**Sunbeam Infotech**

**DAC : CPP - Day 3**

* differece between <headerfilename.h> and “headerfilename.h”
* Header Guard
* Modular approach
* Constructor's member initializer list
* constant data member and member function, mutable keyword
* Reference
* Exception Handling (try, catch, throw)
* Scope
* Stack Unwinding

**\* Difference between <filename.h> and “filename.h”**

-"/usr/include" directory is called standard directory for header files.

-<filename.h> preprocessor try to locate and load header file from standard directory only(/usr/include).

-”filename.h” - preprocessor try to locate and load header file first from current project directory if not found then it try to locate and load from standard directory.

**\* Header Guards**

-If we want to expand contents of header file only once then we should use header guard.

#ifndef HEADER\_FILE\_NAME\_H\_

#define HEADER\_FILE\_NAME\_H\_

//TODO : Type declaration here

#endif

**\*Modular Approach**

**\* Constructor Member Initilization**

* + - If we want to initialize data members according to users requirement then we should use constructor body.
    - Except array we can initialize any member inside **constructors member initializer list.**

**\* Constant Data Member**

- Once initialized, if we dont want to modify state of the data member inside any member function of the class including constructor body then we should declare data member constant.

- If we declare data member constant then it is mandatory to initialize it using constructors member initializer list.

**\*Constant member function**

- We can not declare global function constant but we can declare member function constant.

- If we dont want to modify state of current object inside member function then we should declare member function constant.

**We can not delclare following function constant:**

1. Global Function

2. Static Member Function

3. Constructor

4. Destructor

Since main function is a global function, we can not delcare it constant.

We should declare read only function constant. e.g getter function, printRecord function etc.

**Mutable Keyword:**

In constant member function, if we want to modify state of non constant data member then we should use mutable keyword.

**\*Reference**

**-**Reference is derived data type.

-It alias or another name given to the exisiting memory location / object.

-It is mandatory to initialize reference.

-We can not create reference to constant value.

int &r=a;

**//We should not retrun non static local variable from function by address.**

**//If we want to return local variable from function by address then we should use static keyword.**

**//We should not return local variable from function by reference.**

**\*Execption Handling**

**Errors can be broadly categorized into two types. We will discuss them one by one.**

1. Compile Time Errors
2. Run Time Errors

**Compile Time Errors**– Errors caught during compiled time is called Compile time errors. Compile time errors include library reference, syntax error or incorrect class import.

**Run Time Errors**- They are also known as exceptions. An exception caught during run time creates serious issues.

**In C++, Error handling is done using three keywords:**

try, catch, throw

## **try block**

The code which can throw any exception is kept inside(or enclosed in) a try block. Then, when the code will lead to any error, that error/exception will get caught inside the catch block

1. try is keyword in C++.
2. If we want to inspect exception then we should put statements inside try block/handler.
3. try block must have at least one catch block/handler

**catch block**

catch block is intended to catch the error and handle the exception condition. We can have multiple catch blocks to handle different types of exception and perform different actions when the exceptions occur.

* If we want to handle exception then we should use catch block/handler.
* Single try block may have multiple catch block.
* Catch block can handle exception thrown from try block only.
* A catch block, which can handle any type of exception is called generic catch block / catch-all handler.

## **throw statement**

It is used to throw exceptions to exception handler i.e. it is used to communicate information about error. A throw expression accepts one parameter and that parameter is passed to handler.

throw statement is used when we explicitly want an exception to occur, then we can use throw statement to throw or generate that exception.

If we give wrong input to the application then it generates runtime error/exception.

Exception is an object, which is used to send notification to the end user of the system if any exceptional situation occurs in the program.

**\*\*\* Example simple try, throw, catch block**

**\*\*\* Program to check divide by zero exception**

**\*\*\* Program to check array index out of bound exception**

**\*Scope**

It decides area/region/boundry in which we can access the element.

**Types of scope in C++:**

**1. Block scope**

**2. Function scope**

**3. Prototype scope**

**4. Class scope**

**5. Namespace scope**

**6. File scope**

**7. Program scope**

**\* Example Scope**

int num6; **//Program Scope**

static int num5; **//File Scope**

namespace ntest

{

int num4; **//Namespace scope**

class Test

{

int num3; **//Class Scope**

};

}

void sum( int num1, int num2 ); **//Prototype scope**

int main( void )

{

int num1 = 10; **//Function Scope**

while( true )

{

int temp = 0; **//Block Scope**

}

return 0;

}

**\* Stack Unwinding**

During execution of function if any exception occurs then process of destroying FAR and returning control back to the calling function is called stack unwinding.

Let's take a closer look at how exceptions are handled. When the throw occurs, C++ first copies the thrown object to some neutral place. It then begins looking for the end of the current try block. If a try block is not found in the current function, the function's execution is terminated and control passes to the calling function. A search is then made of that function. If no try block is found there, control passes to the function that called it, and so on up the stack of calling functions. This process is called \unwinding the stack."

Basically the computer searches for a try block by going to higher and higher levels of the program. As stack unwinding occurs, any objects that go out of scope are destructed just as if the function had executed a return statement. This keeps the program from losing assets or leaving objects dangling.

When the encasing try block is found, the code searches for the the first catch phrase immediately following the closing brace of the try block. If the object thrown matches the type of ob ject that the catch phrase expects, then control passes to that catch phrase. If not, a check is made of the next catch phrase. If no matching catch phrases are found, the code searches the next higher level for a try block in an ever outward spiral until an appropriate catch phrase can be found. If no catch phrase is found, the program is terminated