

## Partie 4 : Constructions fréquentes

- ▶ Premiers types de base
- ▶ Les types `int`, `double`, `char`
- ▶ Exemples
- ▶ Le type `bool`
- ▶ Quelques opérateurs arithmétiques
- ▶ Quelques opérateurs logiques
- ▶ Affectation
- ▶ Tests : `if` et `if..else`
- ▶ Boucle `for`
- ▶ Boucle `while`

AlgoProg 2015-16 35

## Premiers types de base

En C toute variable a un type.

Le **type** détermine :

- ▶ l'ensemble des **valeurs** qu'une variable peut prendre,
- ▶ la façon dont ces valeurs **sont stockées** en mémoire, (souvent dépendant de l'implémentation).
- ▶ les **opérations** applicables à la variable.

Quelques types prédéfinis de base :

- ▶ `int` : entiers
- ▶ `double` : flottants (nombres avec partie décimale)
- ▶ `char` : caractères
- ▶ `bool` : booléens

Il y a d'autres types entiers et d'autres types flottants

AlgoProg 2015-16 36

## Les types : `int`, `double`, `char`

- ▶ **`int`** :
  - La taille de l'intervalle d'entiers représenté dépend de la machine et tend à augmenter ( un nombre de type `int` est souvent codé, actuellement, sur 4 octets).
  - ➡ **ne pas faire d'hypothèse sur la taille d'un `int`.**
- ▶ **`double`** : pour représenter les nombres flottants en double précision
  - la précision dépend de la place utilisée pour coder la partie décimale
- ▶ **`char`** : permet de représenter **le jeu de caractères de base.**
  - Constantes entre quotes : `'X'`. **Ne pas confondre `'0'` et `0`.**

AlgoProg 2015-16 37

## Type `double` : exemple

```
#include <stdio.h>
#include <stdlib.h>
double perimetreCercle(double r)
{
    double pi=3.1415;
    return (2*pi*r);
}
int main(void) {
    double R = 5.0;
    double p = perimetreCercle(R);
    printf("Rayon %g Perimetre %g\n", R, p);

    return EXIT_SUCCESS;
}
```

AlgoProg 2015-16 38

## Type char : exemple

```
#include <stdio.h>
#include <stdlib.h>
char caractereSuivant(char c)
{
    c=c+1;
    return (c);
}
int main(void) {
    char p ='A' ;
    char ps = caractereSuivant(p) ;
    printf("Caractere apres %c est %c\n", p, ps);
    return EXIT_SUCCESS;
}
```

AlgoProg 2015-16 41

## Quelques opérateurs arithmétiques

Dans l'ordre de priorité décroissante :

- ▶ + - opérateurs unaires signe
- ▶ \* / % opérateurs arithmétiques
- ▶ + - opérateurs arithmétiques
- ▶ = affectation

Donc  $x+y*2$  correspond à  $x+(y*2)$



La division / n'a pas le même sens sur entiers et flottants.

AlgoProg 2015-16 42

## Le type bool.

Le type Booléen (`bool`)

- ▶ n'a que 2 valeurs possibles : `true` et `false`,
- ▶ on l'utilise pour exprimer les résultats d'une expression logique,
- ▶ a été introduit dans la norme C99,
- ▶ il faut ajouter la directive :

```
#include <stdbool.h>
```

aux fichiers sources dans lesquels on l'utilise.

- ▶ en mémoire `true` est codé par l'entier 1, `false` est codé par l'entier 0.
- ▶ Dans une condition, toute expression
  - non nulle est considérée comme vraie.
  - nulle est considérée comme fausse.

AlgoProg 2015-16 43

## Quelques opérateurs logiques

Dans l'ordre de priorité décroissante :

- ▶ ! NON logique
- ▶ < > <= >= comparaisons (calcule un Booléen)
- ▶ == != comparaisons (calcule un Booléen)
- ▶ && ET logique
- ▶ || OU logique.

Donc  $x < y == z > t$  correspond à  $(x < y) == (z > t)$

En C, l'évaluation des opérateurs `&&` et `||` est «paresseuse».

AlgoProg 2015-16 44

## Quelques opérateurs logiques

and	V	F
V	V	F
F	F	F

or	V	F
V	V	V
F	V	F

	V	F
not	F	V

	or	and
not	and	or

AlgoProg 2015-16 45

## Type Bool : Exemples

bool b	Valeur de b
<code>b =(true &amp;&amp; false);</code>	false
<code>b =(3 &amp;&amp; 0);</code>	false
<code>b =(3 &amp;&amp; 5);</code>	true
<code>b =(3    5);</code>	true
<code>b =(3    0);</code>	true
<code>b =(0    0);</code>	false

AlgoProg 2015-16 52

## Type Bool : Exemples

bool b	Valeur de b
<code>b =(!8);</code>	false
<code>b =(!0);</code>	true
<code>b =( (!0) &amp;&amp; (3  0));</code>	true
<code>b =(3 == 3);</code>	true
<code>b =(3 !=2);</code>	true

AlgoProg 2015-16 58

## Type Bool : Exemples

Rappel loi de Morgan :

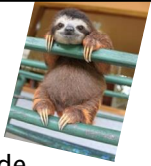
- ▶ `non (A ou B) = non (A) et non (B)`
- ▶ `non (A et B) = non (A) ou non (B)`

```
bool B = ((!(a||b)) == (!(a)&&(!b)));
/*vaut true*/
```

```
bool B = ((!(a&&b)) == (!(a)||(!b)));
/*vaut true*/
```

AlgoProg 2015-16 59

## Evaluation paresseuse



- ▶ Les opérateurs `&&` et `||` sont paresseux
  - ▶ L'évaluation des expressions se fait de gauche à droite
  - ▶ Elle s'arrête dès que possible

```
int a =0;
bool B=(a && (b==c)); /*vaut false*/
int a =1;
bool B=(a ||(b==c)); /*vaut true*/
```



Dans les 2 cas `b==c` n'est pas évaluée

## Affectation

- ▶ Permet de mémoriser une valeur.
- ▶ Affectation de variable :

```
<nom_variable> = <expression>
```

- ▶ Plus généralement :

```
<l_valeur> = <expression>
```

- ▶ `l_valeur` : expression qui a une adresse. Typiquement : variable.

- ▶ L'affectation

- évalue (calcule) l'expression,
- puis, range la valeur calculée à l'adresse de la `l_valeur`.
- est elle même une expression.

## Affectation

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

```
a = 3;
b = 4;
c = 5;
```

```
c=a-1;      /*c vaut 2 maintenant */
b=b+1;      /*b vaut 5 maintenant */
a=a+b+c;    /*a vaut 3+5+2 donc 10 maintenant */
```

```
a+b=2;      /* INTERDIT! */
             /*car a+b n'a pas d'adresse*/
             /* contrairement à a, b ou c */
```

## Tests

- ▶ Deux instructions similaires : `if` et `if...else`.

```
if( <expression> )
  <instruction ou bloc>vrai
```

```
if( <expression> )
  <instruction ou bloc>vrai
else
  <instruction ou bloc>faux
```

## Tests

- ▶ Deux instructions similaires : **if** et **if...else**.

```
if( <expression> )  
<instruction ou bloc>_vrai
```

```
if( <expression> )  
<instruction ou bloc>_vrai  
else  
<instruction ou bloc>_faux
```

- ▶ Permettent d'exécuter ou non des **instructions** suivant la validité de la condition indiquée par l'**expression**.
- ▶ L'expression booléenne peut utiliser les opérateurs
  - **&&** (ET logique)
  - **||** (OU logique)
  - **!** (NEGATION logique)

AlgoProg 2015-16 64

## Instruction if

```
if( <expression> )  
<instruction ou bloc>_vrai
```

- ▶ L'**expression** est évaluée. Si elle est
  - **non nulle**, le test réussit et **<instruction ou bloc>\_vrai** est exécuté.
  - **nulle**, le test échoue. On passe alors à l'instruction qui suit **<instruction ou bloc>\_vrai**, qu'on n'exécute pas.
- ▶ Si on veut exécuter **plusieurs** instructions en cas de test réussi, on doit les mettre **dans un bloc**, entre **{...}**.
- ▶ S'il n'y a qu'une instruction, les accolades ne sont pas nécessaires

AlgoProg 2015-16 65

## Instruction if : exemple

```
#include <stdio.h>  
  
int valeurAbsolue(int x)  
{  
    int res =x;  
    if (x<0)  
        res=-x;  
  
    return(res);  
}
```

AlgoProg 2015-16 66

## Instruction if : exemple

```
#include <stdio.h>  
  
int valeurAbsolue(int x)  
{  
    int res =x;  
    if (x<0)  
    {  
        res=-x;  
        printf("valeur Abs de %d est %d\n",x, res);  
    }  
    return(res);  
}
```

AlgoProg 2015-16 67

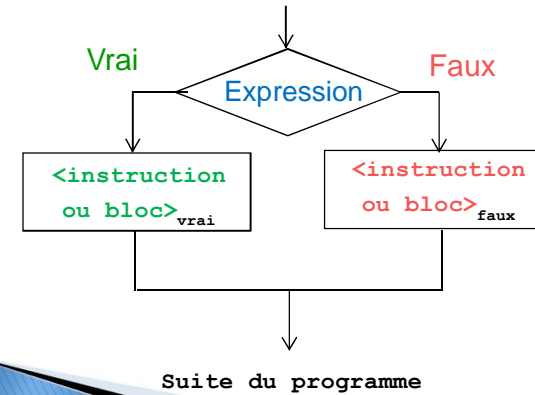
## Instruction if ... else

```
if( <expression> )
  <instruction ou bloc>_vrai
else
  <instruction ou bloc>_faux
```

- ▶ L'**expression** est évaluée. Si elle est
  - **non nulle**, le test réussit et **<instruction ou bloc>\_vrai** est exécuté.
  - **nulle**, le test échoue et **<instruction ou bloc>\_faux** est exécuté.
- ▶ Si on veut exécuter **plusieurs** instructions sous le **if** ou le **else**, on doit les mettre **dans un bloc**, entre {...}.
- ▶ Sinon, les accolades ne sont pas nécessaires.

AlgoProg 2015-16 68

## Instruction if ... else



AlgoProg 2015-16 69

## Instruction if ... else : exemple 1

```
void pair(int n)
{
  if (n%2==0)
    printf("%d est pair.\n",n);
  else
    printf("%d est impair.\n",n);
}
```

AlgoProg 2015-16 70

## Instruction if ... else : exemple 2

```
int pair(int n)
{
  int p=0;
  int i=1;
  if (n%2==0)
    return p;
  else
    return i;
}
```

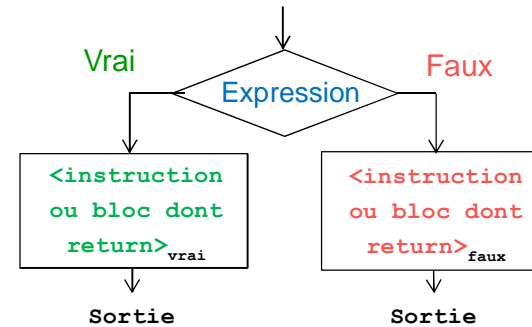
AlgoProg 2015-16 71

## Instruction if ... else : exemple 2

```
int pair(int n)
{
    int p=0;
    int i=1;
    if (n%2==0)
        return p;
    return i;
}
```

AlgoProg 2015-16 72

## Instruction if ... else



AlgoProg 2015-16 73

## Instruction if ... else : exemple 3

```
int pair(int n){
    int p=0;
    int i=1;
    if (n%2==0)
    {
        printf("Pair\n");
        return p;
    }
    else
    {
        printf("Impair\n");
        return i;
    }
}
```

AlgoProg 2015-16 74

## Instruction if ... else : exemple 3

```
int pair(int n){
    int p=0;
    int i=1;
    if (n%2==0)
    {
        printf("Pair\n");
        return p; }
    printf("Impair\n");
    return i;
}
```

AlgoProg 2015-16 75

## Boucle for

```
for( <expr1>; <expr2>; <expr3> )  
<instruction ou bloc>
```

- ▶ <expr1> est une expression d'initialisation exécutée une seule fois (au début de l'exécution de la boucle).
- ▶ <expr2> est un test (d'arrêt) exécuté avant chaque itération.
- ▶ <expr3> est une expression, modifiant au moins une variable de <expr2>, exécutée à la fin de chaque itération.

<expr1>, <expr2> ou <expr3> peuvent être vides.  
for ( ; ; ) /\* boucle infinie ! \*/

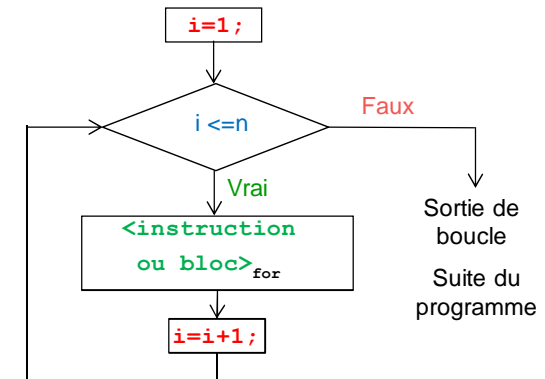
## Boucle for : Exemple 1

```
#include <stdio.h>  
  
void afficherSommeEntiers(int n)  
/* affiche la somme des n premiers nombres entiers*/  
{  
    int res = 0; /* initialisation */  
    for (int i = 1; i <= n; i = i + 1)  
        res = res + i;  
    printf( "Somme des entiers inferieurs à %d= %d  
    \n" , n, res);  
}
```

## Boucle for : Exemple 1

```
#include <stdio.h>  
  
void afficherSommeEntiers(int n)  
/* affiche la somme des n premiers nombres entiers*/  
{  
    int res = 0; /* initialisation */  
    for (int i = 1; i <= n; i++)  
        res = res + i;  
    printf( "Somme des entiers inferieurs à %d: %d  
    =\n" , n, res);  
}
```

## Boucle for : Exemple 1





## Boucle for : Exemple 2

```
#include <stdio.h>

void
afficherMajuscules (void)
/* affiche les 26 caracteres consecutifs apres
'A' */
{
for (char c = 'A' ; c <= 'Z'; c = c+1)
    printf("%c ", c);
printf("\n");
}
```

AlgoProg 2015-16 80

## Boucle for : Exemple 2

```
#include <stdio.h>

void
afficherMajuscules (void)
/* affiche les 26 caracteres consecutifs apres 'A'
*/
{
for (char c = 'A' ; c <= 'Z'; c = c+1) {
    printf("%c ", c);
    printf("\n");
}
}
```

AlgoProg 2015-16 81

## Boucle for : Exemple 3

```
#include <stdio.h>

void
afficherMajusculesInv (void)
/* affiche les 26 caracteres consecutifs avant
'Z' */
{
for (char c = 'Z' ; c >= 'A'; c = c-1){
    printf("%c ", c);
    printf("\n");
}
}
```

AlgoProg 2015-16 82

## Boucle while

```
while( <expression> )
    <instruction ou bloc>
```

1. L'expression est évaluée.
2. Si elle est fausse, on sort du **while**, on passe à l'instruction suivante.
3. Si elle est vraie,
  - ▶ **<instruction ou bloc>** est exécuté,
  - ▶ puis on revient en 1.

AlgoProg 2015-16 83

## Boucle while : Exemple 1

```
#include <stdio.h>

void
AfficherPuissances(int k, int nmax)
{
    int i = 1; /* initialisation */
    while (i <= nmax)
    {
        printf( "%d\n" , i);
        i = i * k;
    }
}
```

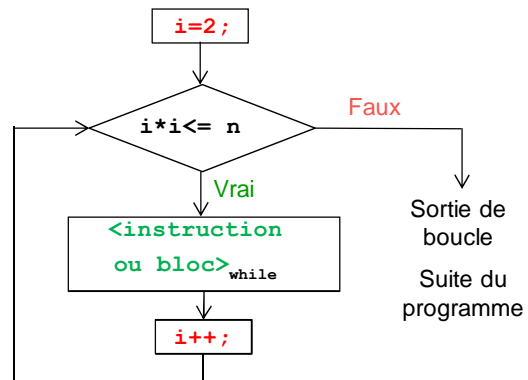
AlgoProg 2015-16 84

## Boucle while : Exemple 2

```
bool estPremier(int n)
{
    int i = 2;
    while (i*i<= n)
    {
        if (n % i == 0)
        {
            printf( "%d est divisible par %d\n" , n, i);
            return false;
        }
        i++;
    }
    printf("%d est premier\n", n);
    return true;
}
```

AlgoProg 2015-16 85

## Boucle while : Exemple 2



AlgoProg 2015-16 86