MA 253: Data Structures Lab with OOP

Tutorial 1

http://www.iitg.ernet.in/psm/indexing_ma253/y13/index.html

Partha Sarathi Mandal

psm@iitg.ernet.in

Dept. of Mathematics, IIT Guwahati

Reference Books

- Cormen, Leiserson and Rivest, **Introduction to Algorithms**, Prentice Hall-India.
- A. H. Aho, J. E. Hopcroft and J. Ullman, **Data Structures and Algorithms**, Addison-Wesley.
- Bjarne Stroustrup, **The C++ Programming Language**, Addison-Wesley (Third Edition & Special Edition).
- Horowitz, Sahni, Mehta**, Fundamentals of Data Structures in C++,** Galgotia.
- Timothy A. Budd, **An Introduction to Object-Oriented Programming,** Third edition, Pearson Education Inc.
- John R Hubbard, **Programming with C++**, Schaum's OuTLines, Tata McGRAW HILL

Exams and Marking

- Quiz + Assignment + Performance in Lab: 50%
- Midsem: 20%
- Endsem: 30%

Policy

- Laboratory attendance is very important and mandatory.
- You must be punctual.
- You must attend the proper section
- There may be **pop quizzes** in the labs.
- (A *pop quiz is a quiz or test given to the class without prior warning or announcement)*

Cheating Gets You “equivalent -ve marks of the Question”

- **Specific Examples Of Cheating:**
  - If you **help** some body for **cheating** or providing your login **passwd.**
  - If you're going to copy someone's **assignment** or **copy code** during test.

Overview on C++

- Classes and objects
- Constructors and destructors
- Overloading operator
- Inheritance
- Polymorphism
- Templates
- Namespace

Class and Object

Class:

- A data structure that can hold both data and functions.
- A template for creating objects.

Object:

- Object is an instance of the class
- Object is usually used to describe variables whose type is a class.

Object-oriented programming involves programs that use classes, where objects are used to interact with one another to design applications and computer programs

Object-Oriented Programming

- How is an **object** different from a **struct ?**
  - **struct** contains just ***data***
  - **object** has ***data*** and ***functionality***
- An object is of a particular class
  - **class** is the ***data type*** of that object
  - an **object** would be a ***variable***
  - defines what data and functionality the object has
- And of course, we have the WALL OF ABSTRACTION dividing up the interface and implementation

Example code

file.cpp

```
#include<iostream>
using namespace std;
int main(){
    float n1, n2;
    cout << "Enter two numbers: ";
    cin >> n1 >> n2;
    float average = (n1+n2)/2;
    cout << "Average = " << average << endl;
    return 0;
}
```

support *Input and Output streaming: the capability to read from the keyboard (input) and write to the screen (output).*

this signals to the compiler that the code I'm writing is within the std (standard) namespace, it tells the compiler that when it sees ***cout*** it is to treat it like std::cout.

C++ now includes a group of supporting code called the **standard library (std)**, which provides **objects** to handle input and output. ***cin*** is an object that handles input from the keyboard, and ***cout*** is an object that handles output to the screen.

main() always returns an int (integer) so we should always declare main to return an integer.

```
$ g++ file.cpp
$ ./a.out
$ Enter two numbers: 3 7
$  Average = 5
```

Few important notes

- Declaration of variable
  - Can be declared right at the place of its first use.
- Reference Variable
  - Both of the variable refer to the same data object in memory

    float x = 10;

    float &y = x;
- Memory allocation (new, delete)

  int *x = new int(10); or int *x; x= new int; *x = 10; *here 10 is value*

  delete x;

  int *a = new int[10]; *create a memory space for an array of size 10*

  delete []a; *free the dynamically allocated array*
- Type Casting
  - float average = sum/ float(i);

Blocks and Local Scope

- A block is a list of statements within curly braces.
- Blocks can be put anywhere a statement can be put.
- Blocks within blocks are *nested* blocks.
- An object name is known only within the block in which it is defined and in nested blocks of that block.
- A parameter can be considered to be defined at the beginning of the block corresponding to the function body.

Local Object Manipulation

```
void f() {
    int i = 1;
    cout << i << endl;              // insert 1
    {
       int j = 10;
      cout << i << j << endl;   // insert 1 10
       i = 2;
       cout << i << j << endl    // insert 2 10
    }
    cout << i << endl;              // insert 2
    cout << j << endl;              // illegal
}
```

Local Object Manipulation

```
void f() {
    int i = 1;
    cout << i << endl;              // insert 1
    {
       int j = 10;
      cout << i << j << endl;   // insert 1 10
     int  i = 2;
       cout << i << j << endl    // insert 2 10
    }
    cout << i << endl;              // insert 1
    cout << j << endl;              // illegal
}
```

Name Reuse

- If a nested block defines an object with the same name as enclosing block, the new definition is in effect in the nested block

Example

```
void f(){
    {
        int i = 1;
        cout << i << endl;          // insert 1
        {
            cout << i << endl;      // insert 1
            char i = 'a';
            cout << i << endl;      // insert a
        }
        cout << i << endl;          // insert 1
    }
    cout << i << endl;             // illegal insert
}
```

Global Scope

- Objects not defined within a block are global objects.
- A global object can be used by any function in the file that is defined after the global object.
  - It is best to avoid programmer-defined global objects
    - Exceptions tend to be important constants.
- Global objects with appropriate declarations can even be used in other program files.
  - **`cout`**, **`cin`** are global objects that are defined in by the iostream library.
- Local objects can reuse a global object's name.
  - Unary scope operator :: can provide access to global object even if name reuse has occurred

Example

```
int i = 1;
int main() {
   cout << i << endl;                  // insert 1
   {
      char i = 'a';
      cout << i << endl;               // insert a
      ::i = 2;
      cout << i << endl;               // insert a
      cout << ::i << endl;             // insert 2
  }
  cout << i << endl;
   return 0;
}
```

Stdlib Library

- Provides in part functions for generating **pseudorandom numbers.**
  - **`rand()`**
    - Returns a uniform pseudorandom unsigned int from the inclusive interval 0 to **`RAND_MAX`**

```
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;
int main(){
   for (int i = 1; i <= 5; ++i)
      cout << rand() << endl;
   return 0;
}
```

Different Sequences

- To produce a different sequence, invoke

```
void srand(unsigned int);
```

- Consider seed.cpp

```
int main() {
   cout << "Enter a seed: ";
   unsigned int Seed;
   cin >> Seed;
   srand(Seed);
   for (int i = 1; i <= 5; ++i)
      cout << rand() << endl;
   return 0;
}
```

Different Sequences

- To automatically get a different sequence each time
  - Need a method of setting the seed to a random value.
    - The standard method is to use the computer's clock as the value of the seed.
    - The function invocation **`time()`** can be used
      - Returns an integral value of type **`time_t`**
      - Invocation **`time(0)`** returns a suitable value for generating a random sequence.

Example

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>

using namespace std;

int main() {
   srand((unsigned int) time(0));
   for (int i = 1; i <= 5; ++i)
        cout << rand() << endl;
   return 0;
}
```

Problem 1

- Deploy 100 random points **inside** a square of size 10 unit.
  - For given a circular with center and radius fine points which are (i) on perimeter of the circle, (ii) inside the circle and (iii) outside the circle.
  - Find three points inside the circle for which area of the triangle is maximum.

Problem 2

- Deploy 100 disks with random center point **inside** a square of size 10 unit with radius 1 unit.
  - Find total area which is not covered by any of the disk inside the square.