CS201 Notes for Final Term (Lecture#39-45) Preparation
Prepared By VU Learning
Video Link: https://youtu.be/XzxLnuSRUMA

Lecture No. 39:

Pointers: Pointer is a special *type of variable* that contains *a memory address*. It is not a variable that contains a value, rather an address of the memory that is contained inside a pointer variable.

The & operator is used to *get the address of a variable or an object*. The & sign is also used as a *short hand for a reference*.

Reference: A reference can be considered as a special type of pointer as it also contains memory address.

Difference: Pointers may *point to nothing* while references always have to *point to something*. The references used when we are implementing the call by reference & we can implement the call by reference without using the * operator.

Call by Value: We call a function and pass an argument, an object or variable to the function. *Original data remains at its place and a temporary copy of it is made and passed to the function*. Whatever the function does with this copy, the original value, in the calling function, remains intact. This is a call by value.

Call by Reference: If we want a function to change something in the original object variable or whatever, *then that variable or object by reference would be passed*. To do this, we don't make temporary copy of that object or variable. Rather, the *address of the variable is sent*. When the function manipulates it, the original object will be manipulated, effecting change in its values.

Which one is Efficient?

The use of *call by reference* is also important for the sake of efficiency. *If we have a large object, sending of its copy will be something insufficient*. It will occupy a large space on the stack. Here, we can use call by reference instead of call by value

only for efficiency while we need not to change the original object. For this, we use a keyword *const* that means that ***a const (constant) reference*** is being passed. The function can use its values but cannot change it.

In static memory allocation, whenever we declare an array, we have to *mention its size*. It is necessary, as otherwise no memory will be allocated.

We do initialization as

int i = 0 ;

OR

int i ;
i = 0 ;

The *default assignment* of C will carry out. (If we have not defined the overloaded operator for Assignment). The default assignment is a *member-to-member assignment.*

A **copy constructor** is a special constructor for creating a new object as a copy of an existing object. Constructor that will create a new object with a full copy of the other object. This is also known as deep copy as opposed to shallow copy.

Rules for Using Dynamic Memory Allocation:

1) First, we must *define a constructor* for it. Otherwise, we will not be able to carry out dynamic memory allocation.
2) Secondly, we must *write an assignment operator* for that class. This assignment operator should first check the self-assignment and then make a deep copy.

3) Thirdly, as we are doing dynamic memory allocation in the constructor, it is necessary to *provide a destructor*. This destructor should free the allocated memory.
   Matrix m2 = m1 ; It is not an assignment operator. It is a copy constructor.

Lecture No. 40:

A class is a user defined data type and it can be used inside other classes in the same way as native data types are used. Thus we can create classes that contain objects of other classes as data members.
Construction of an object is that the contained data members of the object are constructed before the object itself.

The order of the execution of initializes is the same as the order of declarations of objects inside the outer class.
The default scope for members of structures is *public* whereas the default visibility for class members is *private*.

A class can contain instances of other classes as its data members.
• It is a way of reusing the code when we contain objects of our already written classes into a new class.
The order of destruction of an object is reverse to this construction order, where the outer object is destroyed first before the inner data members.

• Initializes list is used to initialize the inner objects at the construction time.
Classes defined within other classes are called nested classes.

Lecture No.41:

A **template** is a sketch to draw some shape or figure. Template is allowing us the reuse of a certain shape/code.

For normal code, you would use a class template when you want to create a class that is parameterized by a type, and a function template when you want to create a function that can operate on many different types.

Macro is a code substitution while *#define* is a value substitution.

Lecture No. 42:

Static variable members are used to define ordinary classes. The static variable has a single copy for the whole class. So there are not separate copies of the static data variable for each object like ordinary data members.

Advantages and Disadvantages of Templates:
Advantages:

- Templates are easier to write than writing several versions of your similar code for different types

- Templates can be easier to understand, since they can provide a straightforward way of abstracting type information.

- ❖ Templates are type-safe. This is because the types that templates act upon are known at compile time, so the compiler can perform type checking before errors occur.
- ❖ Templates help in utilizing compiler optimizations to the extreme.

Disadvantages:

If misused of templates then:

- ❖ Templates can make code difficult to read and follow depending upon coding style.
- ❖ They can present seriously confusing syntactical problems esp. when the code is large and spread over several header and source files.

- ❖ Then, there are times, when templates can "excellently" produce nearly meaningless compiler errors thus requiring extra care to enforce syntactical and other design constraints. A common mistake is the angle bracket problem.

STL is a part of the official standard of C++. It is called STL i.e. Standard Template Library. As a library, it is a tested code base. STL is a lot of important code, pre-developed for us. It is available as a library.

Lecture No. 43:

A matrix is nothing but a two-dimensional array of numbers. It is normally represented in rows and columns.

Order of the matrix is:

no. of rows * no. of columns.

Operations Performed on matrices are:

- *A matrix is added to another matrix.*
- *A scalar value (an ordinary number) is added to a matrix.*
- *A matrix is subtracted from another matrix.*
- *A scalar number is subtracted from a matrix.*
- *A matrix is multiplied with another matrix.*
- *A scalar number is multiplied with a matrix.*
- *A matrix is divided by a scalar.*
- *A matrix is transposed.*

(Also Practice its examples from handouts L#43)

Lecture No.44:

There are two input functions both named as *input*. One is used to get the value from the keyboard while the other to get the values from some file. The *input* function which will get the input from the keyboard, takes an argument of type *istream*.

The functions *getRows()* and *getCols()* are relatively simple. They do not change anything in the object but only read from the object.

The **transpose** of a matrix will interchange rows into columns.

Lecture No. 45:

Arrays

An array is a type of data structure. We use an array to store multiple values of the same data type.

The **variable** is a name for a value. It is like a label on a box in the memory, which contains a value. We can use this label to manipulate the value, instead of using the address of the memory that contains the value.
A **pointer** is an address of a location in the memory.

Loops and Decisions:

The loops and decisions are 'bread and butter' for a programmer.
While talking about decisions, we read that in C and C++ languages, there is a simple *if* statement. There is also '*if-else* statement'. We can write a big structure by using the nested *if-else* statements.

Rules for Programming:

We need simply three constructs to solve any problem.

1) Things should be executed sequentially. That means the statements are executed in a sequence i.e. the second statement follows the first and so on.

2) We need to have a decision. The decision means if something is true, the program executes it. Otherwise, it tries doing something else. So there is simple decision or *if-else* decision.

3) The third construct is loop, which is a repetition structure that performs the same task repeatedly with different values.

(Please also prepare Past Papers for subjective as well as objective.)

Video Link:
https://youtu.be/XzxLnuSRUMA

Channel VU Learning:

https://www.youtube.com/c/VUlearning62

Remember me in your prayers & Keep supporting me

Regards: MARIA PARVEEN (VU Learning)