Name: Sandesh Shreshtha
Roll no: 2K21/CO/427
A6 Batch

# OOPS assignment - I

date: Nov 16, 2022

Submitted to:

Mr. Aman Kumar Pandey

# Unit - 1

Q1) What is a friend function? What are the merits and demerits of using friend functions?

=) A friend function is a function that isn't a member of a class but has access to the class's private and protected members.

Merits:
- used to access the non-public members of a class
- allows to generate more efficient code
- provides additional functionality which is not normally used by class
- can be called without usage of any object

demerits:
- is not passed down to derived classes
- cannot be static or extern because they lack storage specifier

2) Create two classes DM and DB which store the value of distances. DM stores distances in metres and cm and DB in feet and inches. WAP that can read values for the class objects ~~or DB object, display~~ and add one object of DM with another object of DB.
Use a friend function to carry out addition operation.

```
#include <iostream>
using namespace std;
class DB;
class DM{
    private:
        float metres;
        float cm;
    public:
    friend float addition(DM, DB);
     DM();
    DM(float a, float b){
            metres = a;
            cm = b;
          }
    void display(){
        cout << metres << "m" << cm << "cm" << endl;
          }
    };
class DB{
    private:
        float foot;
        float inches;
    public:
        friend float addition(DM, DB);
        DB();
        DB(float a, float b){
            foot = a;
            inches = b;
          }
        void display(){
            cout << foot << "ft" << inches << "in" << endl;
          }
    };
```

```
float addition(DM a, DB b){
    float metres = a.metres + (a.cm * 0.01) +
                   (b.foot * 0.3048) + (b.inches * 0.025);
    return metres;
}
int main(){
    DM obj1(12, 45.56);
    obj1.display();
    DB obj2(10, 55.89);
    obj2.display();
cout << "sum in metres: " << addition(obj1, obj2);
    return 0;
}
```

Q3) Unit-2

Name the operators that cannot be overloaded and write a program to overloading '+' operator and perform complex number addition.

→

We can overload almost all operators in C++ except:

- class member access operator (., .*)
- scope resolution operator (::)
- size operator (sizeof())
- condition operator (?:)

cause it creates ambiguity within the program.

Operator overloading is done with help of special function "operator".

```
#include <iostream>
using namespace std;
class complex {
    int real;
    int imag;
public:
    complex() {
        real = 0;
        imag = 0;
    }
complex(int a, int b): real(a), imag(b) { };
void display() {
    cout << real << "+" << imag << "i" << endl;
    }
complex operator+ (complex a) {
    complex result;
    result.real = real + a.real;
    result.imag = imag + a.imag;
    return result;
    }
};
int main() {
    int x = 0, y = 1;
    cout << x+y << endl;
complex c1(1,2);
complex c2(3,5);
complex c3(4,6);
complex c4 = c1 + c2 + c3;
c4.display();
return 0;
}
```

(Q 4) A template can be considered a kind of macro. Then, what is the difference between them.

| Macro | Template |
|---|---|
| → A macro is merely a string that a compiler replaces with the value that was defined. | → Templates are the blueprint or formula for creating a generic class or a function. |
| → macros are expanded by preprocessor before proper compilation. | → expanded at compile time. |
| → always expanded inline | → can be expanded inline only when compiler deems it appropriate |
| → macros don't have any type checking and therefore are type unsafe | → As they are compile time checking, templates are considered type safe. |

| function overloading | Template |
|---|---|
| → used when multiple functions do similar operations | → used when multiple functions do identical or same operation but only with different datatypes. |
| → can define multiple behaviours of functions with same name or different parameter. | → only the datatype differs in any 2 functions |

difference between class template and template class:

→ class template is used to generate template classes.

class template is used to generate template classes and we cannot define objects of class template. Template class is an instance of class template.

class template example:

```
template <class L, class T> classkey;
```

templateclass example:

```
template <char> class car{};
```

(Q5) WAP to implement multiple parameters template in C++:

```
#include <iostream>
using namespace std;
template <typename t1, typename t2 = int>
class myclass {
    t1 n1;
    t2 n2;
public:
myclass();
myclass(t1 a, t2 b){
    n1 = a;
    n2 = b;
}
void display(){
    cout << n1 << " " << n2 << endl;
}
```

```
int main(){
    myclass<int, char>(5, 'c').display();
    myclass<float, int>(5.55, 6).display();
    myclass<> c(5, 4); // default int
    c.display();
    return 0;
}
```

~~Unit - 3~~

Q 6) Difference between function overloading and overriding.

| Overloading | Overriding |
|---|---|
| → can occur without inheritance | → occurs when one class is inherited from another class. |
| → overloaded functions are in same scope. | → Overridden functions are in different scopes. |
| → overloading is used to have same name functions which behave differently | → when derived class function has to do a different job than base class function. |
| → function signature must be different | → same function signature |
| → return type may or may not be same. | → return type must be same. |
| → checked at compile time | → checked at run time. |

Q 7) differentiate between C++ and Java.

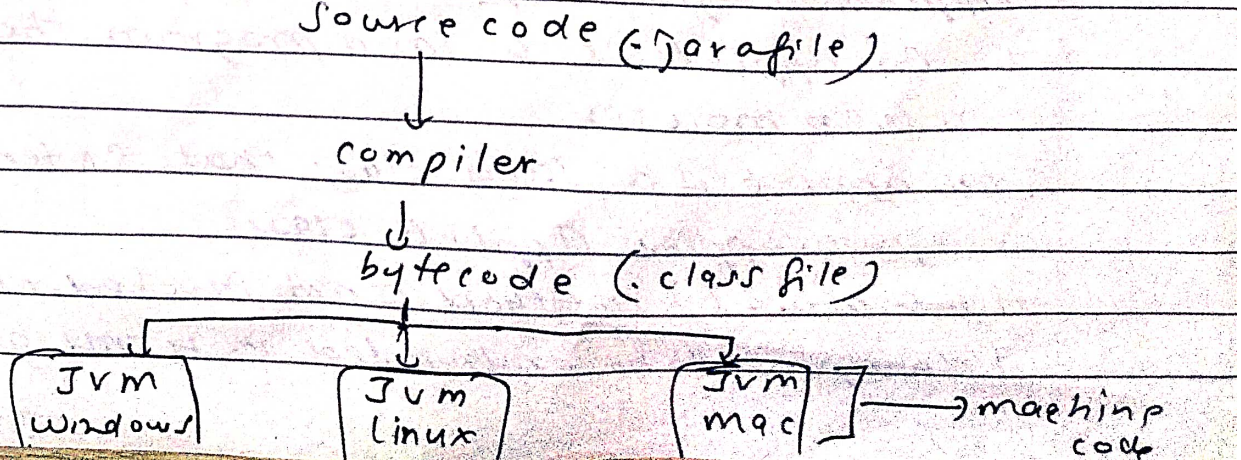
| | C++ | Java |
|---|---|---|
| platform & portability | C++ is a platform dependent pl; hence not portable | platform independent so can run in any OS; hence portable |
| Compilation | both ca it is a compiled only language. | it is both interpreted and compiled language |
| go to operator overloading | supports go to statements supports operator overloading as well as function overloading | doesn't support go to statements doesn't support operator overloading but allows function overloading |
| multiple inheritance | supports both single and multiple inheritances | supports only single inheritance multiple inheritance are achieved partially using interfaces |
| dependency | should be compiled for different platforms | platform independent since java byte code works on any operating system |
| memory mgmt | manual | memory management is system controlled |
| virtual | has virtual keyword | doesn't have virtual keyword |
| global scope | supports both global and namespace scope | supports no global scope |
| parameter passing | supports ~~only~~ pass by value as well as reference | supports only pass by value technique |
| I/O | C in for input and Cout for output | uses system.in for input and system.out for output |

Evergreen
Page No. 9
Date : / /20

| | C++ | Java |
|---|---|---|
| Struct & union | supports Unions and structures. both procedural and Object oriented programming language. | does not support structure and unions. only object oriented programming language. |
| object mangemnt | manual object management using new and delete operators. | automatic object management with garbage collection. |
| pointers | strongly supports pointers | limited support for pointer |

Q 8) Bytecode

Java Bytecode is the instruction set for the JVM (Java Virtual machine). It acts similiar to assembler which is an alias representation of C++ code. As soon as Java program is compiled, java bytecode is generated. with help of javabytecode, we can obtain machine independency in java



advantages of Java Bytecode:

- Bytecode is essentially the ML which runs on the JVM. When a class is loaded, it gets a stream of bytecode per method of the class. Thus, bytecode makes Java a platform independent language leading to its portability.
- JVM is interpreter for Byte code.
- Java is platform independent because of Bytecode

- Bytecode vs Machine code: main difference is that machine code is a set of instructions in machine language a binary which can be directly executed by CPU while bytecode is non-runnable code generated by compiling a source code that relies on a interpreter to get executed.
  + it is compiled to run on virtual machine instead of CPU (JVM instead of CPU).

(09) Applets and Java Swing

Applet Programming:

+ An applet is a Java program that runs in a web browser
+ an applet is a Java class that extends the java.applet.Applet class.
+ ~~no~~ main() method is not invoked on an applet.
+ designed to be embedded in HTML page

a simple Hello world applet:

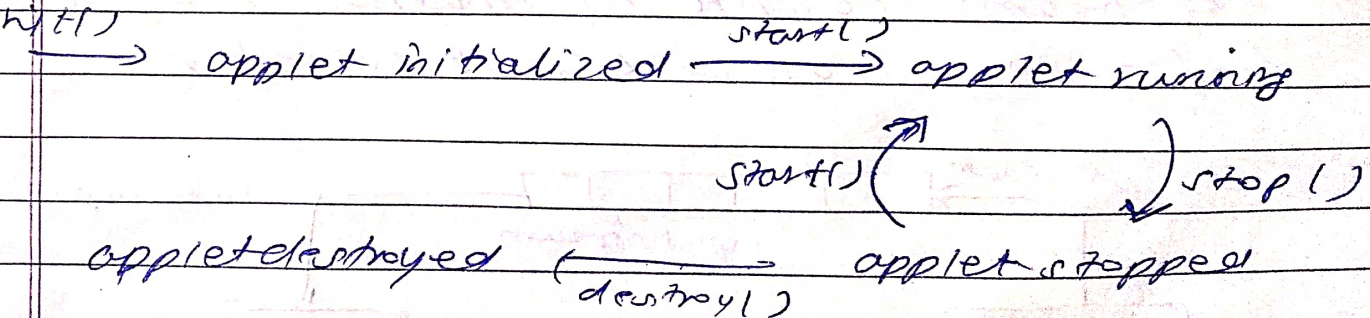
```
import java.applet.*;
import java.awt.*;

public class HelloWorldApplet extends Applet{
    public void paint (Graphics g){
        g.drawString ("Hello world", 25, 50);
    }
}
```

## ~~AWT~~ packages

advantages:

- can be executed by browsers running under any platforms
- secured and works at client side so less response time

## Lifecycle of applet



## (10) Working with files in C++

→ File is a collection of related data stored on a particular area of a disk.

→ Stream is a sequence of bytes that act as a source for input and as a destination for output.

→ inputstream: extracts data from file and provides to program
sourcestream

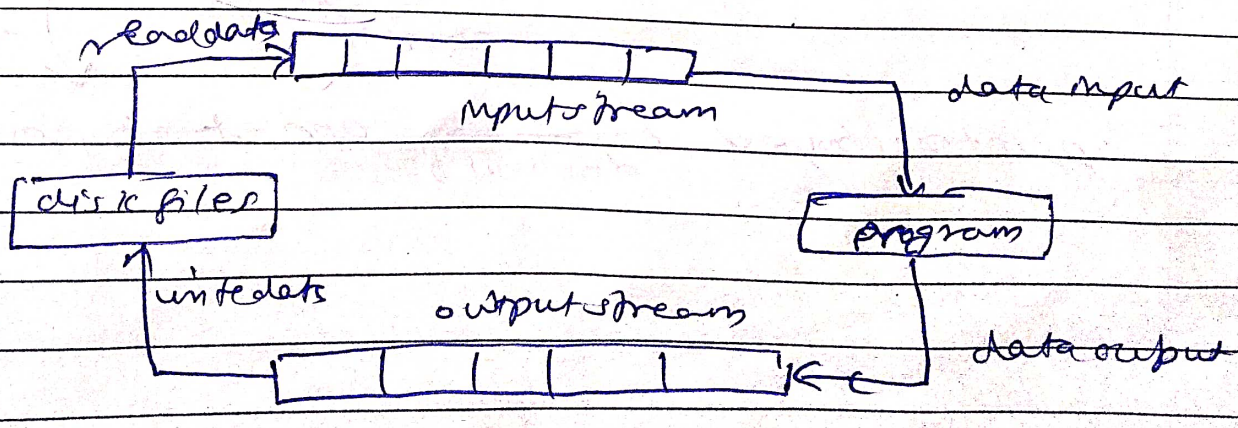
outputstream: gets data from program and writes onto the file
destinationstream

→ ~~cin~~ >> and << operators are overloaded in istream and ostream classes respectively.

→ C++ file handling mechanism to store output of a program in a file and read output from file displayed on disk.

→ 3 classes: fstream, ifstream, ofstream are available in fstream header file

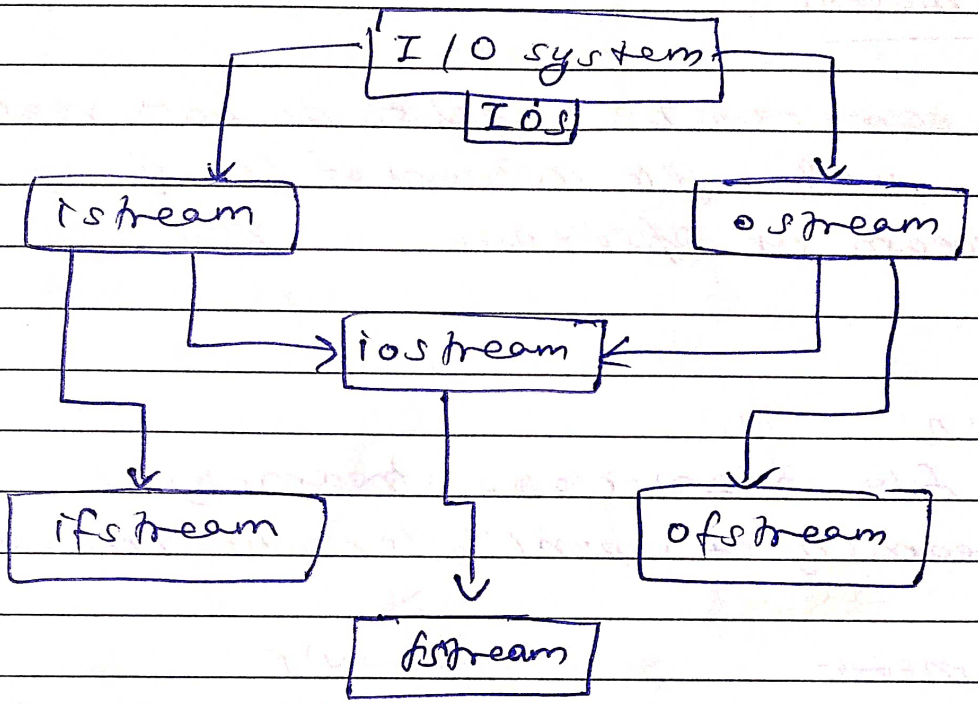
- ifstream: To read information from file
- ofstream: To write information in a file
- fstream: read and write operations




file input & output stream

- open( ) → to open a file
  syntax: stream_object.open("filename");

# Classes for file stream operations:




→ These classes are declared in fstream.h

example: opening and closing a file in c++

```
#include <fstream.h>
    void main(){
        ifstream fileI;
        fileI.open("sample.txt");
        fileI.close();}
```

```
#include <fstream.h>
    void main(){
        ofstream file2;
        file2.open("sample.txt");
    }
```

Date : / /20

Syntax

```
file-stream-class stream-object;
stream-object.open("filename");
```

Filemodes :

→ fstream can be used to do both read and write a file instead of specifying ifstream or ofstream

Syntax:

```
file-stream-class streamobject;
streamobject.open("filename", mode);
```

| Parameter | meaning |
|---|---|
| ios::in | open file read only |
| ios::out | open file write only |
| ios::app | append in end of file |

```
es:- void main()
{ fstream file1, file2;
  file1.open("sample.txt", ios::in);
  "  "  "  , ios::out);
  "  "  "  , ios::out | ios::app);
```

bitwise OR operator combine 2 or more operators

Q 11) Exception handling : The purpose of exception handling mechanism is to provide means to detect and report an "exceptional circumstance" so that appropriate action can be taken.

WAP to demonstrate multiple catch and catchall(---) statements.

```
#include <iostream>
#include <conio.h>
  using namespace std;
  int main(){
   char var = 'a';
   try {
      if(var == 'a' || var == 'A'){
        throw var;
        }
      cout << endl;
   }
   catch(int ex){
      cout << "int exception caught" << ex;
      }
   catch(float ex){
    cout << "float exception caught" << ex;
    }
   catch(...){
      cout << "This is catch All Block.";
    }
   getch();
   return 0;
   }
```

Q12) WAP to throw a class as exception.

```
#include <iostream>
using namespace std;
class myClass {
    public:
    myclass() {
        cout << "default constructor for
                myclass" << endl;
    }
};
int main() {
    int a = 10, b = 0;
    try {
        if (b == 0) {
            throw myClass();}
        cout << a/b << endl;
    }
    catch(myClass e) {
    }
    return 0;
}
```