Université
Paris Cité

COURS 2

Programmation impérative

Langage C – Eléments de base

- les opérateurs
- Les structures de contrôle

SOMMAIRE

Opérateurs

- Opérateurs à 1, 2 ou 3 opérandes : unaires, binaires, ternaires
  - Opérateurs unaires non logiques et non bas niveau

Opérateur	Utilisation	

(type)	cast ou changement temporaire de type
	conversion explicite de type
		int x=45;
		float res;
		res=(float) x;

&	opérateur de déréférencement ou d'adresse retourne l'adresse en mémoire d'une variable	&x;
*	opérateur d'indirection ou de déréférencement sur une adresse permet d'accéder au contenu d'une variable à partir d'une adresse	*(&x);
sizeof	opérateur donnant la taille en octets d'un type	sizeof(int)
-	moins unaire, inversion de signe	int y=-3; x=-y;
++	incrémentation	x++;
--	décrémentation	x--;

Opérateurs

- Opérateurs arithmétiques et d ’affectation binaires

Opérateur	Utilisation
+	Addition arithmétique
-	Soustraction arithmétique
*	Multiplication arithmétique
/	Division entière ou réelle (dépend du type des opérandes)
%	Reste de la division entière 10%8 => 2
=	Opérateur d’affectation

☞ * et / sont prioritaires

Opérateurs

- Opérateurs arithmétiques et Type des opérandes
  - Si les opérandes ne sont pas de même type

⇔le compilateur agrandit en général l'opérande le plus étroit ⇔ conversion implicite

```
int      x=2;
float    res, y=4.5;
res = x+y;
```

⇔ int + float ⇔ le type int est étendu temporairement au type float ⇔ float + float = float

⇔ l 'affectation d 'un type étroit à un type plus large n'entraîne pas de problème alors que l'inverse peut entraîner une perte d'information

```
int      x=2, y= 6;
float    res;
res = x+y;    //8.0000
```

int + int = int

type du résultat (int)étendu au type float

6 + 2 = 8 **8.0000 <= 8**

```
float   x = 4.5,  y=5.6;
int     res;
res = x + y;  // 10  au lieu de 10.1
```

float + float = float 4.5+5.6=10.1

affectation d'un type large (float) à un type plus étroit au type **int**

⇔4.5+5.6=10.1 perte des chiffres après virgule

10 <= 10.100000

Opérateurs

- Type des opérandes et opérateur /

Type opérandes	Division entière ou réelle ?	int x=2, y= 6; float res;
int / int	**division entière**	res = x / y; ⇔2/6=0 résultat étendu au type float res 0.0000 x 2 y 6

float / int
 	
int / float
 	
float / float
	division réelle	res = (float) x / y;
		ou
		
		res= x / (float) y res 0.33333 x 2 y 6
		
		ou
		
		res = (float) x / (float) y;

Opérateurs

- Combinaison Opérateurs d'incrémentation/décrémentation et d'affectation

Position de l'opérateur ++/variable à incrémenter (même chose pour --)	**Quelle opération en 1er ?**	**int v1, var =2;** v1 ? var 2

v1 = var++;
⇔
v1= var;
var++;
	var est incrémentée après l 'affectation	v1 2 var 2 puis v1 2 var 3

v1 = ++var;
⇔
var++;
v1= var;
	var est incrémentée avant l 'affectation	v1 ? var 3 puis v1 3 var 3

Opérateurs

- Combinaison Opérateurs d'arithmétique et d'affectation

Opérateurs combinés	Instructions équivalentes	int var =10; var 10

+=	var += 4;
	⇔
	var = var + 4;
		var 14

-=	var -= 4;
	⇔
	var = var - 4;
		var 6

*=	var *= 4;
	⇔
	var = var * 4;
		var 40

/=	var /= 4;
	⇔
	var = var / 4; //division entière
		var 2

%=	var %= 3;
	⇔
	var = var%3; //reste division
	//entière
		var 1

Opérateurs

- Opérateurs binaires de comparaison

Opérateur	Utilisation
> <	opérateur de test de supériorité, d'infériorité
>= <=	opérateur de test de supériorité ou égalité, d'infériorité ou égalité
= =	opérateur de test d'égalité
!=	opérateur de test de différence

A ne pas confondre avec = opérateur d'affectation

Opérateurs

- Opérateurs unaires et binaires logiques

Opérateur	Utilisation
!	opérateur de négation
&&	opérateur de ET Logique
||	opérateur de OU Logique

x	V	V	F	F
y	V	F	V	F
x&&y	V	F	F	F
x||y	V	V	V	F
!x	F	F	V	V

Opérateurs

- Opérateurs unaires et binaires logiques

Opérateur	Utilisation
!	opérateur de négation
&&	opérateur de ET Logique
||	opérateur de OU Logique

x	V	V	F	F
y	V	F	V	F
x&&y	V	F	F	F
x||y	V	V	V	F
!x	F	F	V	V

Opérateurs

- Opérateurs unaires et binaires de traitement bas niveau
  - directement au niveau des bits
  - s'appliquent sur des int ou des char

Opérateur	Utilisation	
~	donne le complément à un d'une opérande inversion de tous les bits	$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ 28 0 0 0 1 1 1 0 0 ~28 1 1 1 0 0 0 1 1 (227)
&	ET réalise un ET logique sur chacun des bits des 2 opérandes Sert à masquer certains bits	$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ 6 0 0 0 0 0 1 1 0 & 10 0 0 0 0 1 0 1 0 2 0 0 0 0 0 0 1 0
|	OU inclusif réalise un OU logique sur chacun des bits des 2 opérandes met à 1 tous les bits de op1 qui sont à 1 dans op2	$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ 1 0 0 0 0 0 0 0 1 | 121 0 1 1 1 1 1 0 1 121 0 1 1 1 1 1 0 1

Opérateurs

- Opérateurs binaires de traitement bas niveau (suite)

Opérateur	Utilisation	
^	OU exclusif met à 0 tous les bits de op1 qui sont identiques dans op2, et à 1 ceux qui diffèrent	$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ 1 0 0 0 0 0 0 0 1 ^ 121 0 1 1 1 1 1 0 1 120 0 1 1 1 1 1 0 0
<< op1<<op2	opérateur de décalage à gauche décalage vers la gauche de op2 bits de op1, les bits de droite sont remplacés par des 0	$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ 1 0 0 0 0 0 0 0 1 1<<2 0 0 0 0 0 1 0 0 (4) Chaque décalage à gauche ⇔ *2
>> op1>>op2	opérateur de décalage à droite décalage vers la droite de op2 bits de op1, les bits de droite sont remplacés par des 0 pour les entiers non signés, par des 0 ou bit de signe pour les entiers signés	$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ 8 0 0 0 0 1 0 0 0 8>>2 0 0 0 0 0 0 1 0 (2) Chaque décalage à droite ⇔ /2

Opérateurs

- Opérateurs ternaire

Opérateur	Utilisation	

? :
op1? op2 :op3
	Met en jeu 3 opérandes (ici des expressions) et construit une opération de test avec retour de valeur
	Opérande1 ⇔ condition du test
	Si la condition est vérifiée, l'opérande 2 est évaluée et le résultat de l'évaluation est retourné
	Sinon c'est l'opérande 3 qui est évaluée et le résultat de l'évaluation est retourné
		int a=3, b=5, res;
		res = (a<=b) ? a : b;
		a 3 b 5 res ~~?~~3

ValeurAbsolue.c

```c
/*
 * Calcul et affichage de la valeur absolue
 */

#include <stdio.h>
#include <stdlib.h>

int main(){

    float nb;
    float val_abs;
    printf("Entrez une valeur numérique\n");
    fflush(stdout);
    scanf("%f",&nb);
    val_abs= (nb<0)? -nb : nb;
    printf("la valeur absolue de %.2f est %.2f\n",nb,val_abs);
    return EXIT_SUCCESS;
}
```

Problems Tasks Console Properties

<terminated> ProgrammeValeurAbsolue.exe [C/C++ Local App

```
Entrez une valeur numérique
-78.56
la valeur absolue de -78.56 est 78.56
```

Problems Tasks Console Properties

<terminated> ProgrammeValeurAbsolue.exe [C/C++ Local App

```
Entrez une valeur numérique
23.75
la valeur absolue de 23.75 est 23.75
```

Opérateurs

- Priorité

+ prioritaire

- prioritaire

( ), [ ], ->, .
~, ++, --, (signe +/-), *(adresse), &, sizeof(var), !
*, /, %
+ , -
<<, >>
< <=, > >= (opérateurs de comparaison)
==, !=
&
^
|
&&
||
? :
=, tous les combinés (+=, -=, *=, /= …)

Opérateurs

- Ordre de priorité
  - **les opérateurs d'une expression sont évalués sur la base de priorités**
    - Les opérateurs ayant une priorité supérieure sont appliqués avant les opérateurs ayant une priorité inférieure
    - si les opérateurs ont la même priorité dans la même expression, cette expression sera évaluée de **gauche à droite**
      - **4 + 5 * 6 / 3** sera évalué comme **4 + ((5 * 6) / 3)**

    sauf pour opérateur d'affectation
      - **a = b = c = 15** sera évalué comme **a = (b = (c = 15))**

Structures de contrôle

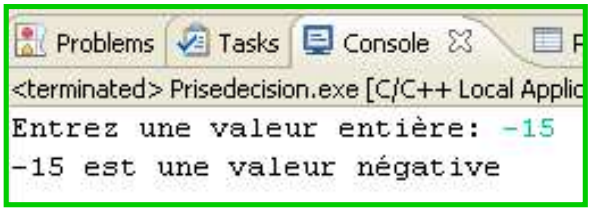
- Branchement conditionnel
  - Expression d'une prise de décision
    - Choix simple ⇔ si …..alors …. sinon
    - Choix multiple
- Bouclage
  - Répétition d'instructions

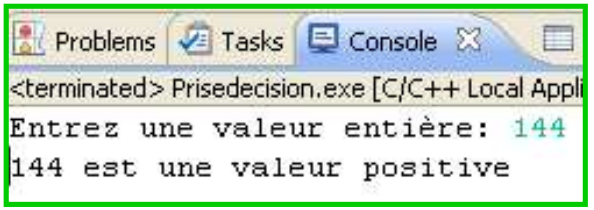
Structures de contrôle

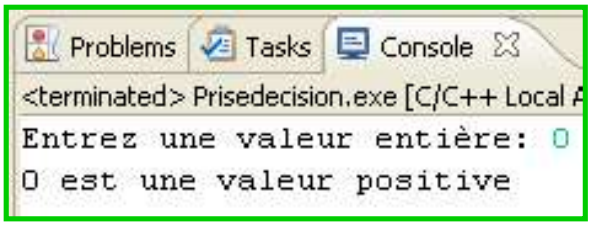
- Prise de décision par choix simple

if (condition){
….
}else{
…..
}

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int x;
    printf("Entrez une valeur entière: ");
    fflush(stdout);
    scanf("%d",&x);

    if(x<0){
        printf("%d est une valeur négative",x);
    }else {
        printf("%d est une valeur positive\n", x);
    }
    return EXIT_SUCCESS;
}
```







Structures de contrôle

- Prise de décision par choix simple
  - Les *if* et *else* peuvent s'enchainer

```c
#include <stdio.h>

int main(){
    int x;
    printf("Entrez une valeur entière: ");
    fflush(stdout);
    scanf("%d",&x);

    if(x<0){
        printf("%d est une valeur inférieure strictement à 0",x);
    }else if (x>0){
          printf("%d est une valeur supérieure strictement à 0\n", x);
          }else {
              printf("%d est une valeur égale à 0\n", x);
          }
    return 1;
}
```

```
Problems  Tasks  Console  Properties
<terminated> Prisedecision.exe [C/C++ Local Application] D:\Enseignement
Entrez une valeur entière: -54
-54 est une valeur inférieure strictement à 0
```

En l'absence de { }, un *else* se raccroche toujours au *if* immédiatement supérieur en partant du bas

```
Problems  Tasks  Console  Properties
<terminated> Prisedecision.exe [C/C++ Local Application] D:\E
Entrez une valeur entière: 0
0 est une valeur égale à 0
```

```
Problems  Tasks  Console  Properties
<terminated> Prisedecision.exe [C/C++ Local Application] D:\Enseignement
Entrez une valeur entière: 121
121 est une valeur supérieure strictement à 0
```

Structures de contrôle

- Prise de décision par choix multiple
  - Pour éviter les imbrications de *if* dans le cas de choix multiples sur des valeurs constantes
  - switch – syntaxe

Ne pas oublier l'instruction *break* à la fin de chaque *case*

Syntaxe table de branchement	Règles

switch(expression){
case value1: instruction1;
instruction 2;
.....
break;
case value2 :
case value3 : instruction1;
break;
default : instruction1;
}
	•L'expression est évaluée comme une valeur entière
	•Les valeurs des case sont évaluées comme des constantes entières (int, short, char)
	•L'exécution se fait à partir du **case** dont la valeur correspond à l'expression et se termine à la rencontre de l'instruction **break**
	•Les instructions qui suivent la condition **default** sont exécutées quand aucune constante des **case** n'est égale à la valeur évaluée de l'expression

Structures de contrôle

- switch
  - illustration

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

int main(void) {
    int x;
    printf("Entrez un nombre entier entre 0 et 9: \n");
    fflush(stdout);
    scanf("%d",&x);
    if(x==6 || x==8){
        printf("Le nombre est supérieur à 5\n");
        printf("Le nombre est pair\n");
    }else if(x==4 || x==2 || x==0){
            printf("Le nombre est pair\n");
         }else if(x==7 || x==9){
                    printf("Le nombre est supérieur à 5\n");
                    printf("Le nombre est impair\n");
                }else if (x==5 || x==3 || x==1)
                        printf("Le nombre est impair\n");
                     else
                        printf("ce n'est pas un nombre compris entre 0 et 9");
    return EXIT_SUCCESS;
}
```

TestSwitch.c

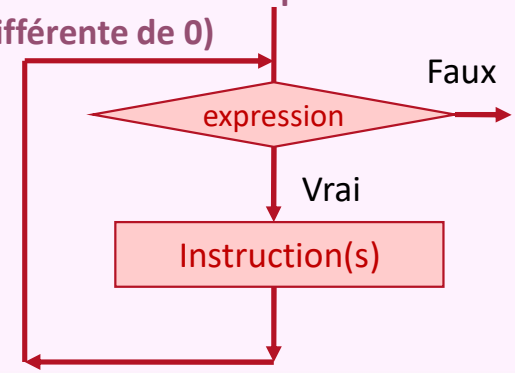
```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int x;
    printf("Entrez un nombre entier entre 0 et 9: \n");
    fflush(stdout);
    scanf("%d",&x);
    switch(x){
        case 0: case 2: case 4: printf("Le nombre est pair\n");
                                break;
        case 1: case 3: case 5 : printf("Le nombre est impair\n");
                                 break;
        case 6: case 8: printf("Le nombre est supérieur à 5\n");
                        printf("Le nombre est pair\n");
                        break;
        case 7: case 9: printf("Le nombre est supérieur à 5\n");
                        printf("Le nombre est impair\n");
                        break;
        default:        printf("ce n'est pas un nombre compris entre 0 et 9");
                        break;
    }
    return EXIT_SUCCESS;
}
```

Structures de contrôle

- Répétitions d'une suite d'instructions ⇔ bouclage

Répétition d'instructions	Règles

while(expression){
  instructions;
}
	• Répétition des instructions tant que la valeur de l'expression s'interprète comme vraie (différente de 0)



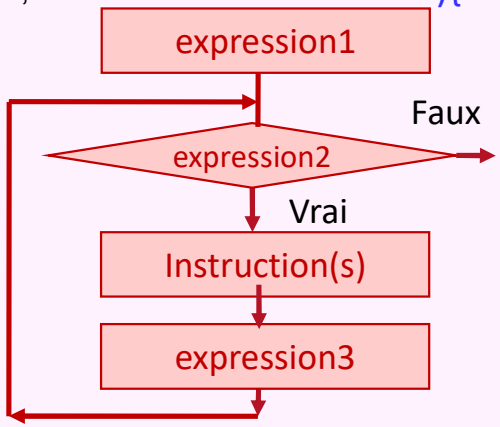

```
int  i = 0 ;            //initialisation
while(i < 10){      //condition
        printf("Bonjour N° %d\n ",i) ;
        printf("----------------\n ") ;
        i++ ;     //incrémentation
                  //ou modification de la valeur la variable impliquée dans la condition
}
```

Structures de contrôle

- Répétitions d'une suite d'instructions ⇔ bouclage

Répétition d'instructions	Règles

for(initialisation; condition_vraie; instruction d'incrémentation){
instructions;
}
	• Le for s'utilise avec 3 expressions séparées par des ; et qui peuvent être vides
	
	expression 1 : instruction d'initialisation, exécutée une fois
	
	expression 2 : condition d'exécution des instructions, testée à chaque tour de boucle
	
	expression 3 : permet de calculer la prochaine valeur avec laquelle la condition va être testée



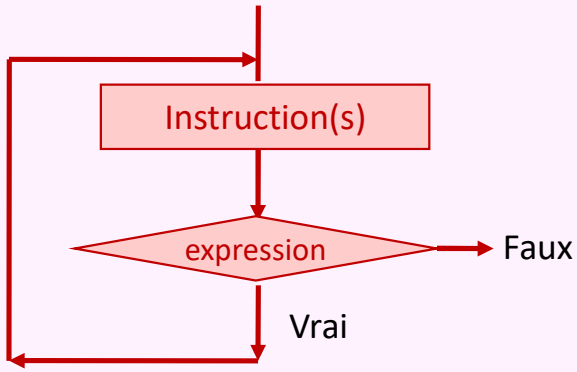

```
int   i;
for(i = 0 ; i < 10 ; i++ ){
  printf("Bonjour N° %d\n ",i) ;
  printf("----------------\n ") ;
}
```

```
int   i, j;
for(i = 0, j=10 ; i < j ; i++, j--){
  …… ;
}
```

Structures de contrôle

- Répétitions d'une suite d'instructions ⇔ bouclage

Répétition d'instructions	Règles
do{ instructions; } while(expression);	• test de continuation se fait en fin de boucle ⇔ on passe au moins une fois dans la boucle

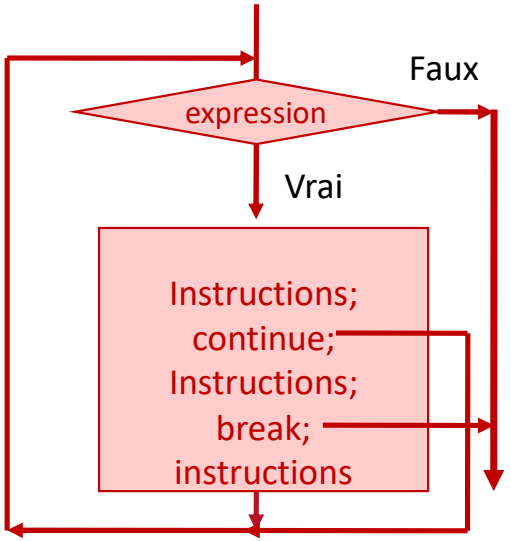



```
int  i = 0 ;              //initialisation
do{
  printf("Bonjour N° %d\n ",i) ;
  printf("----------------\n ") ;
  i++ ;
}while(i < 10) ;
```

Structures de contrôle

- Structures itératives peuvent être imbriquées
  - Respecter les règles d'indentation pour la lisibilité du code
- Ruptures de séquences répétitives
  - Instructions correspondant aux mots clés

Mots clés	Utilisation
~~goto +label~~	~~Permet de sortir d'une séquence et d'aller directement au niveau de l'étiquette précisée par label~~
continue	Provoque le passage à l'itération suivante de la boucle, en sautant directement à la fin de la structure répétitive
break	Provoque la sortie de la structure répétitive



- Le recours à ces mots clés est souvent lié à une mauvaise réflexion au niveau des tests d'arrêt des structures répétitives