reponse>4);  
}
```

Instruction SI

```
{ int c, nbligne = 0;  
while ( ( c = getchar() ) != EOF )  
if ( c == ' \n ' ) nbligne+ = 1;  
printf (" Il y avait %d lignes \n ",  
nbligne); }
```

Instruction CAS ou SELON

```
main ( )
{ int c, nbvoy=0, nbblanc=0, nbautre=0;
  while ( ( c = getchar() ) != EOF)
    switch (c)
      { case 'a':case 'e':case 'o':case 'i':case 'u':case
        'y':
          nbvoy += 1; break;
        case ' ': case '\t' :nbblanc += 1; break;
        default nbautre += 1;
      }
  printf ("Il y avait %d voyelles,%d blanc,%d autre\n",
          nbvoy, nbblanc, nbautre);
}
```

Instruction POUR

```
{ int a, b, p=1, i;  
  scanf ("%d %d", &a,  
    &b);  
  for ( i = 0 ; i < b ; i += 1)  
    p = p*a;  
}
```

```
{ int a, b, i, p;  
  scanf ("%d %d", &a,  
    &b );  
  for ( p = 1, i = 0 ; i < b ;  
    p = p*a, i +=  
1);  
}
```

*/*calcul de n! */*

```
{ int f, n;  
  scanf ("%d", &n);  
  for (f =1; n >0; f*=  
n--);  
  printf (" %d ", f);  
}
```

```
{ int a, b, p;  
  scanf ("%d %d", &a,  
    &b );  
  for ( p = 1; b -- > 0; p* =  
a);  
}
```

VI LES TABLEAUX

1 **Déclaration** *déclaration [expression constante]*

*int t [10]; char lettres[2*26];*

*float mat[N][N]; char lesmots[NBMOTS]
[LONGMAX];*

typedef float Matrice [N] [N]; Matrice a ;

identificateur de tableau = adresse du 1^{er} élément du tableau

pas d'affectation de tableau

opérateur d'indexation : numérotation commence à 0 :

{ int t[5], i;

for (i = 0 ; i < 5 ; i+ = 1) scanf ("%d ", &t[i]);

for (i = 0 ; i < 5 ; i+ +) printf (" %5d ", t[i]);

}

&t[0] == t;

2 LES CHAÎNES

`char c[n]` définit une chaîne de caractères ;
ni initialisations ni affectation de chaîne.

Toute chaîne doit se terminer par un marqueur de fin de chaîne (caractère de code nul: ASCII 0)

"chaîne" -> *"chaîne\0 "*

printf ("ABC"); -> *'A' 'B' 'C' '\0'*

char x[4]= { 'a', 'b', 'c', '\0' }; <=> char x[4]="abc";

des fonctions

strcpy (*ch1*, *ch*) pour copier une chaîne,

strcmp(*ch1*, *ch2*) pour comparer 2 chaînes,...

et aussi *scanf ("%s", ch); printf ("%s", ch);*

VII LES SOUS-PROGRAMMES

Fonctions avec un résultat et fonctions sans résultat.

1 Déclaration de fonction

- * Une fonction se déclare avant de s'utiliser.
- * Une fonction ne s'emboîte pas dans une autre.
- * Deux syntaxes existent : la nouvelle et l'ancienne :

ancienne: *[classe][type] décl ([liste param.]) [decl par.]
bloc*

nouvelle : *[classe][type] décl ([declarer param.]) bloc*

*int somme (a, b) int a, b; { return (a + b); }
int somme (int a, int b); { return (a + b); }*

2 Visibilité des objets

les **paramètres** formels, les variables locales,
les **variables globales** /* fortement déconseillé !!!
*/

3 Paramètres transmis par valeur !

```
int max (int a,int b)      void lignede (int nb, char  
                           car)  
{ return (a > b)? a : b ;      { for ( ; nb>0; --nb )  
                               putchar (car); }  
}
```

4 Paramètre résultat. Tableau. Fonction

Passer un paramètre par adresse = passer par les pointeurs, sauf pour tableau car alors le nom = adresse de la première position du tableau)

```
int strlen( char s[ ] )  
{ int i = 0 ;  
  while (s[ i ] != '\0' ) i  
  ++;  
  return i;  
}
```

```
int strlen2 ( char s[ ] )  
{ int i = 0;  
  for ( i=0; s [i] != '\0'; i+  
  +);  
  return i;  
}
```

```

void faux_permut (int a,
int b)
{ int c; c = a; a = b; b =
c; }
void perm2 ( int t [ ] )
{ int c; c = t [0]; t [0] = t [1];
t [1] = c; }

```

```

void saisie ( float t [ ], int n )
{ int i; for ( i = 0 ; i < n ; i++ )
scanf ( " %f " , &t[i] );
}

```

```

int saisie2 ( float t [ ] )
{ int i , n ;
printf ( "entrer nombre de
valeurs" );
scanf ( « %d » , &n ) ;
for ( i = 0 ; i < n ; i++ )
scanf ( " %f " , &t[i] );
}

```

```

int nboc(char c, char
ch [])
{ int nb = 0, i = 0 ;
while ( ch [i] != '\0' )
if ( ch [i] == c) nb +
+ ;
return nb ;
}

```

```

void saisie3 (int t[ ],
int n)
{
int i = 0;
while ( i < n )
scanf ( "%d" , &t[i+
+ ] );
}

```

```

#include <stdio.h>
#define MAX 1000
typedef float Tab
    [ MAX ];
void saisie3 (Tab t , int
    n)
    { int i = 0;
      while ( i < n )
          scanf ("%f ", &t[i++] );
    }
void aff (Tab t, int n)
    { while ( --n >=0 )
          printf ("%f ", t
    [n]);
      printf(" \n ");
    }
float som( Tab t, int n)
    { float s=0;
      while ( --n) s += t[n];
      return s;    }

```

```

main ( )
{
    int nb ;
    Tab z ;
    printf ("entrer nb ");
    scanf ("%d", &nb);
    saisie3 (z, nb) ;
    aff ( z, nb );
    printf ("somme=%f
\n",          som (z,nb) ) ;
}

```