37. When a floating-point value is stored in an integer variable, the floating-point value is converted to an integer value by truncating its decimal value.
38. The keyword **const** is used to define variables or objects that should not be modified.
39. When writing a program that accepts input interactively it is important that the program issues prompts that clearly indicate the type and the form of the input.
40. The compound assignment operators + =, - =, * =, / =, % = perform an arithmetic operation on a variable and store the resulting value back into the variable.
41. C++ supports special operators, ++ and —, for incrementing and decrementing integral and floating-point objects.
42. A logical expression evaluates to **true** if the value of the expression is either a nonzero integer value or the **bool** value **true**.
43. A logical expression evaluates to **false** if the value of the expression is integer zero or the **bool** value **false.**
44. The relational operators also produce logical values. The relational operators fall into two categories; *equality* and *ordering*.
45. The equality operators = = and ! = and the ordering operators <, < =, >, and >= are defined for all fundamental and pointer types.
46. The **if** statement has two forms. In both forms, a logical expression is evaluated and if that expression is **true**, an action is executed. In one of the forms, an action is also specified for, when the evaluated expression is **false**.
47. The expression used to determine the course of action for a conditional or iterative construct is sometimes known as a *test expression*.
48. The **switch** statement takes actions based upon the value of an integral expression. The programmer specifies the case values of interest for that expression, and for each case value the desired action is specified.
49. Types defined by a programmer are known as user-defined or derived types.
50. The **enum** statement is a method for organizing a collection of integral constants into a type.
51. A loop is a group of statements whose actions are repeated using an iterative construct.
52. The **while** statement permits actions to be repeated while a given logical expression evaluates to **true**. If the logical expression is initially false, then the action of the construct is never executed; otherwise, the action is repeatedly executed until the test expression evaluates to **false**.
53. The **do** statement is similar in nature to the **while** statement; however, its action is always executed at least once. This is because its test expression is evaluated only after its body is executed.
54. The **for** statement is a generalization of the **while** construct that has a test expression, a one-time loop initialization section and an increment section. All sections of a **for** statement are optional. In particular, if the test expression is omitted, then the value **true** is used instead.
55. A **typedef** statement creates a new name for an existing type. Both the new and old names can be used in subsequent definitions.
56. In C++, we can declare a variable as close as possible to its first use.
57. The **class** describes all the properties of a data type, and an object is an entity created according to that description.

101. The location of a variable can be obtained using the address operator &.
102. The literal 0 can be assigned to any pointer type object. In this context, the literal 0 is known as the null address.
103. The value of the object at a given location can be obtained using the indirection operator * on the location.
104. The indirection operator produces an lvalue.
105. The null address is not a location which can be dereferenced.
106. The member selector operator -> allows a particular member of object to be dereferenced.
107. Pointer operators may be compared using the equality and relational operators.
108. The increment and decrement operators may be applied to pointer objects.
109. Pointers can be passed as reference parameters by using the indirection operator.
110. An array name is viewed by C++ as constant pointer. This fact gives us flexibility in which notation to use when accessing and modifying the values in a list.
111. Command-line parameters are communicated to programs using pointers.
112. We can define variables that are pointers to functions. Such variables are typically used as function parameters. This type of parameter enables the function that uses it to have greater flexibility in accomplishing its task.
113. Increment and decrement of pointers follow the pointer arithmetic rules. If **ptr** points to the first element of an array, then **ptr+1** points to the second element.
114. The name of an array of type **char** contains the address of the first character of the string.
115. When reading a string into a program, always use the address of the previously allocated memory. This address can be in the form of an array name or a pointer that has been initialized using **new**.
116. Structure members are **public** by default while the class members are **private** by default.
117. When accessing the class members, use the dot operator if the class identifier is the name of the class and use the arrow operator if the identifier is the pointer to the class.
118. Use **delete** only to delete the memory allocated by **new**.
119. It is a good practice to declare the size of an array as a constant using the qualifier **const**.
120. C++ supports two types of parameters, namely, value parameters and reference parameters.
121. When a parameter is passed by value, a copy of the variable is passed to the called function. Any modifications made to the parameter by the called function change the copy, not the original variable.
122. When a reference parameter is used, instead of passing a copy of the variable, a reference to the original variable is passed. Any modifications made to the parameter by the called function change the original variable.
123. When an **iostream** object is passed to a function, either an extraction or an insertion operation implicitly modifies the stream. Thus, stream objects should be passed a reference.
124. A reason to use a reference parameter is for efficiency. When a class object is passed by value, a copy of the object is passed. If the object is large, making a copy of it can

be expensive in terms of execution time and memory space. Thus objects that are large, or objects whose size is not known are often passed by reference. We can ensure that the objects are not modified by using the **const** modifier.

125. A const modifier applied to a parameter declaration indicates that the function may not change the object. If the function attempts to modify the object, the compiler will report a compilation error.
126. A reference variable must be initialized when it is declared.
127. When you are returning an address from a function, never return the address of local variable though, syntactically, this is acceptable.
128. If a function call argument does not match the type of a corresponding reference parameter, C++ creates an anonymous variable of the correct type, assigns the value of the argument to it and causes the reference parameter to refer the variable.
129. A function that returns a reference is actually an alias for the "referred-to" variable.
130. We can assign a value to a C++ function if the function returns a reference to a variable. The value is assigned to the referred-to variable.
131. C++'s default parameter mechanism provides the ability to define a function so that a parameter gets a default value if a call to the function does not give a value for that parameter.
132. Function overloading occurs when two or more function have the same name.
133. The compiler resolves overloaded function calls by calling the function whose parameters list best matches that of the call.
134. Casting expressions provide a facility to explicitly convert one type to another.
135. A cast expression is useful when the programmer wants to force the compiler to perform a particular type of operation such as floating-point division rather than integer division.
136. A cast expression is useful for converting the values that library function return to the appropriate type. This makes it clear to other programmers that the conversion was intended.
137. An *inline function* must be defined before it is called.
138. An *inline function* reduces the function call overhead. Small functions are best declared inline within a class.
139. In a multiple-file program, you can define an external variable in one and only one file. All the other files using that variable have to declare it with the keyword **extern**.
140. An abstract data type (ADT) is well-defined and complete data abstraction that uses the principle of information-hiding.
141. An ADT allows the creation and manipulation of objects in a natural manner.
142. If a function or operator can be defined such that it is not a member of the class, then do not make it a member. This practice makes a nonmember function or operator generally independent of changes to the class's implementation.
143. In C++, an abstract data type is implemented using classes, functions, and operators.
144. Constructors initialize objects of the class type. It is standard practice to endure that every object has all of its data members appropriately initialized.
145. A default constructor is a constructor that requires no parameters.

146. A copy constructor initializes a new object to be a duplicate of a previously defined source object. If a class does not define a copy constructor, the compiler automatically supplies a version.
147. A member assignment operator copies a source object to the invoking target object in an assignment statement. If a class does not define a member assignment operator, the compiler automatically supplies a version.
148. When we call a member function, it uses the data members of the object used to invoke the member function.
149. A class constructor, if defined, is called whenever a program creates an object of that class.
150. When we create constructors for a class, we must provide a default constructor to create uninitialized objects.
151. When we assign one object to another of the same class, C++ copies the contents of each data member of the source object to the corresponding member of the target object.
152. A member function operates upon the object used to invoke it, while a friend function operates upon the objects passed to it as arguments.
153. The qualifier **const** appended to function prototype indicates that the function does not modify any of the data members. A **const** member function can be used by **const** objects of the class.
154. The client interface to a class object occurs in the **public** section of the class definition.
155. Any member defined in any section — whether **public, protected,** or **private** — is accessible to all of the other members of its class.
156. Members of a **protected** section are intended to be used by a class derived from the class.
157. Data members are normally declared in a **private**. By restricting outside access to the data members in a class, it is easier to ensure the integrity and consistency of their values.
158. Members of **private** section of a class are intended to be used only by the members of that class.
159. An & in the return type for a function or operator indicates that a reference return is being performed. In a reference return, a reference to the actual object in the return expression rather than a copy is returned. The scope of the returned object should not be local to the invoked function or operator.
160. When creating a **friend** function, use the keyword friend in the prototype in the class definition, but do not use this keyword in the actual function definition. Friend functions are defined outside the class definition.
161. Friend functions have access to the private and protected members of a class.
162. An operator can be overloaded many times using distinct signatures.
163. If we want to overload a binary operator with two different types of operands with non class as the first operand, we must use a friend function to define the operator overloading.
164. Do not use implicit type conversions unless it is necessary. If they are used arbitrarily, it can cause problems for future users of the class.

165. Whenever we use **new** in a constructor to allocate memory, we should use **delete** in the corresponding destructor to free that memory.
166. The relationship "is_ a" indicates inheritance. For example, a car is a kind of vehicle.
167. The relationship "has_a" indicates containment. For example, a car has an engine. Aggregate objects are constructed using containment.
168. Both inheritance and containment facilitate software reuse.
169. A new class that is created from an existing class using the principle of inheritance is called a *derived class or subclass*. The parent class is called the *base class or superclass*.
170. When an object that is a instance of derived class is instantiated, the constructor for the base class is invoked before the body of the constructor for the derived class is invoked.
171. A class intended to be a base class usually should use **protected** instead of **private** members.
172. When a derived class object is being created, first its base classes constructors are called before its own constructor. The destructors are called in the reverse order.
173. A constructor of a derived class must pass the arguments required by its base class constructor.
174. A derived class uses the member functions of the base class unless the derived class provides a replacement function with the same name.
175. A derived class object is converted to a base class object when used as an argument to a base class member function.
176. Derived class constructors are responsible for initializing any data members added to those inherited from the base class. The base class constructors are responsible for initializing the inherited data members.
177. When passing an object as an argument to the function, we usually use a reference or a pointer argument to enable function calls within the function to use virtual member function.
178. Declare the destructor of a base class as a virtual function.
179. Destructors are called in reverse order from the constructor calls. Thus, the destructor for a derived class is called before the destructor of the base or superclass.
180. With **public** inheritance, the **public** members of the base class are public members of the derived class. The **private** members of the base class are not inherited and, therefore, not accessible in the derived class.
181. With **protected** inheritance, **public** and **protected** members of the base class become **protected** members of the derived class. The **private** members of the base class are not inherited.
182. With multiple inheritance, a derived class inherits the attributes and behaviors of all parent classes.
183. With **private** inheritance, **public** and **protected** members of the base class become **private** members of the derived class. Private members are not inherited.
184. If a derived class has a base class as a multiple ancestor (through multiple inheritance), then declare the base class as **virtual** in the derived class definition. This would ensure the inheritance of just one object of the base class.
185. A pointer to a base class can be used to access a member of the derived class, as long as that class member is inherited from the base.

234. The member function **eof** of **ios** determines if the end of the file indicator has been set. End-of-file is set after an attempted read fails.
235. To use C++ **strings**, we must include the header file **<string>** of **C++** standard library.
236. C++ strings are not null terminated.
237. Using **STL** containers can save considerable time and effort, and result in higher quality programs.
238. To use containers, we must include appropriate header files.
239. **STL** includes a large number of algorithms to perform certain standard operations on containers.
240. **STL** algorithms use iterators to perform manipulation operations on containers.
241. We may use **const**-cast operator to remove the constantness of objects.
242. We may use **mutable** specifier to the members of **const** member functions or **const** objects to make them modifiable.
243. We must restrict the use of runtime type information functions only with polymorphic types.
244. When we suspect any side-effects in the constructors, we must use **explicit** constructors.
245. We must provide parentheses to all arguments in macro functions.

Appendix G

Glossary of Important C++ and OOP Terms

#define A C++ preprocessor directive that defines a substitute text for a name.

#include A preprocessor directive that causes the named file to be inserted in place of the #include.

Abstract Class A class that serves only as a base class from which classes are derived. No objects of an abstract base class are created. A base class that contains pure virtual functions is an abstract base class.

Abstract Data Type (ADT) An abstraction that describes a set of objects in terms of an encapsulated or hidden data and operations on that data.

Abstraction The act of representing the essential features of something without including much detail.

Access Operations Operations which access the state of a variable or object but do not modify it.

Address A value that identifies a storage location in memory.

Alias Two or more variables that refer to the same data are said to be aliases of one another.

Anonymous Union An unnamed union in C++. The members can be used like ordinary variables.

ANSI C Any version of C that conforms to the specifications of the American National Standards Institute Committee X3J.

ANSI C++ Any version of C++ that conforms to the specifications of the American National Standards Institute. At the time of writing this, the standards exist only in draft form and a lot of details are still to be worked out.

Array A collection of data elements arranged to be indexed in one or more dimensions. In C++, arrays are stored in contiguous memory.

ASCII American Standard Code for Information Interchange. A code to represent characters.

Copy Constructuor The constructor that creates a new class object from an existing object of the same class.

Curly Braces One of the characters { or }. They are used in C++ to delimit groups of elements to treat them as a unit.

Data Flow Diagram (DFD) A diagram that depicts the flow of data through a system and the processes that manipulate the data.

Data Hiding A property whereby the internal data structure of an object is hidden from the rest of the program. The data can be accessed only by the functions declared within the class (of that object).

Data Member A variable that is declared in a class declaration.

Debugging The process of finding and removing errors from a program.

Decision Statement A statement that tests a condition created by a program and changes the flow of the program based on that decision.

Declaration A specification of the type and name of a variable to be used in a program.

Default Argument An argument value that is specified in a function declaration and is used if the corresponding actual argument is omitted when the function is called.

De-referencing Operator The operator that indicates access to the value pointed to by a pointer variable or an addressing expression. *See* also indirection operator.

Derived Class A class that inherits some or all of its members from another class, called **base class**.

Destructor A function that is called to deallocate memory of the objects of a class.

Directive A command to the preprocessor (as opposed to a statement to produce machine code).

Dynamic Binding The addresses of the functions are determined at run time rather than compile time. This is also known as late binding.

Dynamic Memory Allocation The means by which data objects can be created as they are needed during the program execution. Such data objects remain in existence until they are explicitly destroyed. In C++, dynamic memory allocation is accomplished with the operators **new** (for creating data objects) and **delete** (for destroying them).

Early Binding *See* static binding.

Encapsulation The mechanism by which the data and functions (manipulating this data) are bound together within an object definition.

Enumerated Data Type A data type consisting of a named set of values. The C++ compiler assigns an integer to each member of the set.

Error State For a stream, flags that determine whether an error has occurred and, if so, give some indication of its severity.

Escape Character A special character used to change the meaning of the character(s) that follow. This is represented in C++ by the backslash character '\'.

Executive File A file containing machine code that has been linked and is ready to be run on a computer.

Extensibility A feature that allows the extension of existing code. This allows the creation of new objects from the existing ones.

Extraction Operator The operator >>, which is used to read input data from keyboard.

Fast Prototyping A top-down programming technique that consists of writing the smallest portion of a specification that can be implemented that will still do something.

File A group of related records treated as a unit.

Format State For a stream, flags and parameters that determine how output values will be printed and (to a lesser extent) how input values will be read.

Free Store A pool of memory from which storage for objects is allocated. It is also known as heap.

Friend A function that has access to the private members of a class but is not itself a member of the class. An entire class can be a friend of another class.

Friend A function that although not a member of a class is able to access the private members of that class.

Function A procedure that returns a value.

Function Declaration This provides the information needed to call a function. The declaration gives the name of the function, its return type, and the type of each argument.

Function Prototype A function declaration.

Generic Class *See* parameterized class.

Generic Pointer A pointer that can point to any variable without restriction as to the type of variable. A pointer to storage without regard to content.

Global Variables Variables that are known throughout an entire program.

Header File A file containing the declarations that are to be used in one or more source files. A header file is normally included in a source file with an #include directive.

Header File *See* include file.

Heap *See* free storage.

Heap A portion of money used by new to get space for the structures and classes returned by new. Space is returned to this pool by using the delete operator.

Message Passing The philosophy that objects only interact by sending messages to each other. The messages request for some operations to be performed.

Method The means by which an object receives and responds to a particular kind of message. In C++, a method is a member function.

Multiple Inheritance A language feature that allows a derived class to have more than one base class.

New-line Character A character that causes an output device to go to the beginning of a new line.

Non Significant Digits Leading digits that do not affect the value of a number (0s for a positive number, 1s for a negative number in complement form).

Normalization The shifting of a floating-point fraction (and adjustment of the exponent) so there are no leading non significant digits in the fraction.

Null A constant of value 0 that points to nothing.

Null Pointer A pointer that does not point to any data object. In C++, the null pointer can be represented by the constant 0.

Null Character The character whose integer code is 0. The null character is used for terminating strings.

Object An entity that can store data and, send and receive messages. An instance of a class.

Object-based Systems are object-based when they allow objects to encapsulate both the data and methods and when they enforce the object identity.

Object-Oriented Object-oriented systems are object-based, and also support inheritance between classes and superclasses and allow objects to send messages to themselves.

Object-Oriented Design A method of realizing the system requirements it terms of classes, class hierchies and their interrelationships.

Object-Oriented Analysis A method of analysis in which the system requirements are identified in terms of objects and their interactions.

Object-Oriented Programming Implementation of programs using the objects in an object-oriented language like C++.

Octal Number A base-eight number.

One's Complement An operation that flips all the bits in a integer. Ones become zeros and zeros become ones.

Operator A symbol that represents an action to be performed.

Overflow Error An arithmetic error caused by the result of an arithmetic operation being greater than the space the computer provides to store the result.

Overloading A language feature that allows a function or operator to be given more than one definition. The types of the arguments with which the function or operator is called, determines which definition will be used.

Overriding The ability to change the definition of an inherited method or attribute in a subclass.

Parameter A data item to which a value may be assigned.

Parameterized Class A class definition that depends on a parameter. A family of classes can be defined by setting the parameter to different data types. It is also known as *generic class*.

Parameterized Macro A macro consisting of a template with insertion points for the introduction of parameters.

Persistence The property of objects to persist in terms of identity, state and description through time, regardless of the computer session which creates or uses them. The objects are stored and ready for use, on secondary storage.

Pointer A data type that holds the address of a location in memory.

Polymorphism A property by which we can send the same message to objects of several different classes, and each object can respond in a different way depending on its class. We can send such a message without knowing to which of the classes the objects belongs. In C++, polymorphism is implemented by means of virtual functions and dynamic binding.

Preprocessor A part of the compiler that manipulates the program text before any further compiling is done. Three important tasks of the preprocessor are (1) to replace each **#include** directive with the contents of the designated file, (2) to replace each escape sequence with the designated character, and (3) to process macro definitions and expand macro calls.

Preprocessor A program that performs preliminary processing with the purpose of expanding macro code templates to produce C++ code.

Preprocessor Directive A command to the preprocessor.

Private Base Class A base class which allows its public and protected members to be inherited as "private" members of the derived class. Thus, the inherited members are accessible to the members and friends of the derived class, but they are not accessible to the users of the derived class.

Private Member A class member that is accessible only to the member and friend functions of the class. A private member of a base class is not inherited by a derived class.

Program Header The comment block at the beginning of a program.

Programming The process of expressing the solution to a problem in a language that represents instructions for a computer.

Protected Member A protected member is the same as a private member except that a protected member of a base class is inherited by a derived class, whereas a private member is not. For details of inheritance, *see* public base class and private base class.

Pseudocode A coding technique where precise descriptions of procedures are written in easy-to-read language constructs without the bother of precise attention to the syntax rules of a computer language.

This This is a pointer to the current object. It is passed implicitly to an overloaded operator function.

Translation Creation of a new program in an alternate language logically equivalent to an existing program in a source language.

Truncation An operation on a real number whereby any fractional part is discarded.

Turbo C++ A version of the C++ language for personal computers developed by Borland.

Type Conversion A conversion of a value from one type to another.

Typecast *See* cast.

Typedef Name A name given to a type via a type-name definition introduced by the key-word **typedef.**

Union A data type that allows different data types to be assigned to the same storage location.

Value A quantity assigned to a constant.

Variable A name that refers to a value. The data represented by the variable name can, at different times during the execution of a program, assume different values.

Variable Name The symbolic name given to a section of memory used to store a variable.

Virtual Base Class A base class that has been qualified as virtual in the inheritance definition. In multiple inheritance, a derived class can inherit the members of a base class via two or more inheritance paths. If the base class is not virtual, the derived class will inherit more than one copy of the members of the base class. For a virtual base class, however, only one copy of its members will be inherited regardless of the number of inheritance paths between the base class and the derived class.

Virtual Function A function qualified by the **virtual** keyword. When a virtual function is called via a pointer, the class of the object pointed to determines which function definition will be used. Virtual functions implement polymorphism, whereby objects belonging to different classes can respond to the same message in different ways.

Visibility The ability of one object to be a server to others.

Void A data type in C++. When used as a parameter in a function call, it indicates there is no return value. void+ indicates that a generic pointer value is returned. When used in casts, it indicates that a given value is to be discarded.

Windows A graphical partition of screen for user interface.

Appendix H

C++ Proficiency Test

Part A

True / False Questions

State whether the following statements are true or false

1. A C++ program is identical to a C program with minor changes in coding
2. Bundling functions and data together is known as data hiding.
3. In C++, a function contained within a class is called a member function.
4. Object modeling depicts the real-world entities more closely than do functions.
5. In using object-oriented languages like C++, we can define our own data types.
6. When a C++ program is executed, the function that appears first in the program is executed first.
7. In a 32-bit system, the data types **float** and **long** occupy the same number of bytes.
8. In an assignment statement such as

   ```
   int x = expression;
   ```

   the value of x is always equal to the value of the expression on the right.
9. In C++, declarations can appear almost anywhere in the body of a function.
10. C++ does not permit mixing of variables of different data types in an arithmetic expression.
11. The value of the expression 13%4 is 3.
12. Assuming the value of variable x as 10, the output of the statement

    ```
    cout << x--;
    ```

    will be 10.

13. The expression **for(; ;)** is the same as a **while** loop with a test expression of **true**.
14. In C++, arithmetic operators have a lower precedence than relational operators.
15. In C++, only **int** type variables can be used as loop control variables in a **for** loop.
16. A **do** loop is executed at least once.
17. The **&&** and | | operators compare two boolean values.
18. The control variable of a **for** loop can be decremented inside the **for** statement.
19. The **break** statement is used to exit from all the nested loops.
20. The **default** case is required in the **switch** selection structure.
21. The **continue** statement inside a **for** loop transfers the control to the top of the loop.
22. The **goto** statement cannot be used to transfer the control out of a nested loop.
23. A conditional expression such as

```
(x < y) ? x : y
```

can be used anywhere a value can be.

24. A structure and a class use similar syntax.
25. Memory space for a structure member is created when the structure is declared.
26. If **item1** and **item2** are variables of type structure **Item**, then the assignment operation

```
item1 = item2;
```

is legal.

27. When calling a function, if the arguments are passed by reference, the function works with the actual variables in the calling program.
28. A structure variable cannot be passed as an argument to a function.
29. A C++ function can return multiple values to the calling function.
30. A function call of a function that returns a value can be used in an expression like any other variable.
31. We need not specify any return type for a function that does not return anything.
32. A set of functions with the same return type are called overloaded functions.
33. Only when an argument has been initialized to zero value, it is called the default argument.
34. A variable declared above all the functions in a program can be accessed only by the **main( )** function.
35. A static automatic variable retains its value even after exiting the function where it is defined.
36. We can use a function call on the left side of the equal sign when the function returns a value by reference.
37. Returning a reference to an automatic variable in a called function is a logic error.
38. Reference variables should be initialized when they are declared.
39. Using **inline** functions may reduce execution time, but may increase program size.
40. A C++ array can store values of different data types.
41. Referring to an element outside the array bounds is a syntax error.

42. When an array name is passed to a function, the function access a copy of the array passed by the program.
43. The extraction operator >> stops reading a string when a space is encountered.
44. Objects of the **string** class can be copied with the assignment operator.
45. Strings created as objects of the **string** class are zero-terminated.
46. Pointers of different types may not be assigned to one another without a cast operation.
47. Not initializing a pointer when it is declared is a syntax error.
48. Data members in a class must be declared **private**.
49. Data members of a class cannot be initialized in the class definition.
50. Members declared as **private** in a class are accessible to all the member functions of that class.
51. In a class, we cannot have more than one constructor with the same name.
52. A member function declared **const** cannot modify any of its class's member data.
53. In a class, members are private by default.
54. In a structure, members are public by default.
55. A member variable defined as **static** is visible to all classes in the program.
56. An object declared as **const** can be used only with the member functions that are also declared as **const**.
57. A member function can be declared **static**, if it does not access any non-static class members.
58. A non member function may have access to the **private** data of a class if it is declared as a **friend** of that class.
59. The precedence of an operator can be changed by overloading it.
60. Using the keyword **operator**, we can create new operators in C++.
61. We can convert a user-defined class to a basic type by using a one-argument constructor.
62. We can always treat a base-class object as a derived-class object.
63. A derived class cannot directly access the **private** members of its base class.
64. In inheritance, the base-class constructors are called in the order in which inheritance is specified in the derived class definition.
65. Inheritance is used to improve data hiding and encapsulation.
66. We can convert a base-class pointer to a derived class pointer using a cast.
67. When deriving a class from a base class with **protected** inheritance, **public** members of the base class became **protected** members of the derived class.
68. When deriving a class from a base class with **public** inheritance, **protected** members of the base class become **public** members of the derived class.
69. A **protected** member of a base class cannot be accessed from a member function of the derived class.
70. In case constructors are not specified in a derived class, the derived class will use the constructors of the base class for constructing its objects.
71. The scope-resolution operator tells us what base class a class is derived from.
72. A derived class is often called a subclass because it represents a subset of its base class.
73. It is permitted to make an object of one class a member of another class.

74. Virtual functions permit us to use the same function call to execute member functions of different classes.
75. A pointer to a base class can point to an object of a derived class of that base class.
76. An **abstract** class is never used as a base class.
77. A pure virtual function in a class will make the class abstract.
78. A derived class can never be made an **abstract** class.
79. A **static** function can be invoked using its class name and function name.
80. The input and output stream features are provided as a part of C++ language.
81. A file pointer always contains the address of the file.
82. Templates create different versions of a function at runtime.
83. Template classes can work with different data types.
84. A template function can be overloaded by another template function with the same function name.
85. A function template can have more than one template argument.
86. Class templates can have only class-type as parameters.
87. A program cannot continue to execute after an exception has occurred.
88. An exception is always caused by a syntax error.
89. After an exception is processed, control will return to the first statement after the **throw**.
90. An exception should be thrown only within a **try** block.
91. If no exceptions are thrown in a **try** block, the **catch** blocks for that **try** block are skipped and the control goes to the first statement after the last **catch** block.
92. The statement **throw**; rethrows an exception.
93. Two **catch** handlers cannot have the same type.
94. Exceptions are thrown from a **throw** statement to a **catch** block.
95. STL algorithms can work successfully with C-like arrays.
96. Algorithms can be added easily to the STL, without modifying the container classes.
97. A **map** can store more than one element with the same key value.
98. A **vector** can store different types of objects.
99. In an associative container, the keys are stored in sorted order.
100. In a **deque**, data can be quickly inserted or deleted at either end.
101. Two functions cannot have the same name in ANSI C++.
102. The modulus operator(%) can be used only with integer operands.
103. Declarations can appear anywhere in the body of a C++ function.
104. All the bitwise operators have the same level of precedence in Java.
105. If $a = 10$ and $b = 15$, then the statement $x = (a > b)\ ?\ a : b$; assigns the value 15 to x.
106. In evaluating a logical expression of type

```
boolean expression - 1 && boolean expression - 2
```

both the boolean expressions are not always evaluated.
107. In evaluating the expression ($x == y$ && $a < b$) the boolean expression $x == y$ is evaluated first and then $a < b$ is evaluated.
108. The **default** case is required in the **switch** selection structure.
109. The **break** statement is required in the default case of a **switch** selection structure.
110. The expression ($x == y$ && $a < b$) is true if either $x == y$ is true or $a < b$ is true.

111. A variable declared inside the **for** loop control cannot be referenced outside the loop.
112. Objects are passed to a function by use of call-by-reference only.
113. We can overload functions with differences only in their return type.
114. It is an error to have a function with the same signature in both the super class and its subclass.
115. Derived classes of an abstract class that do not provide an implementation of a pure virtual function are also abstract.
116. Members of a class specified as **private** are accessible only to the functions of the class.
117. A function declared as **static** cannot access **non-static** class members.
118. A **static** class function can be invoked by simply using the name of the function alone.
119. It is perfectly legal to assign an object of a base class to a derived class reference without a cast.
120. It is perfectly legal to assign a derived class object to a base class reference.
121. All functions in an **abstract** base class must be declared pure virtual.
122. The length of a **string** object **s1** can be obtained using the expression **s1.length**.
123. A **catch** can have comma-separated multiple arguments.
124. It is an error to catch the same type of exception in two different **catch** blocks associated with a particular **try** block.
125. Throwing an **exception** always causes program termination.

Part B

Fill in the Blanks Questions

1. The wrapping up of data and functions into a single unit is called _____.
2. The process by which objects of one class acquire the attributes of objects of another class is known as _____.
3. The insulation of data from direct access by unauthorized functions is called _____.
4. _____ means the ability that one thing can take several distinct forms.
5. _____ are used to document a program.
6. The _____ object extracts values from the keyboard.
7. The object used to display information on the screen in _____.
8. Every C++ program begins execution at the function _____.
9. Every C++ statement ends with a _____.
10. The _____ operator can be used only with integer operands.
11. Objects communicate with each other by sending _____.
12. Relational operators have a _____ precedence than arithmetic operators.

70. To write data that contains variables of type to an object of type of stream, we should use _____ function.
71. The function _____ writes a single character to the associated stream.
72. To place the input pointer in a specified location in the file, we must use the _____ function.
73. Opening a file in ios::out mode also opens it in the _____ mode by default.
74. The read() and write() functions handle data in _____ form.
75. We must open the file using _____ option for performing both input and output operations.
76. Command-line arguments are accessed through arguments to _____.
77. A _____ provides a convenient way to create a family of classes and functions.
78. A function template definition begin with the keyword _____.
79. A call instantiated from a class template is called a _____.
80. All functions instantiated from a function template have the same name; therefore, the compiler applies the concept of _____ resolution to invoke the required function.
81. A template argument is preceded by the keyword _____.
82. A template function works with _____ data types.
83. An exception is typically caused by _____ error.
84. Exception are thrown from a _____ statement to a _____ block.
85. The code that is likely to produce an exception is enclosed in a _____ block.
86. The catch handler _____ will catch all types of exceptions.
87. By default, if no handler is found for an exception, the program _____.
88. The container deque is a _____ type container.
89. The three STL container adapters are stack, queue, and _____.
90. The STL algorithms operate on container elements indirectly using _____.
91. A _____ is an appropriate container if we are given an element's key value and we want to quickly access the corresponding value.
92. In a _____ container, the data can be quickly inserted or deleted at either end.
93. In _____ containers, keys are stored in sorted order.
94. For using function objects, we must include the header file _____.
95. For using the algorithm accumulate(), we must include the header file _____.
96. The _____ operator is used to change the constantness of objects.
97. The operator _____ returns a reference to a type-info object.
98. Non standard casts between unrelated types may be achieved by using the operator _____.
99. The operator _____ qualifies a member with its namespace.
100. The use of specifier _____ to a data item permits us to modify it even when it is a member of a const object.

Part C

Multiple Choice Questions

1. The range of values for the long type data on a 16-bit machine is
   A. -2^{31} to $2^{31}-1$
   B. -2^{64} to 2^{64}
   C. -2^{63} to $2^{63}-1$
   D. -2^{32} to $2^{32}-1$
2. Which of the following represent(s) a hexadecimal number?
   A. 570
   B. (hex) 5
   C. 0X9F
   D. 0X5
3. Which of the following assignments are valid?
   A. float x = 123.4;
   B. long m = 023;
   C. int n = (int)false;
   D. double y = 0X756;
4. What will be the result of the expression 13 & 25?
   A. 38
   B. 25
   C. 9
   D. 12
5. What will be result of the expression 9 | 9?
   A. 1
   B. 18
   C. 9
   D. None of the above
6. Which of the following are correct?
   A. int a = 16, a >> 2 = 4
   B. int b = – 8, b >> 1 = – 4
   C. int a = 16, a >>> 2 = 4
   D. int b = – 8, b >>> 1 = – 4
   E. All the above
7. What will be the values of x, m, and n after execution of the following statements?

```
int x, m, n;
  m = 10;
      n = 15;
      x = ++m + n++;
```

A. x = 25, m = 10, n = 15
B. x = 27, m = 10, n = 15
C. x = 26, m = 11, n = 16
D. x = 27, m = 11, n = 16

8. If m and n are **int** type variables, what will be the result of the expression

```
m % n
```

when m = 5 and *n* = 2?
A. 0
B. 1
C. 2
D. None of the above

9. If m and n are **int** type variables, what will be the result of the expression

```
m % n
```

when m = – 14 and n = – 3?
A. 4
B. 2
C. – 2
D. – 4
E. None of the above

10. Consider the following statements:

```
int x = 10, y = 15;
x = ((x<y) ? (y+x) : (y-x);
```

What will be the value of x after executing these statements?
A. 10
B. 25
C. 15
D. 5
E. Error. Cannot be executed.

11. Which of the following operators could be overloaded?
A. –
B. +
C. + =
D. ::
E. sizeof

12. What is the result of the expression

```
(1 & 2) + (3 | 4)
```

in base ten?

A. 1
B. 2
C. 8
D. 7
E. 3

13. A variable is defined within a block in body of a function. Which of the following are true?
    A. It is visible throughout the function.
    B. It is visible from the point of definition to the end of the program.
    C. It is visible from the point of definition to the end of the block.
    D. It is visible throughout the block.
14. Which of the following are correct?
    A. A relational expression yields an int type result.
    B. A relational expression yields a bool type result.
    C. A logical operator compare two bool type values.
    D. A logical operator combines two bool type values.
15. Which of the following will produce a value of 22 if x = 22.9?
    A. ceil(x)
    B. log(x)
    C. abs(x)
    D. floor(x)
16. Which of the following will produce a value of 10 if x = 9.7?
    A. floor(x)
    B. abs(x)
    C. log(x)
    D. ceil(x)
17. Which of the following expressions are illegal?
    A. (10 | 5)
    B. (false && true)
    C. bool x = (bool)10;
    D. float y = 12.34;
18. When the **break** statement is encountered inside a loop, which one of the following occurs?
    A. Control goes to the end of the program
    B. Control leaves the function that contains the loop
    C. Causes an exit from the innermost loop containing it
    D. Causes an exit from all the nested loop
19. When the **continue** statement is executed within a loop, the control goes to
    A. the next statement in the loop
    B. the top of the loop
    C. the statement immediately after the loop
    D. the beginning of the program
    E. the end of the program

20. Which of the following are illegal loop constructs?

A.
```
while(int i > 0)
{i--;  other statements;}
```

B.
```
for(int i = 10, int j = 0; i+j > 5; i = i-2, j++)
{
                        Body statements
}
```

C.
```
int i = 10;
while(i)
{
                        Body statements
}
```

D.
```
int i = 1, sum = 0;
do  {loop statements}
while(sum < 10  ||  i < 5);
```

21. Consider the following code

```
if(number >= 0)
   if(number > 0)
   cout << "Number is positive\n";
else
   cout << "Number is negative\n";
```

What will be the output if number is equal to 0?

A. Number is negative
B. Number is positive
C. Both A and B
D. None of the above

22. Which of the following control expressions are valid for an **if** statement?

A. an integer expression
B. a boolean expression
C. either A or B
D. Neither A nor B

23. In the following code snippet, which lines of code contain error?

```
int j = 0;
while(j < 10) {
j++;
if(j == 5) continue loop;
cout << "j is" << j;  }
```

A. Line 2
B. Line 3
C. Line 4

D. Line 5
E. None of the above

24. Consider the following code:

```
char c = 'a';
switch(c)
{
        case 'a':
        cout << "A";
        case 'b':
        cout << "B";          break;
        default:
        cout "C";
}
```

For this code, which of the following statements are true?
A. output will be A
B. output will be A followed by B
C. output will be A, followed by B, and then followed by C.
D. code is illegal and therefore will not compile

25. Which of the following cannot be passed to a function?
A. Reference variable
B. Arrays
C. Class objects
D. Header files

26. Which of the following statements are true when a function is called by reference?
A. The called function copies the values into a new set of variables.
B. The formal arguments in the called function becomes aliases to the actual arguments in the calling function.
C. The called function has access to the original data in the calling program.
D. The called function cannot access the values of actual arguments.

27. Which of the following represent correct form of function prototype?
A. float volume(int x, float y);
B. float volume(int x, y);
C. volume(int x, int y);
D. float volume(int, float);
E. float volume(int, float) { }

28. Which of the following apply to a **static** member variable?
A. It is initialized to zero when the first object of its class is created.
B. A separate copy of the variable is created for each object.
C. It retains the value till the end of the program.
D. It is visible to all the classes in the program.

29. Which of the following properties are true for a **static** member function?
A. It has access to all the members of its class.
B. It has access to only other static members of its class.

B. To help modular programming
C. To facilitate the reusability of code
D. To extend the capabilities of a class
E. To hide the details of base classes

42. Consider the following class definition.

```
class Person
{
};
class Student : protected Person
{
};
```

What happens when we try to compile this class?
A. Will not compile because class body of person is not defined
B. Will not compile because the class body of Student is not defined
C. Will not compile because class Person is not public inherited
D. Will compile successfully.

43. Consider the following class definitions:

```
class Maths
{
     Student student1;
};
class Student
{
     String name;
};
```

This code represents:
A. an 'is a' relationship
B. a 'has a' relationship
C. both
D. neither

44. Which of the following are overloading the function

```
int sum(int x, int y)  {  }
```

A. int sum(int x, int y, int z) { }
B. float sum(int x, int y) { }
C. int sum(float x, float y) { }
D. int sum(int a, int b) { }
E. float sum(int x, int y, float z) { }

45. What is the error in the following code?

```
class Test
```

```
{
    virtual void display( );
}
```

A. No error
B. Function **display( )** should be declared as **static**
C. Function display() should be defined
D. **Test** class should contain data members

46. Which of the following declarations are illegal?
A. void *ptr;
B. char *str1 = "xyz";
C. char str2 = "abc";
D. const *int p1;
E. int * const p2;

47. The function **show()** is a member of the class **A** and **abj** is a object of A and **ptr** is a pointer to A. Which of the following are valid access statements?
A. abj.show();
B. abj→show();
C. ptr→show();
D. ptr.show();
E. ptr*show();
F. (*ptr).show();

48. We can make a class abstract by
A. Declaring it abstract using the static keyword
B. Declaring it abstract using the virtual keyword
C. Making at least one member function as virtual function
D. Making at least one member function as pure virtual function
E. Making all member functions **const**.

49. Consider the following code:

```
class A
{ public : virtual void show() = 0; };

class B : public A
{ public : void display()
     { cout << "B"; } };

class C : public A
{  public : void show()
      { cout << "C"; } };
```

Which of the following statements are illegal?
A. C c1;
B. A a1;

C. B b1;
D. A * arr[2];
E. arr[0] = &c1;
F. arr[1] = &b1;

50. The **friend** functions are used in situations where
A. We want to exchange data between classes
B. We want to have access to unrelated classes
C. Dynamic binding is required
D. We want to create versatile overloaded operators

51. By default, all C++ compilers provide a copy constructor. This constructor is invoked when
A. An argument is passed by reference to a function
B. An argument passed to a function is a pointer
C. An argument is passed by value
D. A function returns a value to an object

52. Which of the following ways are legal to access a class data member using the **this** pointer.
A. this → x
B. this.x
C. *this.x
D. *(this.x)
E. (*this).x

53. Which of the following statements are true with respect to the use of **friend** keyword inside a class?
A. A private data member can be declared as a friend.
B. A function may be declared as a friend.
C. A class may be declared as a friend.
D. An object may be declared as a friend.

54. What are the functions that can have access to the protected members of a class?
A. A function that is a friend of the class.
B. A member function of any class in the program.
C. A member function of a class that is a friend of the class.
D. A function is the program that is declared as static.
E. A member function of a derived class.

55. Which of the following are not keywords?
A. NULL
B. abstract
C. protected
D. mutable
E. string

56. Which of the following are keywords?
A. switch
B. integer

C. default
D. bool
E. object

57. Which of the following keywords are used to control access to a class member?
A. default
B. break
C. protected
D. goto
E. public

58. Which of the following keywords were added by ANSI C++?
A. asm
B. explicit
C. enum
D. extern
E. typename
F. using

59. Which of the following statements are valid array declaration?
A. int number(5);
B. float average[5];
C. double[5] marks;
D. counter int[5];
E. int x[5], y[10];

60. What will be the content of array variable table after executing the following code

```
for(int i=0; i<3; i++)
     for(int j=0, j<3; j++)
            if(j == i) table[i][j] = 1;
             else table[i][j] = 0;
```

A. 0 0 0 B. 1 0 0 C. 0 0 1 D. 1 0 0
0 0 0 1 1 0 0 1 0 0 1 0
0 0 0 1 1 1 1 0 0 0 0 1

61. Which of the following methods belong the **string** class?
A. length( )
B. compareTo( )
C. equals( )
D. substring( )
E. All of them
F. None of them

62. Given the code

```
string s1 = "yes";
string s2 = "yes";
string s3 = string s3(s1);
```

Which of the following would equate to **true**?

A. s1 == s2
B. s1 = s2
C. s3 == s1
D. s1.equals(s2)
E. s3.equals(s1)

63. Suppose that s1 and s2 are two strings. Which of the statements or expressions are correct?

A. string s3 = s1 + s2;
B. string s3 = s1 – s2;
C. s1 <= s2
D. s1.compareTo(s2);
E. int m = s1.length( );

64. Given the code

```
string s("abc");
```

Which of the following calls are valid?

A. s.trim( )
B. s.replace('a', 'A')
C. s.substring(3)
D. s.toUpperCase( )

65. Given the declarations

```
bool b;
int x1 = 100, x2 = 200, x3 = 300;
```

Which of the following statements are evaluated to *true*?

A. b = x1 * 2 == x2;
B. b = x1 + x2 != 3 * x1;
C. b = (x3 - 2*x2 < 0) || ((x3 = 400) < 2*x2);
D. b = (x3 - 2*x2 > 0) || ((x3 = 400) < 2*x2);

66. In which of the following code fragments, the variable x is evaluated to 8.

A. int x = 32;
   x = x >> 2;
B. int x = 33;
   x = x >> 2;
C. int x = 35;
   x = x >> 2;
D. int x = 16;
   x = x >> 1;

67. Which of the following represent legal flow control statements?

A. break;

B. break( );
C. continue outer;
D. continue(inner);
E. return;
F. exit( );

68. Templates enables us to create a range of related
A. classes
B. variables
C. arrays
D. functions
E. main() functions

69. The actual source code for implementing a template function is created when
A. the function is actually executed
B. the declaration of the function appears
C. the definition of the function appears
D. the function is invoked

70. An exception is caused by
A. a hardware problem
B. a problem in the operating system
C. a syntax error
D. a run-time error

71. An exception may be thrown from
A. a throw statement in a catch block
B. a try block in a function
C. a function called in a try block
D. a return statement in a function

72. Which of the following statements are true?
A. A program can continue to run after an exception has occurred
B. After the exception is handled in a catch block, the control goes to the next catch block
C. When a catch block finishes executing, the control goes to the first statement after the last catch block
D. When an exception is thrown, the first catch block is executed
E. If no handler is found for an exception, the program terminates

73. Which one of the following is an associative container?
A. list
B. queue
C. map
D. string

74. Which one of the following is a sequence container?
A. stack
B. deque

C. queue
D. set

75. Which of the following containers support the random access iterator?
A. priority-queue
B. multimap
C. list
D. vector
E. multiset

76. Which of the following are non-mutating algorithms?
A. search()
B. accumulate()
C. for_each()
D. rotate()
E. count()

77. Which of the following functions give the current size of a **string** object?
A. max_size()
B. capacity()
C. size()
D. find()
E. length()

78. Consider the following code:

```
class Base
{
     private : int x;
     protected : int y;
};
class Derived : Public Base
{
     int a, b;
     void change()
     {
          a = x;
          b = y;
     }
};
int main()
{
      Base base;
      Derived derived;
      base.y = 0;
      derived.y = 0;
      derived.change();
}
```

Which of the lines in the above program will produce compilation errors?

A. a = x;
B. b = y;
C. base.y;
D. derived.y;
E. derived.change();

79. Which of the following statements are true in C++?
    A. Classes cannot have data as public members
    B. Structures cannot have functions as members
    C. Class members are public by default
    D. None of these
80. What would be the output of the following program?

```
int main()
{
     int x,y=10,z=10;
     x = (y == z);
     cout << x;
     return 0;
}
```

A. 0
B. 1
C. 10
D. Error

Part D

Short Answer Questions

1. What is the **main( )** function? How is it different from other functions?
2. What are the valid types of data that the **main()** can return?
3. Given the command line

```
prog_10 input.dat
```

how would you open the file **input.dat**?

4. How does the function declaration

```
int fun();
```

differ in C and C++?

5. Compare the behaviour of the operator **sizeof()** in C and C++.

6. How does the use of keyword **struct** differ in C and C++?
7. What is the advantage of using named constants instead of literal constants in a program?
8. What is the difference between the following two declarations?

```
extern int m;
int m = 0;
```

9. How do the following two compare?

```
(a)  #define max(x,y) (((x)>(y) ? (x) : (y))
(b)  inline int max(int x, int y)
     { return (x>y) ? x : y; }
```

10. When the following code is executed, what will be the values of x and y?

```
int x=1, y=0;
y = x++
```

11. What are the values of m and n after the following two statements are executed?

```
int m=5;
int n=m++ * ++m;
```

12. Use type casts to the following statements to make the conversion explicit and clear.

```
float x = 10 + intNumber;
int m = 10.0 * intNumber/floatNumber;
```

13. What are lvalues and rvalues?
14. What are **new** and **delete**?
15. What is the difference between using **new** and **malloc()** to allocate memory?
16. In the following statements, state whether the functions **fun1** and **fun2** are value-returning functions or void functions.

```
(a)  x = 10 * fun1(m,n) + 5;
(b)  fun2(m,n);
```

17. What is the difference between using the following statements?

```
(a)  cin >> ch;
(b)  cin.get(ch);
```

18. Write a single input statement that reads the following three lines of input from the screen.

```
100  200
300
400
```

19. What is the problem with the following code?

```
int array[5],i;
for(i=0;i<=5;i++)
     cout << array[i];
```

20. Comment on the following code:

```
int x[3] = { 1,2,3 };
for(int i=1;i<=3;i++)
      cout << "  " << x[i];
```

21. Suppose we want to store and process a table of item names and their cost. Can we use a two-dimensional array? Explain.
22. A character array is created as follows:

```
char *cpr = new char[20];
```

How could we delete the memory created using the operator **delete**?
23. In the statement given below, what is the order of evaluation of operators?

```
y = a* ++b + m/2;
```

24. What is the situation where we need the use of **goto** statement?
25. Given the declarations

```
int test(int x);
int mul(void);
```

State whether the following function calls are legal.

```
test(sizeof(int));
test(mul());
```

26. Given the array declaration

```
int x[10];
```

what does

```
*(x+3)
```

mean?

27. Given the statements

```
int y[5];
int *p = y;
```

is the following statement legal?

```
p[3] = 10;
```

28. How does a C-string differs from a C++ type string?
29. Does an array of characters represent a character string?
30. What is the difference between the following two statements?

```
const int M = 100;
#define M 100
```

31. Given the statement

```
const int size = 5;
```

can we declare an array as follows?

```
int x[size];
```

32. A character array **name** is defined as follows:

```
char name[30] = "Anil Kumar";
```

what will be the values of **m** and **n** in the following statements?

```
int m = sizeof(name);
int n = strlen(name);
```

33. Write a function **change()** to exchange to double values.
34. Write a function to sort a list of double values using the function **change()**.
35. What will be the value of **test** after the following code is executed?

```
int m = 10, n = -1, test = 1;
if(m<15)
     if(n>1)
        test = 2;
else
     test = 3;
```

36. Rewrite the following code using a **while** loop structure.

```
int i;
for(i=0;x<5;++i)
{
     // C++ statements
}
```

37. When do we need to use the following statement?

```
for( ; ; )
```

38. What is wrong with the following function definition? Correct the error.

```
double divide(int m, int n)
{
     return m/n;
}
```

39. What will be the output of the following code?

```
for(int m=0;m<5;m++)
     cout << m;
```

40. Will the following code work? If not, why?

```
int main()
{
     cout << test();
     return 0;
}
float test()
{
      // Function code
}
```

41. If the total mark is below 300, we want to print "FAIL" and if it is 300 and above but less than 359, we want to print "PASS". If it is 360 or more, we do not want to print anything. Will the following code achieve this? If not, correct the code.

```
if(total >= 300)
     if(total < 360)
        cout << "PASS";
else
     cout << "FAIL";
```

42. Rewrite the following sequence of **if ... then** statements using a single **if ... then ... else** sequence.

```
if(m%2 == 0)
     cout << "m is even number \n";
if(m%2 != 0)
{
     cout << "m is odd number \n";
     cout << "m = " << m << "\n";
}
```

43. Simplify the following code segment, if possible.

```
if(value > 100)
     cout << "Tax = 10";
if(value < 25)
     cout << "Tax = 0";
if(value >= 25 && value <= 100)
     cout << "Tax = 5";
```

44. What does the following loop print out?

```
int m = 1;
while(m < 11)
{
     m++;
     cout << m++;
}
```

45. Write a code segment, using nested loops, to display the following output:

```
1  2  3  4  5
1  2  3  4
1  2  3
1  2
1
```

46. A program uses a function named **convert()** in addition to its **main** function. The function main declares a variable x within its body and the function **convert()** declares two variables **y** and **z** within its body, **z** is made static. A fourth variable **m** is declared ahead of both the functions. State the visibility and lifetime of each of these variables.

47. What is the output of the following program?

```
#include <iostream>
```