Cs301

Mid term solve subjective type

by MEHRAN ALI SHAH

2011

mehranali1991@gmail.com

Access control : A C++ mechanism for prohibiting or granting access to individual members of a class. See public, private, protected, and visibility.

Access declaration : A way of controlling access to a specified member of a base class when it is used in a derived class.

Access specifier A way of labelling members of a class to specify what access is permitted i.e public, private, and protected.

accessor : : A public member subprogram that provides query access to a private data member.

agent: : An object that can both initiate behavior in other objects, as well as be operated upon by other objects.

Allocation : The process of giving memory space to an object. See dynamic storage,static storage, and deallocation.

ANSI : Acronym for American National Standards Institute, a standards body currently standardizing C++.

array : : An ordered collection that is indexed.

array constructor: : A means of creating a part of an array by a single statement.

array overflow: An attempt to access an array element with a subscript outside the array size bounds.

array pointer: : A pointer whose target is an array, or an array section.

array section: : A subobject that is an array and is not a defined type component.

assertion: : A programming means to cope with errors and exceptions.

assignment operator: : The equal symbol, "=", which may be overloaded by a user.

attribute: : A property of a variable that may be specified in a type declaration statement.

base class: : A previously defined class whose public members can be inherited by another class. (Also called a super class.)

behavior sharing: : A form of polymorphism, when multiple entities have the same generic interface. This is achieved by inheritance or operator overloading.

binary operator: : An operator that takes two operands.

bintree: : A tree structure where each node has two child nodes.

call-by-reference: : A language mechanism that supplies an argument to a procedure by passing the address of the argument rather than its value. If it is modified, the new value will also take effect outside of the procedure.

call-by-value: : A language mechanism that supplies an argument to a procedure by passing a copy of its data value. If it is modified, the new value will not take effect outside of the procedure that modifies it.

class attribute: : An attribute whose value is common to a class of objects rather than a value peculiar to each instance of the class.

class descriptor: : An object representing a class, containing a list of its attributes and methods as well as the values of any class attributes.

class diagram: : A diagram depicting classes, their internal structure and operations, and the fixed relationships between them.

class inheritance: : Defining a new derived class in terms of one or more base classes.

class: : An abstraction of an object that specifies the static and behavioral characteristics of it, including their public and private nature. A class is an ADT with a constructor template from which object instances are created.

concrete class: : A class having no abstract operations and can be instantiated.

constructor: : An operation, by a class member function, that initializes a newly created instance of a class.

container class: A class whose instances are container objects. Examples include sets, arrays, and stacks.

container object: : An object that stores a collection of other objects and provides operations to access or iterate over them.

data hiding: : The concept that some variables and/or operations in a module may not be accessible to a user of that module; a key element of data abstraction.

information hiding: : The principle that the state and implementation of an object should be private to that object and only accessible via its public interface.

inheritance: : The relationship between classes whereby one class inherits part or all of the public description of another base class, and instances inherit all the properties and methods of the classes which they contain.

instance: : A individual example of a class invoked via a class constructor.

instantiation: : The process of creating (giving a value to) instances from classes.

interaction diagram: : A diagram that shows the flow of requests, or messages between objects.

interface: : The set of all signatures (public methods) defined for an object.

intrinsic constructor: : A class member function with the same name as the class which receives initial values of all the data members as arguments.

Is-A: : A relationship in which the derived class is a variation of the base class.

linked list: : A data structure in which each element identifies its predecessor and/or successor by some form of pointer.

member data: : Variables declared as components of a defined type and encapsulated in a class.

member function: : Subprograms encapsulated as members of a class.

message passing: : The philosophy that objects only interact by sending messages to each other that request some operations to be performed.

message: : A request, from another object, for an object to carry out one of its operations.

method: : A class member function encapsulated with its class data members.

object : : A concept, or thing with crisp boundaries and meanings for the problem at hand; an instance of a class.

object diagram: A graphical representation of an object model showing relationships, attributes, and operations.

object-oriented (OO): : A software development strategy that organizes software as a collection of objects that contain both data structure and behavior.

object-oriented programming (OOP) : Object-oriented programs are object-based, class-based, support inheritance between classes and base classes and allow objects to send and receive messages.

operation: : Manipulation of an object's data by its member function when it receives a request.

operator overloading: : A special case of polymorphism; attaching more than one meaning to the same operator symbol. 'Overloading' is also sometimes used to indicate using the same name for different objects.

overloading: : Using the same name for multiple functions or operators in a single scope.

overriding: :	The ability to change the definition of an inherited method or attribute in a subclass.
parameterized classes: :	A template for creating real classes that may differ in well-defined ways as specified by parameters at the time of creation. The parameters are often data types or classes, but may include other attributes, such as the size of a collection. (Also called generic classes.)
pointer: :	A single data object which stands such as an array, or defined type.
polymorphism: :	The ability of an function/operator, with one name, to refer to arguments, or return types, of different classes at run time.
private: :	That part of an class, methods or attributes, which may not be accessed by other classes, only by instances of that class.
protected: :	(Referring to an attribute or operation of a class in C++) accessible by methods of any descendent of the current class.
public: :	That part of an object, methods or attributes, which may be accessed by other objects, and thus constitutes its interface.
super class: :	A class from which another class inherits.

Question: What is Abstraction?

Answer: The importance of abstraction is derived from its ability to hide irrelevant details and from the use of names to reference objects. Abstraction is essential in the construction of programs. It places the emphasis on what an object is or does rather than how it is represented or how it works. Thus, it is the primary means of managing complexity in large programs.

Question: What is a Class Diagram?

Answer: A class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects.

Question: What is Method Overriding?

Answer: Method overriding is a language feature that allows a subclass to override a specific implementation of a method that is already provided by one of its super-classes. A subclass can give its own definition of methods but need to have the same signature as the method in its super-class. This means that when overriding a method the subclass's method has to have the same name and parameter list as the super-class's overridden method.

Question: What is Operator Overloading?

Answer: The operator overloading is a specific case of polymorphisms in which some or all of operators like +, - or == are treated as polymorphic (multi) functions and as such have different behaviors depending on the types of its arguments.

Question: What is Method Overloading?

Answer: The method overloading is the ability to define several methods (in same class) all with the same name but different on the basis of i) number of parameters ii) types of parameters.

Question: What is Polymorphisms?

Answer: Polymorphism is a generic term that means 'many shapes'. More precisely Polymorphism means the ability to request that the same operations be performed by a wide range of different types of things.

Question: What is Inheritance?

Answer: Ability of a new class to be created, from an existing class by extending it, is called inheritance.

Question: What is a base class?

Answer: When inheritance is used to create a new class from another, the new class is called the subclass or derived class, and the class from which it was derived is called the base class.

Question: What is a concrete class?

Answer: A concrete class is one that can be used to directly create, or instantiate objects, unlike an abstract base class which can only be used as a base class for other classes which eventually lead to concrete classes

Question: What are data members?

Answer: Objects are miniature programs, consisting of both code and data. The code consists of a series of member functions. The data items are called data members.

Question: What is a constructor?

Answer: Objects are complete, miniature programs and, like any good programs, have well defined initialization and termination phases. They have special routines (i.e. member functions) to look after this. The initialization routine is called the constructor, and C++ ensures that every object is properly initialized by calling its constructor. The designer of the object can have more than one constructor, a situation called overloading and then the compiler will select between them depending on exactly what arguments are passed to the constructor function. However there must always be a default constructor, to be used when no information is supplied.

Question: What is a destructor?

Answer: The termination routine is called the destructor, and C++ will provide a default if none is supplied. If, during the lifetime of the object, it uses heap memory then the designer of the object must provide a destructor function to release such memory to avoid a memory leak.

Question: What is global variable?

Answer: Global variables can be accessed throughout a program. Another way to put this is to say they have global scope.

Question: What is local variable?

Answer: Local variables can only be accessed within the function, or more specifically the compound statement in which they are declared. Another way to put this is to say they have local scope.

Question: What is a null pointer?

Answer: A null pointer is a pointer that is currently pointing to nothing. Often pointers are set to zero to make them null pointers or tested against zero to see if they are null or not.

Question: What is a pointer?

Answer: A pointer is a variable that holds the address of another variable or object.

Question: What is meant by protected?

Answer: The protected keyword in the class statement means that the following members of the class are not available to users of the objects of the class, but can be used by any subclass that inherits from it, and consequently forms part of its implementation.

Question: What is OOP?

Answer: The object oriented programming is commonly known as OOP. Most of the languages are developed using OOP concept. Object-oriented programming (OOP) is a programming concept that uses "objects" to develop a system. An object hides the implementation details and exposes only the functionalities and parameters it requires to its client. Here also an object shares the same concept as that of a bike. While driving a motor bike, we are unaware of its implementation details such as how it is developed, internal working of gears etc.? We know only the functions or actions it can perform.

Question: What are the various elements of OOP?

Answer: Various elements of OOP are: Object Class Method Encapsulation Information Hiding Inheritance Polymorphism

Question: What are the characteristics of Object Oriented programming language?

Answer: Some key features of the Object Oriented programming are: Emphasis on data rather than procedure Programs are divided into entities known as objects Data Structures are designed such that they characterize objects Functions that operate on data of an object are tied together in data structures Data is hidden and cannot be accessed by external functions Objects communicate with each other through functions New data and functions can be easily added whenever necessary Follows bottom up design in program design

Question: What are the basic Concepts used in the Object-Oriented Programming language?

Answer: Object Class Data Abstraction and Encapsulation Polymorphism Inheritance Message passing Dynamic binding

Question: What Is An Object? (Object-Oriented Technology)

Answer: There are many definitions of an object, such as found in [Booch 91, p77]: "An object has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable". This is a "classical languages" definition, as defined in [Coplien 92, p280], where "classes play a central role in the object model", since they do not in prototyping/delegation languages. "The term object was first formally applied in the Simula language, and objects typically existed in Simula programs to simulate some aspect of reality" [Booch 91, p77]. Other definitions referenced by Booch include Smith and Tockey: "an object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain." and [Cox 91]: "anything with a crisply defined boundary" (in context, this is "outside the computer domain". A more conventional definition appears on pg 54). Booch goes on to describe these definitions in depth. [Martin 92, p 241] defines: "An "object" is anything to which a concept applies", and "A concept is an idea or notion we share that applies to certain objects in our awareness". [Rumbaugh 91] defines: "We define an object as a concept, abstraction or thing with crisp boundaries and meaning for the problem at hand." [Shlaer 88, p 14] defines: "An object is an abstraction of a set of real-world things such that:

Question: What Is Object Encapsulation (Or Protection)?

Answer: [Booch 91, p. 45] defines: "Encapsulation is the process of hiding all of the details of an object that do not contribute to its essential characteristics." [Coad 91, 1.1.2] defines: "Encapsulation (Information Hiding). A principle, used when developing an overall program structure, that each component of a program should encapsulate or hide a single design decision... The interface to each module is defined in such a way as to reveal as little as possible about its inner workings. [Oxford, 1986]" Some languages permit arbitrary access to objects and allow methods to be defined outside of a class as in conventional programming. Simula and Object Pascal provide no protection for objects,

meaning instance variables may be accessed wherever visible. CLOS and Ada allow methods to be defined outside of a class, providing functions and procedures. While both CLOS and Ada have packages for encapsulation, CLOS's are optional while Ada's methodology clearly specifies class-like encapsulation (Adts). However most object-oriented languages provide a well defined interface to their objects thru classes. C++ has a very general encapsulation/protection mechanism with public, private and protected members. Public members (member data and member functions) may be accessed from anywhere. A Stack's Push and Pop methods will be public. Private members are only accessible from within a class. A Stack's representation, such as a list or array, will usually be private. Protected members are accessible from within a class and also from within subclasses (also called derived classes). A Stack's representation could be declared protected allowing subclass access. C++ also allows a class to specify friends (other (sub)classes and functions), that can access all members (its representation). Eiffel 3.0 allows exporting access to specific classes.

Question: What Is A Class?

Answer: A class is a general term denoting classification and also has a new meaning in object-oriented methods. Within the OO context, a class is a specification of structure (instance variables), behavior (methods), and inheritance (parents, or recursive structure and behavior) for objects. As pointed out above, classes can also specify access permissions for clients and derived classes, visibility and member lookup resolution. This is a feature-based or intensional definition, emphasizing a class as a descriptor/constructor of objects (as opposed to a collection of objects, as with the more classical extensional view, which may begin the analysis process). Original Aristotlean classification defines a "class" as a generalization of objects: [Booch 91, p93] "a group, set, or kind marked by common attributes or a common attribute; a group division, distinction, or rating based on quality, degree of competence, or condition".

Question: What Is A Meta-Class?

Answer: Meta-Class is a class' class. If a class is an object, then that object must have a class (in classical OO anyway). Compilers provide an easy way to picture Meta-Classes. Classes must be implemented in some way; perhaps with dictionaries for methods, instances, and parents and methods to perform all the work of being a class. This can be declared in a class named "Meta-Class". The Meta-Class can also provide services to application programs, such as returning a set of all methods, instances or parents for review (or even modification). [Booch 91, p 119] provides another example in Smalltalk with timers. In Smalltalk, the situation is more complex

Question: What Is Inheritance?

Answer: Inheritance provides a natural classification for kinds of objects and allowsfor the commonality of objects to be explicitly taken advantage of in modeling and constructing object systems. Natural means we use concepts, classification, and generalization to understand and deal with the complexities of the real world. See the example below using computers. Inheritance is a relationship between classes where one class is the parent base/superclass/ancestor/etc.) class of another. Inheritance provides programming by extension (as opposed to programming by reinvention [LaLonde 90]) and can be used as an is-a-kind-of (or is-a) relationship or for differential programming. Inheritance can also double for assignment

Question: What Is The Difference Between Object-Based And Object-Oriented?

Answer: Object-Based Programming usually refers to objects without inheritance [Cardelli 85] and hence without polymorphism, as in '83 Ada and Modula-2. These languages support abstract data

types (Adts) and not classes, which provide inheritance and polymorphism. Ada95 and Modula-3; however, support both inheritance and polymorphism and are object-oriented. [Cardelli 85, p481] state "that a language is object-oriented if and only if it satisfies the following requirements: - It supports objects that are data abstractions with an interface of named operations and a hidden local state. - Objects have an associated type. - Types may inherit attributes from supertypes. object-oriented = data abstractions + object types + type inheritance These definitions are also found in [Booch 91, Ch2 and Wegner 87]. [Coad 91] provides another model: Object-Oriented = Classes and Objects + Inheritance + Communication with messages

Question No: 19 (Marks: 2)

What is difference between call by reference and call by value?

One application is to find duplicates in a list of numbers.

Let a given list be" 12 34 56 89 33 11 89

the first number in the list is placed in a node that is established as the root of a binary tree. Each number is compared with the node in the root, if the number is larger, we search the right sub-tree else we search the left sub-tree. If the sub-tree is empty, the number is not a duplicate and this will be added as a new node.

2. Binary trees can be used for sorting a given list such that, if we take the first number as root, the numbers less than that number will be transfered to left sub-tree and the greater numbers to right sub-tree.

3. Binary trees are also used for developing the huffman codes.

Question No: 20 (Marks: 3)

What is the functionality of the following method of BST class

```
TreeNode* function(TreeNode* tree)
{
if( tree == NULL )
return NULL;
if( tree->getLeft() == NULL )
return tree; // this is it.
return function( tree->getLeft() );
}
```

Question No: 21 (Marks: 3)

a) Write a C++ statement that declares a valid reference of int i;

b) What is the benefit of reference and where can we use it?

In the last lecture we were discussing about reference variables, we saw three examples; call by value, call by reference and call by pointer. We saw the use of stack when a function is called by value, by reference or by pointer. The arguments passed to the function and local variables are pushed on to the stack. There is one important point to note that in this course, we are using C/C++ but the usage of stack is similar in most of the computer languages like FORTRAN and Java . The syntax we are using here is C++ specific, like we are sending a parameter by pointer using & sign. In

Java, the native data types like *int*, *float* are passed by value and the objects are passed by reference. In FORTRAN, every parameter is passed by reference. In PASCAL, you can pass a parameter by value or by reference like C++. You might have heard of ALGOL, this language had provided another way of passing parameter ca

lled *call by name*. These kinds of topics are covered in subjects like

Question No: 22 (Marks: 5)

Determine what the following recursive "mystery" function computes when given a pointer to the root node of a binary tree.

```
struct bt_s { int key; struct bt_s *left, *right; } bt_t;

int MFunc (bt_t *T) {

int N1, N2;

if (T == NULL) return -1;

N1 = MFunc(T->left);

N2 = MFunc(T->right);

return (N1 > N2 ? N1 : N2) + 1;

}
```

Differences between Linked List and Arrays

Linked lists are data structure which are the self-referential class objects called nodes. Each nodes are connected by reference links. Make yourself clear that linked lists are not dynamically sized array.
An array is data structure (type of memory layout) that stores a collection of individual values that are of the same data type. Arrays are useful because instead of having to separately store related information in different variables (named memory locations), you can store them—as a collection—in just one variable. It is more efficient for a program to access and process the information in an array, than it is to deal with many separate variables.

Consider the following points of differences between the two:

Linked Lists	Arrays
1. Linked lists allow only sequential access to elements. Thus the algorithmic complexities is order of O(n)	1. Arrays allow random access to its elements and thus the complexity is order of O(1)
2. Linked lists require an extra storage for references. This makes them impractical for lists of small data items such as characters or boolean values.	2. Arrays do not need an extra storage to point to next data item. Each element can be accessed via indexes.
3. The size of Linked lists are dynamic by nature.	3. The size of array is restricted to declaration.

4. Elements can be inserted and deleted in linked lists indefinitely.	4. Insertion/Deletion of values in arrays are very expensive. It requires memory reallocation.

Differences between Array and stack

Answer

Briefly, there are two main differences between an array and a stack. Firstly, an array can be multi-dimensional, while a stack is strictly one-dimensional. Secondly, an array allows direct access to any of its elements, whereas with a stack, only the 'top' element is directly accessible; to access other elements of a stack, you must go through them in order, until you get to the one you want.

Give the names of basic Queue Operations

Ans:
Definition: A collection of items in which only the earliest added item may be accessed. Basic operations are add (to the *tail*) or enqueue and delete (from the *head*) or dequeue. Delete returns the item removed. Also known as "first-in, first-out" or FIFO.

Question No: 18 (Marks: 1)

Give one benefit of using Stack.

In computer science, a stack is a last in, first out (LIFO) abstract data type and data structure. A stack can have any abstract data type as an element, but is characterized by only two fundamental operations: push and pop. the data structure itself enforces the proper order of calls.

1. Why List wastes less memory as compared to Arrays.

Ans:
1. Linked lists do not need contiguous blocks of memory; extremely large data sets stored in an array might not be able to fit in memory.
2. Linked list storage does not need to be preallocated (again, due to arrays needing contiguous memory blocks).
3. Inserting or removing an element into a linked list requires one data update, inserting or removing an element into an array requires n (all elements after the modified index need to be shifted).
Array is a collection of same data type. In linked list there are two field one is address and other is pointer. In array elements are arranged in a specific order

2. Why we can change the size of list after its creation when we can not do that in simple arrays.

Some how the answer will be same as part 1 because Inserting or removing an element into a linked list requires one data update, inserting or removing an element into an array requires n (all elements after the modified index need to be shifted).
Array is a collection of same data type. The size of array is mentioned with its declaration. in arrays the elements are in contiguous position. one must after another. while in linked list we gave the address of next element in the next part of node.