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

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

Q.1 a) Define the syntax and semantics. Compare and contrast the axiomatics semantics and denotational semantics. [6]

Ans

Syntax

1. Syntax is the form of its expressions, statements, and program units.
2. It refers to the well formed programs of the language
3. It can be formalized by grammar
4. Defined as a set of rules that define the combination of symbols that are considered to be correctly structured programs in that language.
5. Eg :
   (i) FORTRAN statements are one per line; modern languages are free-format
   (ii) Pascal uses semicolons between statements; C uses semicolons after statements
   (iii) Pascal uses begin…end to group statements; C uses { and }
   (iv) Pascal uses the keyword integer; C uses int

Semantics

1. Semantics is the meaning of those expressions, statements, and program units.
2. It gives formal analysis such as whether the language is strongly typed, block structured etc.
3. Describing semantics is more difficult because there is no universally accepted notation for semantics

Axiomatic Semantics	Denotational Semantics
Start with a formal specification of what the program is supposed to do, and then rigorously prove that the program does that by using a systematic series of logical steps.	Define the meaning of each type of statement that occurs in the (abstract) syntax as a state-transforming mathematical function. Thus, the meaning of a program can be expressed as a collection of functions operating on the program state
Based on formal logic (first order predicate calculus)	A technique for describing the meaning of programs in terms of mathematical

	functions on programs and program components
The idea in axiomatic semantics is to give specifications for what programs are supposed to compute	denotational models show what programs compute.
The primary use of axiomatic semantics is to prove that program fragments function according to specification – program verification. This is called program proof of correctness.	It Can be used to prove the correctness of programs
it is much less useful for language users and compiler writers	Can be an aid to language design

Q.1 b) List and Discuss the Statement level control structures and Unit-level control structures with their syntax. [7]
Ans.
Statement Level Control Structures include:

1) **Selection statement** : provides the means of choosing between two or more paths of execution .Two general categories exist :
   – Two-way selectors
   – Multiple-way selectors
2) **Iterative Statement** : The repeated execution of a statement or compound statement is accomplished either by iteration. Two common strategies:
   - Counter-controlled
   - Logically-controlled

Two-way selectors :

The general form of a two-way selector is as follows:

If *control_expression*
then *clause*
else *clause*

control_expression in C, C++ or Java ,ust be relational expression.

Multiple-way selectors :

If-elseif statement	Switch statement

```
if (expr)
{
 ...                        then clause
}
elseif (expr)
{
 ...                   elseif clause
}
else
{
 ...                        else clause
}
```

```
switch (expr) {
 case  v1:       ... case  clause
 case  v2:
        ...
 case  vn:
 default:
        ...          default clause
}
```

- The if –elseif statement can be represented by a nested if-else statement.
- A switch statement can be represented by an switch-case statement.
- The expression in a 'switch' must evaluate to an integer, and the value in a case clause must be a constant.

Iteration

- An iterative statement causes a collection of statements to be executed zero, one, or more times.
- *Pretest:* if the condition for loop termination is tested at the top of the loop (before the statements).
- *Posttest* loop: the condition for exiting the loop is tested after the statements in the loop.

Counter-controlled Loop

- Has a variable of a numeric type, called the *loop variable* in which the count value is maintained.
- Has *loop parameters* specifying the initial and terminal values and optionally a step size, of the loop variable. These values determine the number of iterations performed.
- The loop variable is not allowed to be modified inside the loop.

PASCAL:

```
            for k:= 1 to 10 do
            begin
                  i:=i-1;
                  p:=i*5;
      end
```

C :

```
for (k=1; k<=10; k++)      {      ...   }
```

Logically Controlled Loop

Logically controlled loops are much simpler than counter controlled loops, and are based on the value of a logical expression.

Pretest	Posttest

while(expr)
{
...
}
	do
	{
	...
	} while(expr);

The posttest loop body will always be executed at least once.

Unit Level Control Structures include:

break

The break statement terminates the loop (for, while and do...while loop) immediately when it is encountered. The break statement is used with decision making statement such as if...else.

continue Statement

The continue statement skips some statements inside the loop. The continue statement is used with decision making statement such as if...else.

Syntax of break statement

```
break;
```

Syntax of continue Statement

```
continue;
```

Or

Q.2 a) Explain how following concepts are used in design of data types with examples: [6]

Ans

(i) Data Aggregates and type constructors
(ii) Cartesian Product
(iii) Sequencing

(i) Data Aggregates and Type Constructors :

• Programming language perform aggregations of elementary objects. And for this , certain constructors are provided

- Such constructed objects are called compound objects
- For example: array constructor constructs the aggregates of similar type of elements
- An aggregate object has a unique name
- FORTRAN provides array constructor
- COBOL provides only record constructor

There are various type constructor as follows:

- Cartesian product
- Finite mapping
- Union
- Power set
- Sequence
- Recursion

(ii) Cartesian Product :

- Cartesian product to n sets:

$$A1 \times A2 \times ... \times An = \{ (a1, a2 , ..., an) \mid a1 \varepsilon A1, a2 \varepsilon A2, ..., an \varepsilon An \}.$$

Describe | A1x A2x ... x An | in terms of the cardinalities of the component sets.

- The Cartesian product is used to denote type constructor as follows:
  int X int -> int
- The example of Cartesian product constructor is structure in C.

Following is a example

```
struct employee
{
  int eid;
  char ename;
}e;
```

struct employee e is a type constructor denoted by Cartesian product as:
int X char -> e

(iii) Sequencing :

- A sequence consists of any number of occurrences of elements of a certain component type
- The important property of the sequencing constructor is that the number of occurrences of the component is unspecified; it therefore allows objects of arbitrary size to be represented.
- The C++ standard library provides a number of sequence implementations, including vector and list.

Q.2 b) What is importance of reliability and maintainability to programming languages? List the factors which ensure the reliability and maintainability 7M

Ans.

- A program is said to be reliable if it performs to its specifications under all conditions.
- Reliability is much important for following reasons :
  1) Maintain execution speed
  2) Possibility of crashing system such as when a user may input abnormal data into it

Maintainability is the ease with which a product can be maintained in order to:

- correct defects or their cause,
- repair or replace faulty or worn-out components without having to replace still working parts,
- prevent unexpected working condition,
- maximize a product's useful life,
- maximize efficiency, reliability, and safety,
- meet new requirements,
- make future maintenance easier, or
- cope with a changed environment.

Factors that ensure reliability are :

- Writability : Provision of adding new operations and data types easily enhances readability
- Readability : Allow algorithms to be expressed easily, Eg. Pascal is simpler than C++ but less powerful
- Simplicity : If language provides no features that make it possible to write harmful programs….. Like goto then its safe
- Safety : Ability to deal with undesired events like arithmetic overflow.
- Robustness

Factors that ensure maintainability are :

- Factoring : The language should allow programming to factor related features into one single unit.
- Locality : The effect of language feature is restricted to small local portion of the entire program otherwise if it extends to most of the program the task of making the changes can be exceedingly complex.

Q.3 a) What are *four* main programming paradigms? Which programming languages are based on these? Explain the features of any *one* of these. [6M]

Ans.

The four kinds of programming paradigms are:

1) Imperative :
    - In the imperative programming paradigm the programme comprises of a list of instructions for the computer to execute in the order in which they were written, unless stated otherwise.
    - Computations are performed through a guided sequence of steps, in which variables are referred to or changed.
    - The order of the steps is crucial, because a given step will have different consequences depending on the current values of variables when the step is executed.

- Fortran, BASIC, and C are based on this paradigm.

2) Logical :
    - The Logical Paradigm takes a declarative approach to problem-solving. Various logical assertions about a situation are made, establishing all known facts. Then queries are made. The role of the computer becomes maintaining data and logical deduction.
    - Prolog is a logical programming language, DataLog is a very clean, simple Logic Programming language , others include Mercury

3) Functional :
    - In the functional approach, the programmer defines the desired outcome
    - and cannot define the way the functions are executed. Functional programming relies on the concept of calculus (lambda-calculus), which defines a way to convert any computable (as defined in Turing's paper "On computable numbers") function to a mathematical function expressible in functional languages
    - The most important functional programming language is LISP (LISt Processing). Some more examples of functional languages are Scheme (a dialect of LISP), Haskell, and Mathematic

4) Object-Oriented:
    - The main focus of the Object-oriented paradigm is an object.
    - Object is a way of grouping data structures and functions that can be carried out on the data, which are called methods.
    - Using the proper typing of the data and methods, the programmer can achieve a high level of encapsulation.
    - The first programming language to implement features of the Object-oriented paradigm was Smalltalk.
    - Most influential Object-oriented language of today, C++, Java, Smalltalk, Delphi, C#, Perl, Python, Ruby, and PHP.

FEATURES OF OOP:

1) Object : Object is a collection of number of entities. Each object contain data and code to manipulate the data. Objects can interact without having know details of each others data or code.
2) Class: Class is a collection of objects of similar type
3) Data Hiding and Encapsulation: Combining data and functions into a single unit called class and the process is known as Encapsulation.
   Data encapsulation is important feature of a class. Class contains both data and functions. Data is not accessible from the outside world and only those function which are present in the class can access the data. The insulation of the data from direct access by the program is called data hiding or information hiding. Hiding the complexity of program is called Abstraction and only essential features are represented. In short, we can say that internal working is hidden.
4) Dynamic Binding:Refers to linking of function call with function definition is called binding and when it is take place at run time called dynamic binding.
5) Message Passing: The process by which one object can interact with other object is called message passing.
6) Inheritance: it is the process by which object of one class aquire the properties or features of objects of another class. The concept of inheritance provides the idea of reusability means we can add additional features to an existing class without modifying it.
7) Polymorphism: A greek term means ability to take more than one form. An operation may exhibite different behaviours in different instances. The behaviour depends upon the types of data used in the operation.
   Example:
   - Operator Overloading
   - Function Overloading

Q.3b Write a program which receives *n* integers. Store the integers in an array. Program outputs the number of odd and even numbers present in this array. [6M]
Ans.

```
#include <iostream>
using namespace std;
int main() {
  int input, remainder, even = 0, odd = 0;
  int Array[10];
  int evenArray[even];
  int oddArray[odd];
  cout << " Enter any 10 values  ";  // Receive 10 integers
```

```
for(int i=1;i<=10;i++)
{
cin >> input;
Array[i] = input;              //Store the 10 integers in an Array
}

for(int i=1;i<=10;i++)
{
  remainder = Array[i] % 2;
      if (remainder == 0) {
         evenArray[even] = Array[i];
         even++;
      }
      else {
         oddArray[odd] = Array[i];
         odd++;
      }
   cout << "\nThe number of evens is " << even << ".\n";    //Displaying number of evens
   cout << "The even values are: ";
   for(int i = 0; i < even; i++) {
      cout << evenArray[i] << " ";
   }
   cout << endl;
   cout << "The number of odds is " << odd << ".\n";   //Displaying number of odds
   cout << "The odd values are: ";
   for(int i = 0; i < odd; i++) {
      cout << oddArray[i] << " ";
   }
}
```

Or

Q.4 a) What are primitive data types? List the primitive data types in Java and their respective storage capacity. [6M]

Ans.

In computer science, primitive data type is either of the following :

- A basic type is a data type provided by a programming language as a basic building block. Most languages allow more complicated composite types to be recursively constructed starting from basic types.

- A built-in type is a data type for which the programming language provides built-in support.

Table 1: List of Java's primitive data types

Type	**Size in Bytes**	**Range**
byte	1 byte	-128 to 127
short	2 bytes	-32,768 to 32,767
int	4 bytes	-2,147,483,648 to 2,147,483, 647
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) *Java implements IEEE 754 standard*
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)
char	2 byte	0 to 65,536 (unsigned)
boolean	not precisely defined*	true or false

Q.4 b) Explain various methods of grouping programming units in Ada. What is advantage of grouping the units? [6M]

Ans

In Ada method to group programming units is package.

- Specification package (the interface) : consists of the interfaces provided by each of procedures, functions, variables
- Body package (implementation of the entities named in the specification : declared separately and contains all the details that are hidden from the callers.

Advantage of grouping the units is :

- It allows grouping together program components that combine to provide certain service

- Also make only relevant aspects visible to the caller

Q.5 a) What is interface in Java? How is this different than a class. Give an example of interface. [6M]
Ans

- An interface in java is a blueprint of a class. It has static constants and abstract methods.
- The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.
- Java Interface also represents IS-A relationship.
- It cannot be instantiated just like abstract class.

Difference between Interface and Class:

Sr.No	**Interface**	**Class**
1	An interface has fully abstract methods i.e. methods with nobody and final variables.	Class is a collection of fields and methods that operate on fields.
2	An interface can never be instantiated.	A class can be instantiated
3	The members of a class can be private, public or protected.	The members of an interface are always public.
4	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.
5	A class can have constructors to initialize the variables.	An interface can never have a constructor as there is hardly any variable to initialize.

Example of Interface

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Q.5 b) What do you mean by method overloading? Write a program which adds two integers and three integers by using overloaded methods for adding two and three integers respectively. [7M]

Ans

Function overloading means two or more functions can have the same name but either the number of arguments or the data type of arguments has to be different. Return type has no role because function will return a value when it is called and at compile time compiler will not be able to determine which function to call.

```
class Adder
 {
        static int add(int a,int b){return a+b;}
        static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1
 {
      public static void main(String[] args)
      {
          System.out.println(Adder.add(11,11));
         System.out.println(Adder.add(11,11,11));
      }
}
```

Or

Q.6 a) What is the use of static variables and methods in Java? Give Example of static 7M

Ans.

A file pointer is used to navigate through a file. We can control this movement of the file pointer by ourselves. The file stream class supports the following functions to manage such situations.

• seekg ():moves get pointer (input) to a specified location.
• seekp ():moves put pointer (output) to a specified location.
• tellg ():gives the current position of the get pointer.
• tellp (): gives the current position of the put pointer.

The other prototype for these functions is:

seekg(offset, refposition);
seekp(offset, refposition);

The parameter offset represents the number of bytes the file pointer is to be moved from the location specified by the parameter refposition. The refposition takes one of the following three constants defined in the ios class.

ios::beg	**start of the file**
ios::cur	**current position of the pointer**
ios::end	**end of the file**

example:

```
file.seekg(-10, ios::cur);
```

Example

```
#include<iostream>
#include<fstream>
struct record
   {
      char  code[6];
      char name[20];
      int i;
   }r;

int main()
{
   fstream file("Temp.dat",std::ios::trunc|std::ios::in|std::ios::out|std::ios::binary);
   if(!file)
   {
      std::cout<<"unable to open file";
      exit(0);
   }
 cout<<"enter character code, name and an int\n";
 cin.getline(r.code,6);
 cin.getline(r.name,20);
 cin>>r.i;
 file.write((char *)&r,sizeof(r));

 std::cout<<"\n\n"<<file.tellg()<<'\n'<<file.tellp();
 file.seekg(3);
 std::cout<<"\n\n"<<file.tellg()<<'\n'<<file.tellp();
file.seekp(5);
std::cout<<"\n\n"<<file.tellg()<<'\n'<<file.tellp();
file.seekg(3,ios::cur);
std::cout<<"\n\n"<<file.tellg()<<'\n'<<file.tellp();
file.seekp(-5,ios::end);
std::cout<<"\n\n"<<file.tellg()<<'\n'<<file.tellp();
 }
```

Q.6 b) Explain command line arguments in C++? Write program to explain the same. [6]

Ans If any input value is passed through command prompt at the time of running of program is known as command line argument. It is a concept to passing the arguments to the main() function by using command prompt.

Command line arguments application main() function will takes two arguments that is;

- argc
- argv

(i) argc: argc is an integer type variable and it holds total number of arguments which is passed into main function. It take Number of arguments in the command line including program name.

(ii) argv[]: argv[] is a char* type variable, which holds actual arguments which is passed to main function.

Command line arguments are not compile and run like normal C++ programs, these programs are compile and run on command prompt. To Compile and Link Command Line Program we need TCC Command.

(i) First open command prompt
(ii) Follow the directory where code is saved.
(iii) For compile -> C:/TC/BIN>TCC mycmd.cpp
(iv) For run -> C:/TC/BIN>mycmd 10 20

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

void main(int argc, char* argv[])
{
int i;
clrscr();
cout<<"Total number of arguments: "<<argc;
for(i=0;i< argc;i++)
{
cout<<endl<< i;<<"argument: "<<argv[i];
getch();
}
```

Output

```
C:/TC/BIN>TCC mycmd.cpp
C:/TC/BIN>mycmd 10 20
Number of Arguments: 3
0 arguments c:/tc/bin/mycmd.exe
1 arguments: 10
2 arguments: 20
```

Q.7 a) Use minimum 8 functions of vector STL. Write a program to explain the same. [7]

Ans Vectors are sequence containers which represent arrays which can change in size. Thus, we need not specify its length at the time of declaration and can change it later in the program.

1. **size()** function returns the length (i.e. number of elements in the vector).
2. **at()** function is used to access the element at specified position (index).
3. The **front()** function returns the first element of a vector.
4. **back()** function returns the last element of a vector.
5. **empty()** function checks whether a vector contains any element or not. It returns 1 if the length of a vector is 0 and 0 if it contains some element.
6. **resize()** function resizes a vector so that it contains the specified number of elements.
7. **push_back()** function adds a new element at the end of the vector (at the end of the last element of the vector), thus increasing the size of the vector by one.
8. **capacity()** function returns the storage space allocated for the vector. In other words, it returns the number of elements which can be stored in the storage space allocated for the vector. Vector capacity is always greater than or equal to the vector size.
9. **insert()** function inserts a new element in a vector before the element at the specified position.
10. **clear()** removes all elements of a vector.
11. **erase()** function removes either a single element or a range of elements from a vector.

```
#include <iostream>
#include <vector>
#include <iterator>
```

```
int main()
{
    vector<int> marks = {50, 45, 47, 65, 80};
    vector<int> vtest;
    cout << "length of array : " << marks.size() << std::endl;
    marks = {50, 47, 60};
    cout << "length of array : " << marks.size() << std::endl;
    cout << "marks[" << 2 << "] = " << marks.at(2) << endl;
    cout << marks.front() << endl;
    cout << marks.back() << endl;
   cout<<"The output following if 1 vector is empty and 0 vector is not empty"<<endl;
   cout << "Marks vector empty : " <<marks.empty() << endl;
   cout << "vtest vector empty : " <<vtest.empty() << endl;
    marks.resize(5);
    cout << "length of array : " << marks.size() << std::endl;
   marks.push_back(87);
   cout << "elements of marks" << endl;
    for(int i = 0; i < marks.size(); i++)
    {
        cout << marks[i] << endl;
    }
   marks.capacity();
   iterator it = marks.begin();
    marks.insert(it,20);
    for(int i = 0; i < marks.size(); i++)
        cout << marks[i] << endl;

     it = marks.begin();
    marks.insert(it+4, marks.begin(), marks.end());

    for(int i = 0; i < marks.size(); i++)
        cout << marks[i] << endl;

   marks.erase(marks.begin()+4);  // removing a single element at position 4
    for(int i = 0; i < marks.size(); i++)
        cout << marks[i] << endl;

   marks.erase(marks.begin()+1, marks.begin()+2);    // removing range of elements from
//position 1 till 2
marks.clear();
  return 0;
}
```

Q.7 b) What is STL? List different types of STL Containers? [6]

Ans

The C++ STL (Standard Template Library) is a powerful set of C++ template classes to provide general-purpose templatized classes and functions that implement many popular and commonly used algorithms and data structures like vectors, lists, queues, and stacks.

Core of the C++ Standard Template Library are following three well-structured components:

Component	Description
Containers	Containers are used to manage collections of objects of a certain kind. There are several different types of containers like **deque, list, vector, map etc.**
Algorithms	Algorithms act on containers. They provide the means by which you will **perform initialization**, **sorting, searching, and transforming** of the contents of containers.
Iterators	Iterators are used to step through the elements of collections of objects. These collections may be containers or subsets of containers.

The STL contains many different container classes that can be used in different situations. Generally speaking, the container classes fall into three basic categories: Sequence containers, Associative containers, and Container adapters. In Total, STL Defines **10** Containers.

- **Sequence**
  1. vector
  2. deque
  3. list
- **Associative**
  1. set
  2. multiset
  3. map
  4. multimap
- **Derived**
  1. stack
  2. queue

3. Priority_queue

Or

Q.8 a) Write a program to implement Map using STL. [6]

Ans

Here is source code of the C++ Program to demonstrate Map in Stl.

```
/*
 * C++ Program to Implement Map in Stl
 */
#include <iostream>
#include <map>
#include <string>
#include <cstdlib>
using namespace std;

int main()
{
  map<char,int> mp;
  map<char, int>::iterator it;
  int choice, item;
  char s;
  while (1)
  {
    cout<<"\n---------------------"<<endl;
    cout<<"Map Implementation in Stl"<<endl;
    cout<<"\n---------------------"<<endl;
    cout<<"1.Insert Element into the Map"<<endl;
    cout<<"2.Delete Element of the Map"<<endl;
   cout<<"3.Size of the Map"<<endl;
    cout<<"4.Find Element at a key in Map"<<endl;
    cout<<"5.Dislplay by Iterator"<<endl;
    cout<<"6.Count Elements at a specific key"<<endl;
    cout<<"7.Exit"<<endl;
    cout<<"Enter your Choice: ";
    cin>>choice;
    switch(choice)
    {
    case 1:
      cout<<"Enter value to be inserted: ";
      cin>>item;
```

```
        cout<<"Enter the key: ";
        cin>>s;
        mp.insert(pair<char,int>(s  ,item));
        break;
    case 2:
       cout<<"Enter the mapped string to be deleted: ";
        cin>>s;
        mp.erase(s);
        break;
    case 3:
      cout<<"Size of Map: ";
      cout<<mp.size()<<endl;
        break;
    case 4:
      cout<<"Enter the key at which value to be found: ";
        cin>>s;
        if (mp.count(s) != 0)
            cout<<mp.find(s)->second<<endl;
        else
            cout<<"No Element Found"<<endl;
        break;
    case 5:
      cout<<"Displaying Map by Iterator: ";
        for (it = mp.begin(); it != mp.end(); it++)
        {
            cout << (*it).first << ": " << (*it).second << endl;
        }
        break;
    case 6:
        cout<<"Enter the key at which number of values to be counted: ";
        cin>>s;
        cout<<mp.count(s)<<endl;
        break;
   case 7:
        exit(1);
      break;
    default:
        cout<<"Wrong Choice"<<endl;
    }
  }
```

```
    return 0;
  }
```

Q.8 b) What is container? List the Container classes in C++. Explain any *one* of them using program? [7]
Ans
Containers are used to manage collections of objects of a certain kind. A container class describes an object that holds other objects.There are several different types of containers like deque, list, vector, map etc.
Container Classes :

1. vector
2. deque
3. list
4. set
5. multiset
6. map
7. multimap
8. stack
9. queue
10. priority_queue

Stacks are a type of container adaptor, specifically designed to operate in a LIFO context (last-in first-out), where elements are inserted and extracted only from one end of the container.
Stacks are implemented as containers adaptors, which are classes that use an encapsulated object of a specific container class as its underlying container, providing a specific set of member functions to access its elements. Elements are pushed/popped from the "back" of the specific container, which is known as the top of the stack.
The container shall support the following operations:

- empty
- size
- back
- push_back
- pop_back
-

```
#include <iostream>
#include <stack>
```

```
#include <string>
#include <cstdlib>
using namespace std;

int main()
{
   stack<int> st;
   int choice, item;
   while (1)
   {
      cout<<"\n---------------------"<<endl;
      cout<<"Stack Implementation in Stl"<<endl;
      cout<<"\n---------------------"<<endl;
      cout<<"1.Insert Element into the Stack"<<endl;
      cout<<"2.Delete Element from the Stack"<<endl;
        cout<<"3.Size of the Stack"<<endl;
      cout<<"4.Top Element of the Stack"<<endl;
      cout<<"5.Exit"<<endl;
      cout<<"Enter your Choice: ";
      cin>>choice;
      switch(choice)
      {
      case 1:
         cout<<"Enter value to be inserted: ";
         cin>>item;
         st.push(item);
         break;
      case 2:
         item = st.top();
         st.pop();
           cout<<"Element "<<item<<" Deleted"<<endl;
         break;
      case 3:
           cout<<"Size of the Queue: ";
           cout<<st.size()<<endl;
         break;
      case 4:
           cout<<"Top Element of the Stack: ";
           cout<<st.top()<<endl;
         break;
      case 5:
         exit(1);
           break;
      default:
         cout<<"Wrong Choice"<<endl;
      }
```

```
    }
    return 0;
}
```