**SUNBEAM INFOTECH**

**CPP Day – 4 (DAC)**

* Exception Specification List
* Exception Handling ( class as a Exception/user defined exception)
* Inspector function
* Mutator function
* Facilitator function
* Destructor
* Dynamic Memory Management
* Static Data Member
* Static Member Function

**\* Exception Specification List**

* An **exception specification** provides a solution that can be used to list the exceptions a function may throw with the function declaration.
* An exception specification follows the function parameter list. It is specified with the keyword throw followed by a list of exception types enclosed in parentheses.
* If an function fails to perform operation then it can throw exception. To maintain documentation of exception thrown by the function we should use exception specification list.
* To define exception specification list, we should use throw keyword.
* If exception specification list do not contain type of thrown exception then during failure it doesnt execute catch block rather C++ runtime give call to **std::unexpected** function which implicitly gives call to the **std::terminate** function which will give call to abort function.
* There might be a situation where , if function throw any excecption that is not in the specification list, then C++ runtime give call to **std::unexpected** function which implicitly gives call to the **std::terminate** function which will give call to abort function.
* If exception is not handled then c++ runtime give call to **std::unexpected** function which implicitly gives call to the **std::terminate** function which will give call to abort function.

class iStack {

public:

// ...

void pop( int &value );

// throw popOnEmpty

void push( int value );

// throw pushOnFull

private:

// ...

};

class iStack {

public:

// ...

void pop( int &value ) throw(popOnEmpty);

void push( int value ) throw(pushOnFull);

private:

// ...

};

**\* Class as a exception / user defined exception**

* **Destructor**
  + It is a member function of a class which is used to release the resources.
  + It is considered as special function of the class
    - Its name is same as class name and always preceds with tild operator( ~ )
    - It doesnt have return type or doesn't take parameter.
    - It is designed to call implicitly.
  + Destructor calling sequence is exactly opposite of constructor calling sequence.
  + If we do not define destructor inside class then compiler generates default destructor for the class.
  + Default destructor do not deallocate resources allocated by the programmer. If we want to deallocate it then we should define destructor inside class.
* **Mutators/setter : modify state of object**
* **inspector/getter : do not change the state of the object**
* **facilitator**

Getters and Setters allow you to effectively protect your data.

This is a technique used greatly when creating classes. For each variable, a get method will return its value and a set method will set the value.

Firstly, if you decide that some action should be taken every time you change the value of a particular variable, you only need to edit the set method instead of looking for every place you change the variable. Another reason is that a person using your code can look for all methods starting with get or set instead of looking for specific variables that they need to remember.

**\* Dynamic Memory Allocation**

**Memory in your C++ program is divided into two parts −**

* **The stack** − All variables declared inside the function will take up memory from the stack.
* **The heap** − This is unused memory of the program and can be used to allocate the memory dynamically when program runs.

If we want to allocate memory dynamically then we should use **new operator** and to deallocate that memory we should use **delete operator.**

If pointer contains, address of deallocated memory then such pointer is called **dangling pointer.**

When we allocate space in memory, and if we loose pointer to reach to that memory then such wastage of memory is called **memory leakage.**

**\* Dynamic Memory Allocation for Arrays**

Consider you want to allocate memory for an array of characters, i.e., string of 20 characters. Using the same syntax what we have used above we can allocate memory dynamically as shown below.

Char \*pvalue = NULL; //pointer initialized with null

pvalue = new char[20]; // request memory for the variable

**OR**

**char \*pvalue=new char[20] ;**

**char arr\_name=new char[size] ; //syntax**

**To remove the array that we have just created the statement would look like this −**

delete [] pvalue; // Delete array pointed to by pvalue

**//Memory allocation and deallocation for single dimensional array**

## **\*Dynamic Memory Allocation for Objects**

**Syntax:**

**Classname \*obj\_name=new classname[size];**

To deallocate the memory assigned by new operator in array of objects we need to compulsory use delete operator

otherwise dangling pointer will occur and memory leakage will occur.

**//Syntax to deallocate the memory using delete the number of objects**

**delete[] dobj\_name;**

**dobj\_name=NULL;**

**\* Static Variable**

All the static and global variables get space only once during program loading / before starting execution of main function

* Static variable is also called as shared variable.
* Unintialized static and global variable get space on BSS segment.
* Intialized static and global variable get space on Data segment.
* Default value of static and global variable is zero.
* Static variables are same as global variables but it is having limited scope.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*Program \*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Static Methods or Static Member Functions**

* Except main function, we can declare global function as well as member function static.
* To access non static members of the class, we should declare member function non static and can be accessed using object ame of class.
* Access static members of the class we should declare member function static and access it using class name.
* static member function is also called as class level method.
* To access class level method we should use classname and ::(scope resolution) operator.