S.E. (Computer Engg.) (Second Semester) Examination 2017
PRINCIPLES OF PROGRAMMING LANGUAGES
(2015 Pattern)
Nov / Dec 2017

Time : 2 Hours **Maximum Marks : 50**

Q.1 a) List the Programming paradigms. For any three state which programming languages are based on them and how ?. [6]

Ans

Programming paradigms are a way to classify programming languages based on their features. Languages can be classified into multiple paradigms.

1. Procedural Programming Language Construct used: procedures
2. Functional Programming Language Construct used: functions
3. Abstract Data Type Programming Language Construct used: definition of abstract data types
4. Module Based Programming Language Construct used: modularization (grouping of entities) of Variables Functions Procedures Types
5. Object oriented Programming Language Construct used: classes and objects
6. Generic Programming Language Construct used: templates in C
7. Declarative or logical Programming

Q.1 b) What are benefits of implementing built-in data types in programming languages ? State the built-in data types implemented by C++. [7]

Ans.

Benefits :

1. Hiding underlying representation
2. Correct use of variables which can be checked at compile time
3. Resolution of overloaded operators
4. Accuracy control

Built-In Data Types in C++ are:

a. booleans, i.e., truth values TRUE and FALSE, along with the set of operations defined by Boolean algebra;
b. characters, e.g., the set of ASCII characters;
c. integers, e.g., the set of 16-bit values in the range <-32768, 37767>; and
d. reals, e.g., floating point numbers with given size and precision

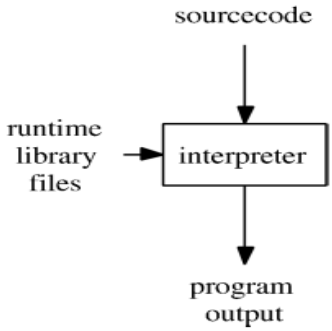
Or

Q.2 a) What is interpretation and translation process ? With neat diagram state the purpose of each activity in language processing with interpretation and translation. [6]

Ans

Interpretation Process :

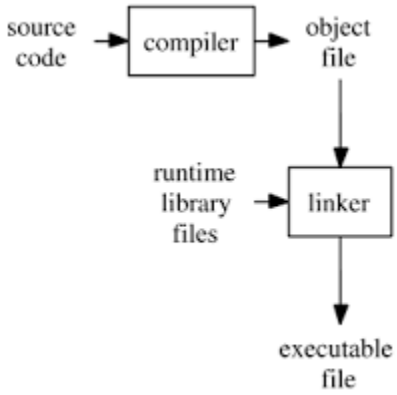
1. An interpreter converts the source code into machine language at the same time the program runs. This is illustrated below:



2. In this a special program called an interpreter converts the source code, combines with runtime libraries, and executes the resulting machine instructions all during runtime.
3. This conversion process makes the program run slower.
4. It stops/gives error when it encounters the first mistake in any line of code. This process is interpretation.

Translation Process

During the translation process the translator translates the text from the source language to target language



1. A compiler takes the program code (source code) and converts the source code to a machine language module (called an object file).

2. Another specialized program, called a linker, combines this object file with other previously compiled object files (in particular run-time modules) to create an executable file.
3. This executable file is then loaded into the memory by the loader.

Q.2 b) What are abstract data types ? How C++ implements abstract data types ? Give example.7M

Ans.

1. Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations.
2. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.
3. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.
4. It is called "abstract" because it gives an implementation independent view.

5. **In C++, classes provides great level of data abstraction**. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data, i.e., state without actually knowing how class has been implemented internally.
6. In C++, we use classes to define our own abstract data types (ADT).

Eg.

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

class Adder {
  public:
    // constructor
    Adder(int i = 0) {        total = i;       }

    // interface to outside world
    void addNum(int number) {        total += number;       }

    // interface to outside world
    int getTotal() {        return total;       };

  private:
    // hidden data from outside world
```

```
    int total;
};

int main() {
  Adder a;

  a.addNum(10);
  a.addNum(20);
  a.addNum(30);

  cout << "Total " << a.getTotal() <<endl;
  return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Total 60

Above class adds numbers together, and returns the sum. The public members - addNum and getTotal are the interfaces to the outside world and a user needs to know them to use the class. The private member total is something that the user doesn't need to know about, but is needed for the class to operate properly.

Q.3 a) What are generic data structures and generic algorithms ? How C++ implements this generic programming constructs ? Give example of each. [6M]

Ans.

Generic algorithms and Generic data structures

• Data Structures that works on any data type

C++ Implementation

C++ Implements the generic programming construct using **Templates.**

In C++, a **Standard Template Library (STL)** provides similar facilities. For example, an example of using generic **vector class for strings** can be defined as:

```
#include <vector>
#include <string>
using namespace std;
main()
{
 vector<string> V;
 V.push_back("10");
 V.push_back("20");
 V.push_back("30");
 V.pop_back();
```

```
 cout << "Loop by index:" << endl;
 int i;
 for(i=0; i < V.size(); i++)
 {
 cout << V[i] << endl;
 }
}
```

Generic Algortihms

1. A generic algorithm is one that can work with many possible data structures and many possible kinds of data.
2. Eg : Function Templates
3. To use any of the generic algorithms, you must first include the appropriate header file. The majority of the functions are defined in the header file algorithm.
   Eg.

```
/ sort algorithm example
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector

bool myfunction (int i,int j) { return (i<j); }

struct myclass {
  bool operator() (int i,int j) { return (i<j);}
} myobject;

int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  std::vector<int> myvector (myints, myints+8);

  // using default comparison (operator <):
  std::sort (myvector.begin(), myvector.begin()+4);

  // using function as comp
  std::sort (myvector.begin()+4, myvector.end(), myfunction);

  // using object as comp
  std::sort (myvector.begin(), myvector.end(), myobject);

  // print out content:
  std::cout << "myvector contains:";

  for (std::vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';
```

```
        return 0;
}
```

Q.3b Justify the meaning of each characteristic of Java in the statement "Java is simple, architecture neutral, portable, interpreted and robust and secured programming language". [6M]

Ans.

Simple

According to Sun, Java language is simple because:

- syntax is based on C++ (so easier for programmers to learn it after C++).
- It has removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.
- No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

Architecture Neutral

- There is no implementation dependent feature e.g. size of primitive types is fixed.
- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

Portable

- We may carry the java bytecode to any platform.

Interpreted

- The Java compiler generates byte-codes, rather than native machine code.
- To actually run a Java program, you use the Java interpreter to execute the compiled byte-codes. Java byte-codes provide an architecture-neutral object file format.
- The code is designed to transport programs efficiently to multiple platforms.

Robust

- Robust simply means strong. Java uses strong memory management.
- There are lack of pointers that avoids security problem. There is automatic garbage collection in java.
- There is exception handling and type checking mechanism in java.
  All these points makes java robust.

Secure

- Java is secured because: No explicit pointer
- Access restrictions are enforced (public, private)
- Byte codes are verified, which copes with the threat of a hostile compiler

Or

Q.4 a) What are challenges for Programming in Large ? How these are addressed by programming languages. [6M]

Ans.

1. Setting up modules that will not need altering in the event of probable changes **(Modularity)**
2. Abstraction-creating skills. **(Encapsulation)**
3. Better Correlation between physical program and logical design structure **(Interface).**

Address by Programming Languages.

Encapsulation

1. In Pascal is achieved by Procedures and Functions
2. In C by Files (Header files)
3. In C++ by Class
4. In Ada by Package

Q.4 b) Write a program in Java to perform the addition of two matrices (multidimensional arrays) and set the diagonal elements of resultant matrix to 0. [6M]

Ans

```
import java.util.Scanner;

class AddTwoMatrix
{
  public static void main(String args[])
  {
    int m, n, c, d;
    Scanner in = new Scanner(System.in);

    System.out.println("Enter the number of rows and columns of matrix");
    m = in.nextInt();
    n  = in.nextInt();

    int first[][] = new int[m][n];
    int second[][] = new int[m][n];
    int sum[][] = new int[m][n];

    System.out.println("Enter the elements of first matrix");

    for (  c = 0 ; c < m ; c++ )
```

```
      for ( d = 0 ; d < n ; d++ )
        first[c][d] = in.nextInt();

    System.out.println("Enter the elements of second matrix");

    for ( c = 0 ; c < m ; c++ )
      for ( d = 0 ; d < n ; d++ )
        second[c][d] = in.nextInt();

    for ( c = 0 ; c < m ; c++ )
      for ( d = 0 ; d < n ; d++ )
         sum[c][d] = first[c][d] + second[c][d];

    System.out.println("Sum of entered matrices:-");

    for ( c = 0 ; c < m ; c++ )
    {
      for ( d = 0 ; d < n ; d++ )
        System.out.print(sum[c][d]+"\t");
       System.out.println();
    }
  }
System.out.println("Setting the diagonal elements of resultant matrix to 0:-");

    for ( c = 0 ; c < m ; c++ )
    {
      for ( d = 0 ; d < n ; d++ )
          if(c==d)
            sum[c][d] = 0;
    }
  }
    System.out.println("Output After Setting :-");

    for ( c = 0 ; c < m ; c++ )
    {
      for ( d = 0 ; d < n ; d++ )
        System.out.print(sum[c][d]+"\t");
       System.out.println();
    }
  }
```

}

Q.5 a) Explain the concept of dynamic dispatch while overriding method in inheritance. Give example and advantages of doing so. [5M]

Ans

Method Overriding is an example of runtime polymorphism. When a parent class reference points to the child class object then the call to the overridden method is determined at runtime, because during method call which method(parent class or child class) is to be executed is determined by the type of object. This process in which call to the overridden method is resolved at runtime is known as dynamic method dispatch.

Eg.

```
class ABC{
  //Overridden method
  public void disp()
  {
        System.out.println("disp() method of parent class");
  }
}
class Demo extends ABC{
  //Overriding method
  public void disp(){
        System.out.println("disp() method of Child class");
  }
  public void newMethod(){
        System.out.println("new method of child class");
  }
  public static void main( String args[]) {
        /* When Parent class reference refers to the parent class object
         * then in this case overridden method (the method of parent class)
         *  is called.
         */
        ABC obj = new ABC();
        obj.disp();

        /* When parent class reference refers to the child class object
         * then the overriding method (method of child class) is called.
         * This is called dynamic method dispatch and runtime polymorphism
         */
        ABC obj2 = new Demo();
        obj2.disp();
```

```
  }
}
```
Output:

disp() method of parent class
disp() method of Child class
In the above example the call to the disp() method using second object (obj2) is runtime polymorphism (or dynamic method dispatch).

Advantages of Dynamic Method Dispatch

1. Dynamic method dispatch allow Java to support overriding of methods which is central for run-time polymorphism.
2. It allows a class to specify methods that will be common to all of its derivatives, while allowing subclasses to define the specific implementation of some or all of those methods.
3. It also allow subclasses to add its specific methods subclasses to define the specific implementation of some.

Q.5 b) Write a program in Java which defines Class CONVERSION which converts one unit of length into another using multiplying factor. This class has data members unit_in, unit_out and multiplier. When user creates object, constructor accepts value of multiplier and sets this for further conversion of units. The object uses methods to get value of unit_in and output value of unit_out and stores these in class variables. [8M]

Ans.

```
import java.util.Scanner;
public class CONVERSION
{
        float unit_in, unit_out, multiplier;
        public static Scanner scan;
        public CONVERSION(float multiplier_in)
        {
          multiplier = multiplier_in;
        }
        public void get_values()
        {
                 System.out.println("Enter value of unit_in:");
                unit_in = scan.nextFloat();
        }
        public void convert_units()
```

```
    {
        unit_out = unit_in * multiplier;
        System.out.println("unit_out="+unit_out);
    }

    public static void main(String args[])
    {
        scan = new Scanner(System.in);
        CONVERSION obj = new CONVERSION(3.28); //meter to feet
        obj.get_values();
        obj.convert_units();
    }
}
```

Or

Q.6 a) State two major differences in class and an interface. "Interface gives multiple inheritance facility just as in C++" justify. 7M
Ans.

Basis for Comparison	Class	Interface
Basic	A class is instantiated to create objects.	An interface can never be instantiated as the methods are unable to perform any action on invoking.
Keyword	class	interface
Access specifier	The members of a class can be private, public or protected	The members of an interface are always public.
Methods	The methods of a class are defined to perform a specific action	The methods in an interface are purely abstract.
Implement/extend	A class can implement any number of interface and can extend only one class.	An interface can extend multiple interfaces but cannot implement any interface.
Constructor	A class can have constructors to initialize the variables.	An interface can never have a constructor as there is hardly any variable to initialize.

When one class extends more than one classes then this is called multiple inheritance . For example: Class C extends class A and B then this type of inheritance

is known as multiple inheritance. Java doesn't allow multiple inheritance. In this article, we will discuss why java doesn't allow multiple inheritance and how we can use interfaces instead of classes to achieve the same purpose

Why Java doesn't support multiple inheritance?

- C++, Common lisp and few other languages supports multiple inheritance while java doesn't support it. Java doesn't allow multiple inheritance to avoid the ambiguity caused by it. One of the example of such problem is the diamond problem that occurs in multiple inheritance.
- Interface gives multiple inheritance facility just as in C++ because we can implement more than one interface in our program because that doesn't cause any ambiguity.

Eg.

```
interface X
        {
                public void myMethod();
        }
interface Y
        {
                public void myMethod();
        }
class JavaExample implements X, Y
        {
                public void myMethod()
                {
                        System.out.println("Implementing more than one interfaces");
                }
                public static void main(String args[])
                {
                         JavaExample obj = new JavaExample();
                        obj.myMethod();
                }
         }
```

Output:

Implementing more than one interfaces

Q.6 b) State the use of the following constructs in Java with example:

1. final method declaration in super class while inheritance

2. abstract class declaration.

3. method overriding [6]

Ans

1. **final method declaration in super class while inheritance**

   While method overriding is one of Java's most powerful features, there will be times when you will want to prevent it from occurring. To disallow a method from being overridden, specify final as a modifier at the start of its declaration.

   Methods declared as final cannot be overridden.

   The following fragment illustrates final:

```
class A
        {
         final void meth()
         {
           System.out.println("This is a final method.");
         }
        }
        class B extends A
        {
          void meth()
            {
                  // ERROR! Can't override. System.out.println("Illegal!");
             }
        }
```

   Because meth( ) is declared as final, it cannot be overridden in B. If you attempt to do so, a compile-time error will result. Methods declared as final can sometimes provide a performance enhancement: The compiler is free to inline calls to them because it knows they will not be overridden by a subclass.

2.abstract class declaration….explanation in above question

3. method overriding [6]…answer in above question

Q.7 a) Define the term exception, State the advantage of exception handling. What are types of exceptions? [6]

Ans

Exception in Java is an indication of some unusual event. It usually indicates the error. Advantages of Exception Handling:

1. Using exceptions the main application logic can be separated out from the code which may cause some unusual conditions.
2. When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.
3. Using exception handling, various types of errors in the source code can be grouped together.
4. Problems occurring at the lower level can be handled by the higher methods

Q.7 b) State the use of the following methods for programming applet. Give example of using each of these, init(),. Start(), paint(), stop(), destroy(), update(). (6 Marks)
Ans

・init

– This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.

・start

– This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.

・stop

– This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

・destroy

– This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet

- paint

– Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

```
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.applet.Applet;
import java.awt.Graphics;

public class ExampleEventHandling extends Applet implements MouseListener
    {
      StringBuffer strBuffer;
      public void init()
      {
          addMouseListener(this);
          strBuffer = new StringBuffer();
          addItem("initializing the apple ");
      }
      public void start()
      {
          addItem("starting the applet ");
      }
```

```
    public void stop()
    {
        addItem("stopping the applet ");
    }
    public void destroy()
    {
         addItem("unloading the applet");
    }
  void addItem(String word)
    {
        System.out.println(word);
        strBuffer.append(word);
        repaint();
    }
    public void paint(Graphics g)
  {
    // Draw a Rectangle around the applet's display area.
    g.drawRect(0, 0, getWidth() - 1, getHeight() - 1);

  // display the string inside the rectangle.
    g.drawString(strBuffer.toString(), 10, 20);
  }
  public void mouseEntered(MouseEvent event)
  { }
  public void mouseExited(MouseEvent event)
  { }
  public void mousePressed(MouseEvent event)
  { }
  public void mouseReleased(MouseEvent event)
  { }
   public void mouseClicked(MouseEvent event)
   {
    addItem("mouse clicked! ");
   }
} //class close
```

Or

8.a) What is difference between byte streams and character streams? Demonstrate the use of console class to get inputs and show results. (6 Marks)
Ans

Sr. No	Byte Stream	Character Steam
1	The byte stream is used for inputting and outputting the bytes	The character stream is used for inputting and outputting the characters
2	There are two super classes used in byte stream and those are- InputStream and OutputStream	There are two super classes used in character stream and those are- Reader and Writer class
3	A byte is a 8-bit number type that can represent values from -127 to 127. Only ASCII values can be represented with exactly 8 bits	A character is 16 bit in size that can represent Unicode
4	Data only ASCII format can be handled	Data in Unicode can be handled.

```
import  java.io.Console;

class  ReadStringTest
{
        public static void  main(String args[])
        {
                Console c=System.console();
                System.out.println("Enter your name: ");
                String n=c.readLine();
                System.out.println("Welcome "+n);
        }
}
```

8b) Write a program in Java to calculate the value of ((x + y)/(x – y)). Program should prevent the condition x – y = 0. (6 Marks)
Ans

```
class ExceptionDemo
{
        static void fun(int x, int y)
        {
                int c;
                try
                {
                        if((x-y)==0)
                                throw new ArithmeticException();
                        else
```

```
                    c = ((x + y)/(x - y));

                }
                catch(ArithmeticException e)
                {
                    System.out.println("Caught Exception: "+ e);
                }
        }
        public static void main(String args[])
        {
           fun(2,2);
        }
}//class ends
```