Nov/Dec 2008

Computer Science and Engineering

CS1402 – OBJECT ORIENTED PROGRAMMING

PART – A

1. How do we achieve scalability in programming C++ when we specify array sizes by constant variables?

   We have to declare the array variables as empty size array variable. The Array size is assigned according to the number of elements given as constant.

   Ex. Int a[]={10,20,30}; // so array size 3 will be allocated.

2. Write a function header called evaluate that returns a integer and that takes as parameters integer 'x' and a pointer to function 'poly'. Fuction poly takes an integer parameter and returns an integer.

```
int evalutate(int x,int(*fn)(int))
{
        --------
        --------
        return(a);
{
int poly(int y)
{
        -----
        -----
        return(y);
}
```

3. Compare and constrast the notions of struct and class in C++.

   Struct:

   - Member as public default.
   - Cannot use inheritance and polymorphism concept
   - Functions cannot be declared and defined.

   Class:

   - Member are private, protected and public
   - Inheritance and polymorphism concept used
   - Functions can be declared and defined.

4. what is dynamic binding? Give C++ syntax to achive it

   Dynamic binding means that the code associates with given procedure call is not known until achieved by is called at run time. This can be achieved by operator overloading and function overloading.

   Syntax:

```
Return-type function name(parameters)
{
    //statements;
}
```

5. what are user defined stream manipulators? Explain with example.

   The user can design their own manipulators to control the appearance of the output depending upon their use and need.

   Syntax:

```
Ostream & manipulator(Ostream & output,argument if)
{
    ----//manipulation code
    return output;
}
```

6. What are the different ways function terminate can be called?

   Using return statement

7. what is a local, member and class variable in java?

   Local:

   Local variable is a variable declared with in a mainfunction.

   Member:

   Member variable is a variable declared with in class.

   Class:

   Object is a class variable.

8. what does it mean that a method or variable is static in java?
   - Declare a method as static the memory storage can be allocated as whole, not to every time we are calling.
   - The keyword static allows main() to be called with out having to instantiate a particular instance of the class.

9. what are the different identifier states of a thread?
    - New born state
    - Runnable state
    - Running state
    - Dead state

10. what is final, finalize() and finally?

Final:

The value of a final variable can never be changed

Finalize():

It is simply finalized and can be added to any class.

- Explicitly define the tasks to be perform
- Actions are performed before an object destroyed.

Finally:

- Executed offer try/catch block has completed
- Executed whether (or) not an exception is thrown

PART B

11.(a)(i) Write a C++ program that inputs a series of 10 numbers and determines and prints the largest of the numbers

```
#include<stdio.h>
int main()
{
    int a[10],I,large;
    cout<<"enter 10 numbers";
    for(i=0;i<10;i++)
        cin>>a[i[;
    large=a[10];
    for(i=1;i<10;i++)
        if(large<a[i])
            large=a[i];
    cout<<"large no is "<<large;
    return 0;
}
```

(ii) Write a c++ program that inputs a line of text, tokentizes the line with function strtok() and outputs the tokens in the reverse order.

Program:

```
#include<iostream.h>
#include<cstring.h>
int main()
{
char sentence[]=" this is a sentence";
char *tp;
cout<<"string to be tokenized "<<sentence<<"the tokens are:";
tp=strtok(sentence ,"");
while(tp!=NULL)
{
cout<<tp<<"\n";
tp=strtok(NULL,"");
}
cout<<"after srtok,sentence=",<<sentence;
return 0;
}
```

11(b)(i) Write a program in C++ that contains a function which takes an integer array as argument and returns the smallest element of the array

```
#include<iostream.h>
int main()
{
        int I,n,m,a[100];
int small(int[],int);
cout<<"enter the no";
cin>>n;
cout<<"enter "<<n<<"element";
for(i=0;i<n;i++)
        cin>>a[i];
m=small(a,n);
cout<<"smallest number"<<m;
```

```
return 0;
}
int small(int x[],int y)
{
int j,min;
min=x[0[;
for(j=1;j<y;j++)
if(min>x[j]
min=x[j];
return(min);
}
```

(ii) Write a program using C++ pointers and strings to check if the string is palindrome or not.

```
#include<iostream.h>
int main()
{
char str1[20],str2[20];
cout<<"enter string";
cin.getline(str1,20);
strcpy(str2,str1);
strrev(str1);
if(!strcmp(str1,str2))
    cout<<"palindrome";
else
    cout<<"not palindrome";
}
```

12(a)(i)Explain the usage of friend keyword with an example program

Friend keyword

- It is required to allow functions outside a class to access and manipulate the private members of a class.
- The function declaration must be prefixed by the keyword friend
- The functions that are declared with the keyword friend are called friend function.

Characteristics

- Scope of a friend function is not limited to the class in which it has been declared as friend.
- It cannot access the class members directly.
- However it can use the object and the dot operator with each member name to access both private and public members.

Syntax:

```
Friend returntype operator operatorsymbol(arg1[arg2])
{
//body of the operator;
}
```

program:

```
#include<iostream.h>
class sample
{
private:
int s1,s2,s3;
public:
friend int totalmarks(sample t)
void getmarks();
};
void sample::getmarks()
{
cout<<"enter 3 marks";
cin>>s1>>s2>>s3;
}
int totalmarks(sample t)
{
return(t.s1+t.s2+t.s3);
}
int main()
{
sample e;
e.getmarks();
cout<<"total marks"<<totalmarks(e);
return 0'
}
```

(ii) Write a C++ program using operator overloading concept to add two complex numbers, subtract two complex numbers and to print the complex numbers as real and imaginary part.

```
#include<iostram.h>
class complex
{
private:
float real;
float imag;
public:
complex()
{
real=imag=0.0;
}
void getdata()
{
cin>>real>>imag;
}
void outdata()
{
cout<<real<<"+j"<<imag<<"\n";
}
complex operator+(complex c)
{
complex t;
t.real=real+c.real;
t.imag=imag+c.imag;
return t;
}
complex operator – complex c)
{
complex t;
t.real=real-c.real;
t.imag=imag-c.imag;
return t;
}
};
void main()
```

```
{
complex c1,c2,c3;
c1.getdata();
c2.getdata();
c3=c1+c2;
c3.outdata();
c3=c1-c2;
c3.outdata();
}
```

12(b)(i) Explain with an example program the usage of Virtual function.

A virtual function is nothing but a function that is declared in base class and redefined by a derived class.

When we use the same function name in both base and derived class the function in base class is declared by using the keyword virtual.

When a function is made virtual, C++ determine which function is to use at runtime based on the type of object pointer to by the base pointer rather than type of the pointer.

Syntax:

```
Virtual returntype functionname(arguments)
{
-------
------
}
```

program:

```
#include<iostream.h>
class sample
{
public:
virtual void show()
{
cout<<"hello";
}
{;
class sample1:public sample
{
public:
void show()
```

```
{
cout<<”welcome”;
}
};
void main()
{
sample *s,x;
sample1 y;
cout<<”display baseclass”;
s=&x;
s->show();
cout<<”display derived class”;
s=&y;
s->show();
getch();
}
```

12.(b)(ii) Write an inheritance hierarchy for class Quadrilateral, trapezoid, parallelogram, rectangle and square. Use quadrilateral as the base class of the hierarchy. Make the hierarchy as deep as possible. The private data of quadrilateral should be the (X,Y) co-ordinate pairs for the four endpoints of the quadrilateral. It also contains a member function to calculate the perimeter.

Write a driver program that instantiates and displays objects of each of these classes.

Inheritance Hierarchy:

```
#include<iostram.h>
#include<conio.h>
#include<math.h>
class quadrilateral
{
private:
int x1,x2,x3,x4,y1,y2,y3,y4;
float a,b,c,d,p;
public:
void perimeter()
{
cout<<”enter the value”;
cin>>x1>>x2>>x3>>x4>>y1>>y2>>y3>>y4;
a=sqrt(pow((x1-x2),2)+pow((y1-y2),2);
```

```
b=sqrt(pow((x2-x3),2)+pow((y2-y3),2);
c=sqrt(pow((x3-x4),2)+pow((y3-y4),2);
d=sqrt(pow((x4-x1),2)+pow((y4-y1),2);
p=a+b+c+d;
cout<<”perimeter is”<<p;
}
};
class rectangle:public quadrilateral
{
public:
void peri()
{
quadrilateral::perimeter();
}
};
class square:public quadrilateral
{
public:
void peri()
{
quadrilateral::perimeter();
}
};
class trapezoid:public quadrilateral
{
public:
void peri3()
{
quadrilateral::perimeter();
}
};
class parallelogram:public quadrilateral
{
public:
void peri2()
{
quadrilateral::perimeter();
```

```
}
};
void main()
{
clrscr();
rectangle r;
square s;
parallelogram p;
trapezoid t;
r.peri();
s.peri1();
p.peri2();
t.peri3();
getch();
}
```

13.(a)(i)Write a program using get and getline member functions to explain stream input.

Get():

- The classes istream and ostream define 2 member function get() & put() respectively
- To handle the single character input/output operator
- There are two types of get() function
    - Get(char *)
    - Get(void)

Syntax:

```
Char c;
C=cin.get();
```

Program:

```
#include<iostream.h>
int main()
{
int count=0;
char c;
cin.get();
while(c!='\n')
{
cout.put©;
count++;
```

```
cin.getr©;
}
count<<”no. of char =”<<count;
}
```

Getline():

It reads a whole line of text that ends with a new line character.

This function can be invoked by using the object cin, cin.getline(line,size);

This function call invokes the function getline() which reads character input into the variable line .

The reading is terminated as soon as either the newline character ‘\n’ is encountered (or) size-1 characters are read.

Example:

```
#include<iostream.h>
int main()
{
int s=20;
char c[20];
cout<<”enter city name”;
cin>>c;
cout<<c;
cout<<”enter city name again”;
cin.getline(c,s);
cout<<c;
return 0;
}
```

(ii)Write a program using sequential file access to maintain and access employee payroll.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
#include<fstream.h>
class employee
{
char name[10],dept;
int empid;
public:
void readdata();
void writedata();
```

```
};
void employee::readdata()
{
cout<<”enter the name”;
cin>>name;
cout<<”enter the department”;
cin>>dept;
cout<<”enter the empid”;
cin>>empid;
}
void employee::writedata()
{
cout<<setiosflags(ios::left);
cout<<setw(10)<<name;
cout<<setiosflags(ios::right);
cout<<setw(10)<<dept;
cout<<setprecision(2);
cout<<setw[10]<<empid;
}
void main()
{
employee e3];
fstream f1;
f1.open(“emp.dat”,ios::in|ios::out);
cout<<”enter the employee details”;
for(i=0;i<3;i++)
e[i].readdata();
f1.write((char *)&e[i],sizeof(e[i]]);
}
f1.seekg(0);
cout<<”content in emp.dat”;
for(i=0;i<3;i++)
f1.read((char *)&e[i],sizeof(e[i])),e[i].writedata[];
}
f1.close();
getch();
}
```

(b)(i)Explain the advantages in using a random access file. With an example program

Advantages of random access file:

Updating is a routine task in the maintenance of any data file. The updating would include one (or) more of the following task.

- Displaying the contents of a file
- Modifying an existing file
- Adding a new item
- Deleting an existing item

The actions require the file points to move to a particular location

It can be implemented if the file contains a collection of items/objects of equal length

In organization of C++ the random file organization of accessing a particular record in a file

Seekg()-move the file either backward (or)forward

Seekp()- moves the associated files current put pointer offset no. of characters.

Tellg()- function used to get the current position of input stream

Tellp()- function used to get the current position of output stream.

Program:

```
#include<iostream.h>
#include<fstream.h>
void main()
{
long curpos,endpos;
char fname[20];
cout<<"Enter file name";
cin.getline(fname,20);
fflush(stdin);
ifstream infile(fname,iios::in|iso::binary;
curpos=infile.tellg();
cout<<"current position<<curpos;
infile.seekg(0,ios::end);
endpos=infile.tellg();
cout<<"end position"<<endpos;
infile.close();
cout<<"size of "<<fname;
cout<<"is ("<<endpos<<"-"<<curpos<<")bytes";
}
```

(ii) Explain how to rethrow an exception with an example code.

Rethrow an exception:

- A handle may decide to rethrow the exception caught with out processing it.
- Using throw keyword without any arguments we can rethrow.

Program:

```
#include<iostream.h>
void divide(double x,double y)
{
cout<<"inside function";
try
{
if(y==0.0)
throw y;
else
cout<<"division"<<x/y<<"\n";
}
catch(double)
{
throw;
}
void main()
{
try
{
divide(10.5,2.0);
divide(20.0,0.6);
}
catch(double)
{
cout<<"the value is";
}
}
```

14.(a).(i) Write a java program to do the following:

(1) To output the question "who is the inventor of C++?"

(2) To accept an answer

(3) To print "Good" if answer is correct

(4) To print 'try again' if answer is wrong

(5) To print correct answer when answer is wrong at the third attempt and stop

Program to perform Quiz master:

```
Import java.io.*;
Class test
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputstreamReader(System.in));
String srt="";
System.out.println("who is the invertor of C++);
int i=0;
do
{
str=br.readlinte();
if(str.equals("Bjarne strourtrup");
{
System.out.println("correct answer…Good");
Break;
}
else
{
System.out.println("Invalid answer …. Try again");
}
i++;
}while(i<3);
if(i==3)
System.out.println("correct answer is Bjarne strourtrup");
}
}
```

(ii) Explain with an example program to demonstrate interface and its implementation

Interfaces are used as "super classes" whole properties are inherited by classes. It is therefore, to create a class that inheit the given interface.

```
Class classname implements interfacename
{
//body of classname;
```

}

- When a class implements more than one interace, they are separated by a comma
- Any number of dissimilar classes can implement a interface.
- To implement the methods, we need to refer to the class objects as of the interface rather the types of their respective classes.

Program:

```
Import java.io.*;
Interface area
{
find static float pi=3.14;
float computer(float x,float y);
}
class rectangle implements area
{
public float compute(float x,float y)
{
return(x*y);
}
}
class circle implements area
{
public float compute(float x,float y)
{
return(pi*x*x);
}
}
class interfacetest
{
public static void main(String args[])
{
rectangle r=new rectangle();
circle c=new circle();
area a;
a=r;
System.out.println("Area of  rectangle="+a.compute(10,20));
A=c;
System.out.println("Area of circle="+a.compute(10,10));
```

```
}
}
```

(b)(i) Admission to a professional course is subject to the following conditions:

Marks in maths>=60

Marks in physics>=50

Marks in chemistry>=40

Total in all 3 subjects >=200 (or) Total in maths and physics>=150. Give an marks in 3 subjects, write a program to process the applications to list the eligible candidates.

Program:

```
Import java.io.*;
Class mark
{
void admission()
{
try
{
DataInputStream din=new  DataInputStream(System.in);
System.out.println("Enter the no. of appln:");
int n=Integer.parseInt(din.readLine());
for(int i=0;i<n;i++)
{
System.out.println("enter the math mark");
int m=Integer.parseInt(din.readLine());
System.out.println("enter the physics mark");
int p=Integer.parseInt(din.readLine());
System.out.println("enter the Chemistry mark");
int  c=Integer.parseInt(din.readLine());
if(m>=60&&p>=50&&c>=40)
{
int t=m+p+c;
int t1=m+p;
if(t>=200||t1>=150)
System.out.println("Eligible");
else
System.out.println("Not eligible");
}
```

```
catch(IOException e)
{
System.out.println(e);
}
}
}
class ex
{
public static void main(String args[])
{
mark m=new mark();
m.admission();
}
}
```

!4.(b)(ii) Write a program to demonstrate the difference between the two access specifiers private and public when used in class.

Program:

```
Import java.io.*;
Class test
{
int a;
public int b;
private int c;
void setc(int i)
{
c=I;
}
int getc()
{
return c;
}
}
class accesstest
{
public static void main(String args[])
{
test ob=new test();
```

```
ob.a=10;
ob.b=20;
ob.setc(100);
System.out.println("a,b and c:"+ob.a+""+ob.getc());
}
}
```

15.(a)(i) Write a program to demonstrate how to include a class or interface in a package.

Program:

```
Package p1;
Public class prot
{
int n=1;
private int np=2;
protected int n_p1=3;
public int npu=4;
public prot()
{
System.out.println("base constructor");
System.out.println("n="+n);
System.out.println("np="+np);
System.out.println("n-p1="+n-p1);
System.out.println("npu="+npu);
}
}
package p1;
class derived extends protection
{
derived()
{
System.out.println("derived constructor");
System.out.println("n="+n);
System.out.println("n-p1="+n-p1);
System.out.println("npu="+npu);
}
}
```

Demo

```
package p1;
public class demo
{
public static void main(String args[])
{
prot p1=new prot();
derived d1=new derived();
spackage s1=new spackage();
}
package p1;
class spackage
{
prot p=new prot();
System.out.println("same package constructor");
System.out.println("n=+p.n);
System.out.println("n-p1="+p.n-1);
System.out.println("npu="+p.npu);
}
}
```

(ii) What is synchronization? Why is it necessary? Discuss.

Synchronization:

- When two or more threads need access to a shared resource will be used only one thread at a time
- The process by which this is achieved is called synchronization
- Key to synchronization is the concept of the monitor. A minitor is an object that is used as a mutually exclusive lock.
- Only one thread can own/monitor at a give time , When a thread acquires a lock, it is said to have entered the monitor.
- All other threads attempting to enter the locked monitor will be suspended until the first thread exist the monitor. These other threads are said to be waiting for the monitor.

Using Synchronized method:

- Synchronization is easy in java, because all objects have their own implict monitor associated with them.
- To enter an objects monitor just call a method that has been modified with the synchronized keyword.

Program:

Class callee implements runnable

```
{
string m;
call_me t;
thread t1;
public callee(call_me ta, string s)
{
t=ta;
m=s;
t1=new thread(this);
t1.start();
}
public void run()
{
synchronized(t)
{
t.call(m);
}
}
}
```

(b)(i) Explain with a program how exception handling mechanism can be used for debugging a program

Program:

```
Import java.io.*;
Class myex extends Exception
{
myex(string m)
{
rupee(string m);
}
}
class t
{
public static void main(String args[])
{
int x=5,y=1000;
try
```

```
{
System.out.println("the value of x"+x+"y");
System.out.println("calculating float");
float z=(float)x/(float)y;
if(z>0.001)
throw new myexp("no. is too small");
}
catch(myexp e)
{
System.out.println(e.getmessage());
}
finally
{
System.out.println(" I am always here");
}
}
}
```

The object e which contains the error message "no. is too small" is caught by the catch block which then displays the message using the getmessage() method.

(ii) Describe the major tasks of input and output stream classes.

Input and output stream classes:

- Java programs perform input an output through streams. A stream is an abstraction that either produces or consumer information
- A stream is linked to a physical device by the java input an output system. The same input and output classes and methods can be applied to any type of device
- Java implements streams within class hierarchies defined in the java.io.package
- Java defines 2 types of streams, they are:
    - Byte stream
    - Character stream

Byte stream

1. It provides a convenient means for handling input and output of bytes.
2. Byte stream are used for reading and writing binary data
3. Input stream and output stream are designed for byte stream
4. A byte stream can be with any type of objects, including binary data.
5. The byte stream classes are,
    - File Input Stream/File Output Stream
    - Byte Array Input Stream/byte Array Output Stream

- Buffered Input Stream/Buffered Output Stream
- Push back Input Stream/Push back Output Stream
- Sequence Input Stream/Sequence Output Stream

Character streams

- ❖ It provides a convenient means for handling input and output of characters.
- ❖ They use Unicode, so can be internationalized character streams are more efficient than byte stream.
- ❖ Reader and writer are designed for character streams
- ❖ File Reader/File Writer
- ❖ Char Array Reader/Char Array Writer
- ❖ Buffered Reader/Buffered Writer
- ❖ Print Writer.