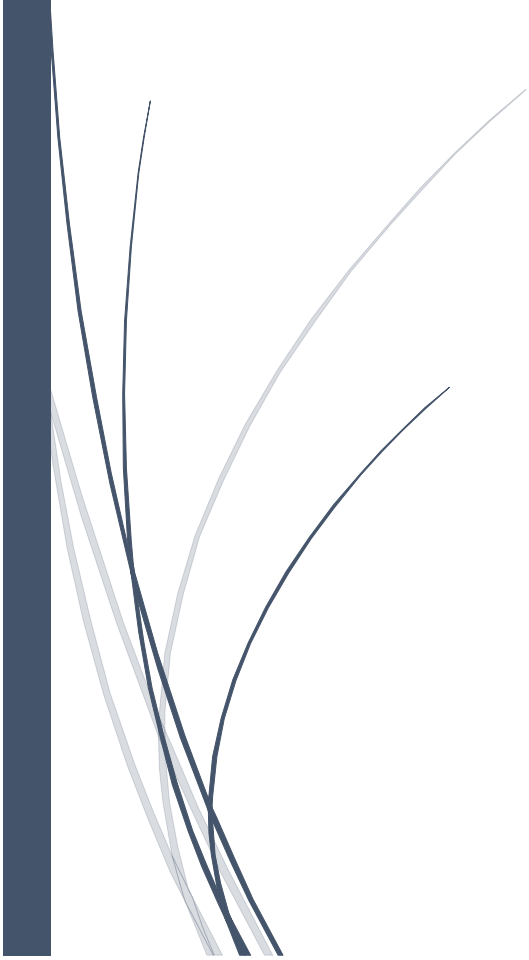
2023/2024

Informatique III

Programmation Orientée Objet en C++



UNIVERSITY BATNA 2 DEPARTEMENR MATHEMATIQUES

Chapitre1 : Introduction aux concepts de la Programmation Orientée Objet

Il convient tout d'abord de se rappeler en quoi consiste la programmation dite *procédurale* pour comprendre ce qui diffère en *Programmation Orientée Objet* (POO).

1.La programmation procédurale

Avec des langages tels que le C ou le Pascal, la résolution d'un problème informatique passe généralement (mais pas nécessairement) par l'analyse descendante. On décompose ainsi un programme en un ensemble de sous programmes appelés qui coopèrent pour la résolution d'un problème.
Les procédures et fonctions sont généralement des outils qui produisent et/ou modifient des structures de données.

La programmation procédurale suit l'équation de Wirth :

ALGORITHMES + STRUCTURES DE DONNEES = PROGRAMME

Inconvénient de la programmation procédurale

L'évolution d'une application développée suivant ce modèle n'est pas évidente. En effet, la moindre modification des structures de données d'un programme conduit à la révision de toutes les procédures manipulant ces données. En outre, pour de très grosses applications, le simple fait de répertorier toutes les fonctions manipulant une structure de données peut déjà être problématique.

2. Programmation Orientée Objet (POO) :

La Programmation Orientée Objet (POO) est un paradigme de programmation qui utilise des objets et des classes pour organiser et structurer le code. En C++, la POO est largement utilisée et offre des fonctionnalités puissantes pour créer des applications modulaires et réutilisables. Voici une introduction à la POO en C++ :

2.1 Objet :

Un objet est une instance d'une classe. Il contient des données (variables) et des méthodes (fonctions) qui agissent sur ces données. Par exemple, un objet "Voiture" peut avoir des variables telles que la vitesse et la couleur, ainsi que des méthodes pour accélérer et freiner.

Exemple1

Soit l'objet $Z = a + i\,b$ un nombre complexe dont a est la partie réelle et b la partie imaginaire.
Cet objet est caractérisé par les données *a et b*. Il est manipulé par des traitement ou procédures caractérisées par leurs noms suivi de paramètres réels et formels d'appels et de résultats qui sont par exemple ModuloZ(a, b), ARGZ(a, b) etc…

Exemple2

Soit l'EDP à 2D suivante :

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \alpha \dfrac{\partial^2 u}{\partial y^2} = 0, \\ CI, CL\,. \end{cases}$$

Cet objet est caractérisé par les données *de* conditions initiales *''CI'',* de conditions aux limites *‘''CL''* et la constante ''α'' ainsi que d'autres variables qui peuvent apparaitre lors de la résolution du problème comme le pas de discrétisation, la variable matrice etc.

Et pour les méthodes est caractérisée par :

- *Lecture des données*
- *Discrétisation*
- *Impression des résultats…*

2.2 Classe d'objet

Une classe est un plan ou un modèle à partir duquel les objets sont créés. Elle définit les variables et les méthodes que les objets auront. Par exemple, une classe "Voiture" pourrait définir les variables vitesse et couleur, ainsi que les méthodes accélérer et freiner.

Exemple de classes en C++

```
// point.h
#include<stdio.h>
class point
{
private:
float x;
float y;
public:
void Init(float abs, float ord){ x = abs; y = ord;}
float GetX() { return x; }
float GetY() { return y; }
void Affiche(){ printf("( %f , %f )\n", x , y ); }
};
int main()
{
point p1;
p1.Init(10,20);
p1.Affiche();
printf("abscisse de p1 : %f\n", p1.GetX());
return 0;
}
```

Ce premier exemple en C++ définit une classe appelée point. Les objets de cette classe représentent tout simplement des points du plan.

Les données membres : les champs x et y de type float représentent les données membres des objets de cette classe. Ces champs permettront simplement de stocker des coordonnées. L'attribut private indique que ces champs ne sont pas accessibles depuis l'extérieur de l'objet (les données sont encapsulées).

Les méthodes : les fonctions Init(), GetX(), GetY() et Affiche() sont les méthodes de cette classe. Ces fonctions joueront le rôle d'interface.

Init() : permet l'initialisation des coordonnées d'un point

GetX(), GetY() : retournent respectivement l'abscisse et l'ordonnée d'un point

Affiche() : permet l'affichage à l'écran d'un point sous la forme « (x,y) »

L'accès à un membre d'un objet se fait par l'opérateur « . », comme pour les types structurés. Par exemple, la syntaxe

p1.Affiche();

correspond à l'appel de la méthode Affiche() sur l'objet p1, ce qui permet à l'objet p1 de « s'afficher » à l'écran.

Remarque

Par défaut, si on n'utilise pas les niveaux de visibilités tout le contenu de la classe d'objet est ''private'' donc protégé et on ne peut pas réaliser les appels.

2.3 Encapsulation des données

L'encapsulation est le principe de cacher les détails internes d'un objet et de n'exposer que ce qui est nécessaire à l'extérieur. Cela se fait en utilisant des modificateurs d'accès tels que "private" et "public" dans une classe.

Trois niveaux de visibilités sont offerts par les langages orientés objet :

- Niveau public : les fonctions (traitements) de toutes les classes peuvent y accéder aux données ou aux ''méthodes'' d'une classe (le mot réservé utilisé est 'public').
- Niveau protégé : l'accès aux données est réservé aux fonctions de la classe héritières seulement (le mot réservé

utilisé est ‘protected’).
- Niveau privé : l’accès aux données est limité seulement aux méthodes de la classe d’objet elle-même (le mot réservé utilisé est ‘private’).

2.4 Héritage

L’*héritage* correspond à la possibilité de créer une classe d’objets, dite *classe dérivée*, à partir d’une classe existante, dite *classe de base*.
Dans le processus d’héritage, la classe dérivée « hérite » des caractéristiques (structures de données + méthodes) de la classe de base. En plus de cet héritage, on peut ajouter dans la classe dérivée des données et des fonctionnalités nouvelles. L’héritage conduit ainsi à une spécialisation de la classe de base.

Pourquoi créer une sous classe ?
La permission de la création d’une sous classe à partir d’une classe existante est autorisée pour éliminer la redondance d’informations.

Syntaxe :
<Nom de la nouvelle classe> hérite de <nom de la superclasse>
{- définition du champ ‘nouveau attributs’ de la sous classe
- définition du champ ‘nouvelles méthodes’ de la sous classe
} ;

2.5 Polymorphisme

Le Polymorphisme est un terme grec qui signifie plusieurs formes. Il est utilisé en langage C++ pour permettre à un code d’être utilisé avec différents types. Et l’un de plus grands avantages du polymorphisme est que le fait d'utiliser un même nom de méthode pour plusieurs types d'objets différents, permet au programmeur de ne pas se soucier du type précis de l’objet que la méthode va s’appliquer lorsqu'il programme cette dernière.

Chapitre2 : Extensions au langage C

Nous présentons dans ce chapitre les nouveautés apportées par le C++ en dehors des objets. Certaines d'entre elles améliorent la facilité d'utilisation du langage et peuvent justifier à elles-seules l'utilisation du C++ à la place du C. L'aspect objet du langage sera abordé ultérieurement.

1.Vocabulaire

Le langage de programmation orienté objets C++ met à notre disposition plusieurs formes de symboles, on distingue :

Les caractères numériques 0 ..9,

les caractères alphabétiques a..z et A..Z,
les caractères spéciaux tels que opérateurs arithmétiques, opérateurs de relations et opérateurs de comparaisons, des séparateurs, caractère de monnaie. L'utilisation et la manipulation de ces symboles exige de nous, de connaitre leurs natures et leurs syntaxe (forme d'écriture).

Remarques

- Il faut distinguer les lettres minuscules des lettres majuscules.
- Lors de de l'écriture des messages on peut utiliser même des caractères non utilisés par le POO**.**

2. Notion de type

Le langage de programmation orienté objets C++ dispose de plusieurs formes de types pour les variables et qui doivent obligatoirement identifier et déclarer au préalable avant de s'en servir dans les traitements. Les types les plus employés sont int, float, double, string, char et bool.

Syntaxe : <type> < variables> ;

Exemple

Soit x une variable de type entier, sa déclaration est : int x ;

3.Notion de variable

En C++, une variable est un identificateur, composé de caractères alphanumériques dont le 1er caractère est alphabétique. Il peut aussi comporter seulement le caractère spécial '' _ '' (celui de la touche du chiffre 8 du clavier). Cet identificateur est une quantité qui possède un type et peut varie en fonction du temps. On distingue deux sortes de variables :
- variable simple qui ne peut contenir qu'une seule valeur à un moment donné.

- variable indicée qui peut contenir plusieurs valeurs à un moment donné, ou bien une suite de cases contiguës qui permet de stocker de façon ordonnée des données de même type.

Remarques

- Une variable simple z se crée et se déclare par son type suivi de son nom.
- La déclaration des variables indicées (ou tableaux) en C++ se fait en spécifiant le type de données des éléments du tableau, suivi du nom du tableau et de sa taille (c'est-à-dire le nombre d'éléments qu'il peut contenir). Voici la syntaxe de base pour déclarer des tableaux en C++ :

typedef <type> <nom de la variable indicée> [taille1][taille2]
#define [taille1] < valeur correspondante à la variable taille1>
#define [taille2] < correspondante à la variable taille1

- L'initialisation des indices associés aux variables indicées se commencent toujours à partir de la valeur zéro.
- Le C++ utilise aussi le qualificatif ''const'' pour déclarer des variables indicées (Si on reprend l'exemple ci-dessus, alors on déclare comme suit :
  const taille1=20, taille2=30, taille3=10 ;
  float A[taille1][taille2];
  int Tab[taille3] ;

4. Les opérations d'entrées/sorties

On peut utiliser les instructions classiques d'entrée et sorties du langage procédural ou impératif C, qui exigent des formatages des données (le spécification de format), cependant, le C++ offre de nouvelles possibilités d'utiliser des instructions ne nécessitant pas de formatages et qui sont très simples à apprendre et à insérer dans des programmes, on distingue
- Instruction de lecture (ou introduction des données) : cin>>variables ;
- Instruction de sortie (ou d'impression des résultats) : cout<<variables ;

Syntaxe

cin>>n ; **équivalent à** scanf(''%d'',&n) ;
cout<<n ; **équivalente à** printf(''%d'',n) ;

On présente quelques bibliothèques correspondantes aux différentes instructions de C++ les plus souvent utilisé:

Bibliothèques	Instructions
stdio.h	Instruction d'entrées et sorties classiques: printf, scanf
iostream	Les nouvelles instructions d'entrées et de sorties : cin, cout
math.h	Toutes les fonctions standards de mathématiques
string	Pour la manipulation et la gestion de chaînes de caractères
conio.h	Instructions d'ordre techniques et organisationnelles : clscr, getch(), …

Remarques

La fonction **getch()** est l'abréviation de **get char** qui permet de lire un caractère ou une touche au clavier.

La fonction **clrscr** permet d'effacer l'écran.

5.Affectation

Une affectation est une assignation d'unrésultat à une variable en utilisant le symbole d'égalité ''=''.

Syntaxe: <Nom de variable> = <expression arithmétique ou valeur> ;

Le C++ autorise l'utilisation d'une autre forme d'assignation en utilisant les symboles de parenthèses '' ( )''.

Exemple

```
int x,y;
float k,
cin>>x>>y ;
k=(x+y)/x ; ⟺      k((x+y)/x) ;
```

6.Opérateurs logiques

Les opérateurs de comparaison et les opérateurs de relation sont des opérateurs logiques qui renvoient toujours un et un seul résultat vrai ou faux, ces opérateurs on les retrouve dans la syntaxe d'écritures des primitives conditionnelles d'actions qu'on décrira par la suite.

symbole	Désignation	Exemple	Résultat
<	Inférieur	4<2	0
>	Supérieur	5>3	1
>=	Supérieur ou égale	12>=4	1
<=	inférieur ou égale	3<=2	0
!=	Non égale (différent)	2=!1	1
==	Egale (égalité)	3==5	0
&&	Le 'et' logique	(4>2)&&(5==3)	0
||	Le 'ou' logique	(4>2) || (5==3)	1
!	La négation	!((4>2) || (5==3))	0

6. Opérateurs d'incrémentation et de décrémentation

Les opérations d'incrémentations et de décrémentations des variables peuvent être effectuée respectivement par des opérateurs unaires '++' et '--'. L'avantage de l'utilisation de ces opérateurs est qu'ils rendent des programmes plus courts, bien structurés et moins de risque de faire des erreurs lors de la l'écriture des programmes.

Exemple

```
int i=10 ;

int x=i++ ;

int y=i-- ;
```

Le traitement de cette séquence se fait dans l'ordre suivant:
1- On affecte à la variable simple de type entier i la valeur 10
1- On affecte à la variable simple entière x la valeur de i (donc x reçoit 10)
2- On incrémente la variable i de 1(i reçoit de nouveau i+1 et donc i prend la valeur 11)
3- On affecte maintenant à la variable entière simple y la valeur de i qui est 11, (y=11)
4- Et enfin on décrémente la variable i de 1 et on aura i=10

Cette forme est dite post-incrémentation/ post-décrémentation car l'opérateur ++/-- est situé après la variable.

De même on peut rencontrer des opérateurs pré-incrémentations et pré-décrémentations. Pour bien comprendre, reprenons l'exemple ci-dessus tout en changeant la position des opérateurs :

```
int i=10 ;
int x=++i ;
int y=--i ;
```

1- On affecte à la variable simple de type entier i la valeur 10
2- Avant d'affecter à la variable simple entière x la valeur de i, il faut incrémenter d'abord la variable i de1 (donc i reçoit 11)
3- puis affecter cette nouvelle valeur de i à la variable x et on aura x=11
4- de la même manière, effectuer la décrémentation de la variable i de 1 et donc on aura i=10
5- puis affecter cette nouvelle valeur de i à la variable y et on aura y=10

Remarque1 :

On peut aussi rencontrer une simplification d'écritures des expressions arithmétiques de la manière suivante :

x++ ; ⟷ x=x+1 ;

x+=10 ; ⟷ x=x+10 ;

x-=8 ; ⟷ x=x-8 ;

Remarque2 :

En C++, la priorité des opérations est similaire à celle de nombreux langages de programmation. Voici une liste générale de la priorité des opérations en C++, du plus élevé au plus bas :

1. Parenthèses : ()
- Les opérations à l'intérieur des parenthèses sont évaluées en premier.
2. Opérateurs unaires : ++x, --x
- Les opérateurs unaires sont appliqués avant les opérations binaires.
3. Multiplication, division et modulo : *, /, %
- Ces opérations sont évaluées de gauche à droite.
4. Addition et soustraction : +, -
- Ces opérations sont évaluées de gauche à droite.
5. Opérateurs de comparaison : <, >, <=, >=
- Ces opérations comparent les valeurs.
6. Égalité : ==, !=
- Ces opérations vérifient l'égalité ou la non-égalité des valeurs.
7. Opérateurs logiques : &&, ||
- Les opérations logiques && (ET) sont évaluées avant || (OU).
8. Opérateurs de l'assignation : =, +=, -=, *=, /= et autres opérateurs d'assignation combinés
- L'opération d'assignation est évaluée en dernier parmi les opérateurs binaires.
9. Ternaire (conditionnel) : ? :
- Il permet d'exprimer des conditions sous forme de condition ? valeur_si_vrai : valeur_si_faux.

Il est important de noter que cette liste est générale, et certains opérateurs peuvent avoir des priorités spécifiques en fonction du contexte. De plus, vous pouvez toujours utiliser des parenthèses pour spécifier l'ordre d'évaluation et éviter toute ambiguïté dans vos expressions**.**

7. Les Structures Conditionnelles et de Boucle

Ces structures sont au nombre de 4, il s'agit des instructions if (avec éventuellement else), for, switch et while. Ces instructions permettent d'exécuter des blocs d'instructions si certaines conditions sont remplies (if,switch), ou plusieurs fois de suite (for,while).

Un bloc d'instructions est une suite d'instructions réunies ensembles et encadrées par des accolades ouvrante '{' puis fermante '}'.

Lorsque vous utilisez des structures conditionnelles ou de boucle, il est conseill´e d'indenter l'´ecriture du code, pour des raisons de lisibilité.

7.1 **L'instruction if...else**

L'instruction if est l'instruction de test la plus simple, elle correspond à un test si ... alors. Elle réalise un bloc d'instructions si une condition est satisfaite. Elle s'écrit sous la forme :

if (condition) { liste d'instructions; }

Si la liste d'instructions ne comporte qu'une seule instruction, les accolades ne sont pas n´ecessaires. La condition est une expression booléenne et peut donc comporter des opérateurs logiques et des opérateurs de comparaison.

La plupart du temps, on souhaite exécuter un bloc d'instructions si une condition est réalisée mais aussi exécuter un autre bloc d'instructions dans le cas contraire. D'où l'instruction if...else, correspondant à un si ... alors ... sinon

if (condition)

{ liste d'instructions; }

else

{ liste d'instructions; }

Remarque

Il existe un opérateur logique mais un opérateur logique ternaire **?:** (mais peut être un résultat qcq) qui peut être utilisé comme alternative à if_else.

Exemple

```
int a,b,max ;
a=3 ;
b=2 ;
if (a>b)
   max=a;
else
   max=b;
```

peut être remplacée par :

```
int a,b,max ;
a=3 ;
b=2 ;
max=(a > b)? a :b;
```

7.2 L'instruction switch

Cette instruction permet de tester une variable et d'exécuter différentes instructions suivant la valeur de son contenu. Il s'agit d'un branchement conditionel, sa syntaxe est la suivante :

```
switch (Variable){
 case Valeur1:
liste d'instructions;

break; case Valeur2:
 liste d'instructions;
break;
....
default:
liste d'instructions;
 break;
```

```
}
```

Exemple

```
#include<stdio.h>

int main(void){
   int jour;

   printf("saisir le numéro du jour : ");
   scanf("%d",&jour);

   switch(jour){
      case 1 : printf("Lundi");
      break;
      case 2 : printf("Mardi");
      break;
      case 3 : printf("Mercredi");
      break;
      case 4 : printf("Jeudi");
      break;
      case 5 : printf("Vendredi");
      break;
      case 6 : printf("Samedi");
      break;
      case 7 : printf("Dimanche");
      break;
      default: printf("jour invalide");
      break;
   }

   return 0;
}
```

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

int main(void){
   int a,b;
   char op;

   printf("saisir une valeur pour a");
   scanf("%d",&a);

   printf("saisir une valeur pour b");
   scanf("%d",&b);

   printf("saisir l'opérateur :");
  op=getch();

   switch(op){
```

```
        case '+' : printf("%d + %d = %d",a,b,(a+b));
        break;
        case '-' : printf("%d - %d = %d",a,b,(a-b));
        break;
        case '*' : printf("%d * %d = %d",a,b,(a*b));
        break;
        case '/' : printf("%d / %d = %d",a,b,(a/b));
        break;
        default: printf("opérateur invalide");
        break;
    }

    return 0;
}
```

7.3 L'instruction for

L'instruction for est une instruction de boucle qui permet d'exécuter plusieurs fois un même bloc d'instructions. Le nombre d'exécutions successives du bloc d´dépend d'un compteur, qui sera modifié après chaque exécution du bloc et testé avant chaque exécution de ce bloc. Sa syntaxe est :

for (initialisation du compteur; condition pour continuer; modification du compteur) {

liste d'instructions;

}

Exemple

```
#include<iostream>
int main() {
int i,n;
std ::cout<<"donnez lavaleur de n \n";
std ::cin>>n;
for (i = 0 ; i < 10 ; i ++)
{
n*=i;
std ::cout<<"la valeur de ndans l'itération"<<i<<"="<<n<<"\n";
}
std ::cout<<"valeur de n apres la boucle="<<n<<"\n";
return 0 ;
}
```

7.4L'instruction while

L'instruction while correspondant à un tant que .... C'est une instruction de boucle conditionnelle dont la syntaxe est la suivante :

while (condition) { liste d'instructions; }

Exemple

```
#include <iostream>
using namespace std;
```

```
int main()
{int i;
cout <<"Tapez une valeur entre 0 et 20 bornes incluses : ";
cin>>i;
while (i<0|| i>20)
 {cout <<"ERREUR ! ";
   cout <<"Tapez une valeur entre 0 et 20 bornes incluses : ";
  cin >> i;
}
return 0;
}
```

La directive ''using namespace std'' :

Vous pourriez écrire du programme en C++ qui a une fonction par exemple appelée *pgm ( )* et il y a une autre bibliothèque disponible qui a également le même fonction *pgm ( )*. Maintenant, le compilateur n'a aucun moyen de savoir à quelle version de la fonction *pgm ( )* vous faites référence dans votre programme, c'est pour cela qu'on écrit au début du programme ''using namespace std'' pour faire la distinction entre les noms des fonctions utilisées (celles que vous définissez), les noms de classe objets etc… avec les mêmes noms disponibles dans différentes bibliothèques.

L'instruction using namespace signifie simplement que dans la portée où elle est présente, rendez toutes les choses sous l'espace de noms std disponibles sans avoir à préfixer std :: avant chacune d'elles.

8.Procédures et fonctions :

En C++, les procédures et les fonctions sont des éléments fondamentaux de la programmation. Les procédures et les fonctions sont des blocs de code qui peuvent être réutilisés pour effectuer des tâches spécifiques. La principale différence entre les procédures et les fonctions réside dans le fait qu'une fonction retourne une valeur, tandis qu'une procédure n'en retourne pas.

8.1 Procédures :

Une procédure est une fonction qui ne retourne pas de valeur. Elle est déclarée avec le type de retour void. Voici un exemple de procédure simple :

```
#include <iostream>

// Définition d'une procédure

void afficherMessage() {

   std::cout << "Bonjour, le monde !" << std::endl;

}

int main() {

   // Appel de la procédure

   afficherMessage();

   return 0;}
```

8.2 Fonctions :

Une fonction est une procédure qui retourne une valeur. Vous spécifiez le type de retour dans la déclaration de la fonction. Voici un exemple de fonction simple :

```
#include <iostream>

// Définition d'une fonction

int additionner(int a, int b) {

   return a + b;

}

int main() {

   // Appel de la fonction

   int resultat = additionner(3, 5);

   std::cout << "3 + 5 = " << resultat << std::endl;

   return 0;

}
```

N'oubliez pas que les fonctions peuvent avoir différents types de retour (int, double, char, etc.) en fonction du type de valeur qu'elles doivent retourner.

Remarque :

Lorsque vous passez un paramètre par valeur à une fonction, vous envoyez une copie de la valeur à la fonction. Toute modification apportée au paramètre dans la fonction n'affectera pas la valeur d'origine. C'est le comportement par défaut en C++ pour les types de données de base.

Exercice1 :

Exécuter la séquence programme suivante s'il n y a pas d'erreurs et affiché le résultat final pour chaque variable :

```
#include <math.h>
#include <iostream>
using namespace std;
int main( )
{int x=45 ;
cout<<x; int y=x+10 ;
cout<<x<<y;
 if (x!=0)
 {int z=y/x ;
cout<<x<<y<<z ;
 float exp=z--+x*y ;
cout<<x<<y<<z <<exp;
float f=10*sqrt(z+4)+x++-y/(++z) ;
cout<<x<<y<<z <<exp<<f;}
return 0 ;}
```

	x	y	z	exp	f
x=45	45				
y=x+10 ;	45	55			
z=y/x ;	45	55	1		
exp=z--+x*y ;	45	55	0	2476	
f=10*sqrt(z+4)+x ++-y/(++z) ;	46	55	1	2476.	10.

Exercice2 :

Soient les expressions arithmétiques suivantes :

F1 = + + sqrt(x)– x; F2 = x –(– – y x)+ +; F3 = y--y; F4 = (–y* x + +)+ x;*

Evaluer en C++ les expressions pour *x=25* et *y=10*, dans le cas ou une expression est fausse affecter la valeur 0 aux variables ; sinon afficher les résultats de *Fi(i=1,2..,5, x, y*

Expression *i*	Variable *x*	Variable *y*	Variable *Fi*
i=1	0	0	0
i=2	0	0	0
i=3	0	0	0
i=4	26	10	-224

Exercice3

Programme pour calculer la somme de deux matrices :

a)

```
#include<stdio.h>
int main()
{
  int mat1[10][10], mat2[10][10],result[10][10];
  int i,j,row,col;

  printf("Combien de lignes et de colonnes?\n");
  scanf("%d%d",&row,&col);

  printf("\nEntrez la première matrice:\n");
  for(i=0; i < row; ++i)
    for(j = 0; j < col; ++j)
      scanf("%d",&mat1[i][j]);

  printf("\nEntrez la deuxième matrice:\n");
  for(i = 0; i < row; ++i)
    for(j = 0; j < col; ++j)
      scanf("%d",&mat2[i][j]);

  printf("\nMatrice après l'addition:\n");
  for(i = 0; i < row; ++i)
  {
    for(j=0; j < col; ++j)
    {
```

```
      result[i][j] = mat1[i][j] + mat2[i][j];
      printf("%d ",result[i][j]);
    }
    printf("\n");
  }
 return 0;
}
```

b)

```
#include<iostream>
#include <string>

using namespace std;

class Matrix {
private:
   int rows, cols;

public:
   void getSize() {
      cout << "Enter the number of rows and columns: ";
      cin >> rows >> cols;
   }

   void inputMatrix(int m[10][10],string name) {
      cout << "Enter the elements for " << name << " matrix:" << endl;
      for (int i = 0; i < rows; ++i) {
         for (int j = 0; j < cols; ++j) {
            cout << name<<" matrix" << "[" << i + 1 << "][" << j + 1 << "]: ";
            cin >> m[i][j];
         }
      }
   }

   void addMatrices(int m1[10][10],int m2[10][10],int r[10][10]) {
      for (int i = 0; i < rows; ++i) {
         for (int j = 0; j < cols; ++j) {
            r[i][j] = m1[i][j] + m2[i][j];
         }
      }
   }

   void displayResult(int m[10][10],string name) {
      cout <<name<<" matrix"<< endl;
      for (int i = 0; i < rows; ++i) {
         for (int j = 0; j < cols; ++j) {
            cout << m[i][j] << "  ";
         }
         cout << endl;
      }
   }
};

int main() {
                int mat1[10][10], mat2[10][10],result[10][10];
   Matrix matrix;
```

```
    matrix.getSize();
    matrix.inputMatrix(mat1, "first");
    matrix.inputMatrix(mat2, "second");
    matrix.addMatrices(mat1,mat2,result);
    matrix.displayResult(mat1,"first");
    matrix.displayResult(mat2,"second");
    matrix.displayResult(result,"after addition");
    return 0;
}
```