C++ Notes Day-4 Date: 08 June 2025

Lets revise

- Structure in C++

Class and Object and related C++ concepts

- Class
  - A class is blueprint of an object.
  - It represents attribute and behaivour of a real world entity known as object/instance.
  - Attributes are defined using data members and behaivour defined using member functions / methods / procedure.
  - We can defined inside a class:
    - Data Members
      - static: It is releted to the class or common data among all the objects.
      - non-static: It related to the objects/instances. It is indivisual data of the objects.
    - Member Functions
      - static
      - non-static
        - const
        - virtual
    - Constructor
    - Destructor
    - Class, Structure, Enum, any Nested types
- Object
  - Variable of the class is known as Object.
  - Object is also known instance.
  - Syntax to create the object of class

```
Employee e1;    // Statically created object of class Employee: Stack Frame
new Employee(); //Dynamically created object of the class Employee: Heap Section
```

- Example:

```
#include<iostream>
using namespace std;

class Employee          //Class is a blueprint of the object
{
public:
    int EmpId;          //Public Data Members of the class
    char Name[50];
    float Salary;
```

```
    int Age;

    void Scandata()
    {
        printf("Enter EmpId: ");
        fflush(stdout);
        scanf("%d",&this->EmpId);
        fflush(stdout);
    }
    void PrintData()
    {
        printf("Emp Id:%d", EmpId);
    }
};
int main()
{
    Employee e;     //Object Employee class

    e.Scandata();   //Calling Member function with the help of class object using
Member Selection operator (.) is known as Message Passing
    e.PrintData();  //Message Passing

    return 0;
}
```

Custom Header File and Header Guard

- #include<abc.h> versus #include"abc.h"
  - Standard directory for standard header file : C:\MicGW\include
  - If we include header file in angular bracket(< >) then preprocessor try to locate that file inside standard directory only.
  - Example: #include<stdio.h>
  - If we include header file in double quotes(" ") then preprocessor first try to locate that file inside current project directory. If not found then it will try to locate it from standard directory.
  - Example:
    - #include<stdio.h>
    - #include"stdio.h"
- Importance of Header Guard
  - If we want to expand contents of header file only once then we should use Header Guard inside header file.

```
#ifndef STUDENT_H_
#define STUDENT_H_
//Declarations
#endif /* STUDENT_H_ */
```

Storage Classes C/C++

- There are four type of storage classes in C/C++
  - auto
  - extern
  - static
  - register
- storage classes describe scope and lifetime of the variable.
- non-static vs static global variable
  - We can access non static global variable inside same file where it is declared as well as inside diffrent file using extern keyword.
  - We can access static global variable inside same file where it is declared. But we can not access it inside diffrent file. We will get linker error.

```
#include <cstdio>
int main()
{
    extern int Num1;    //Here extern will look for Num1 which is declared
globally
    printf("Global Num1:    %d\n",Num1);
    //extern int Num2;  //NOT OK, static and extern can't be used together
    //printf("Static Global Num1:   %d\n",Num2);
    return 0;
}
int Num1=100;           //Non-Static Global Variable
static int Num2=200;    //Static Global Variable
```

Scope in C/C++

- There are four type of scope in C:
  - Block Scope
  - Function Scope
  - File Scope
  - Function Prototype Scope

```
#include <cstdio>
static int Num3=400;    //File Scope
int Num1=200;       //Program Scope
int Add(int X, int Y)   //Function Prototype Scope
{

}
int main()
{
    int Num2=100;   //Function Scope
    for(int i=0;i<5;i++)    //Block Scope
    {
    }
    return 0;
}
```

- Ref: https://en.cppreference.com/w/c/language/scope
- Scope in C++
    - Block Scope
    - Function Scope
    - File Scope
    - Function Prototype Scope
    - Class Scope
    - Enumuration Scope
    - Program Scope
    - Namespace Scope
- Lifetime of the variables
    - Lifetime describes time i.e how long object will be exist inside memory.
    - Lifetime in C/C++
        - Automatic Lifetime
            - All the local variables are having automatic lifetime.
        - Static Lifetime
            - All the static and global variables are having static lifetime
        - Dynamic Lifetime
            - All the dynamic objects are having dynamic lifetime.

Namespace in C++

- We can not give same name to the multiple variables inside same scope.
- We can give same name to the local variable as well as global variable.
- If name of the local variable and global variable are same then preference will be given to the local variable.
- Example:

```
int num1 = 10; //Global Variable
int main( void ){
 int num1 = 20; //Local variable
 //int num1 = 20; //error: redefinition of 'num1'
 printf("Num1 : %d\n", num1); //20
 return 0;
}
```

- Using scope resolution operator, we can use value of global variable inside program.

```
int num1 = 10; //Global Variable
int main( void ){
 int num1 = 20; //Local variable
 printf("Num1 : %d\n", ::num1); //10
 printf("Num1 : %d\n", num1); //20
 return 0;
}
```

- Example:

```
int num1 = 10; //Global Variable
int main( void ){
 int num1 = 20; //Local variable
 printf("Num1 : %d\n", ::num1); //10
 printf("Num1 : %d\n", num1); //20
 {//Start of block
 int num1 = 30;
 printf("Num1 : %d\n", ::num1); //10
 printf("Num1 : %d\n", num1); //30
 }
 return 0;
}
```

- We can use scope resolution operator with function too.

```
void Show( ){
 printf("Hello C++\n");
}
int main( void ){
 Show( ); //OK
 ::Show( ); //OK
 return 0;
}
```

- Why namespaces?
- Example:

```
int num1 = 10; //OK
int num1 = 20; //error: redefinition of 'num1'
int main( void ){
 int num2 = 30; //OK
 //int num2 = 40; //error: redefinition of 'num2'
 return 0;
}
```

- Namespace is a C++ feature which is designed:
  - to avoid name clashing / conflict / collision / ambiguity.
  - to group/organize functionally equivalent / related types together.
- namespace is a keyword in C++.
- Example 1:

```
namespace na{
 int num1 = 10;
}
int main( void ){
 printf("Num1 : %d\n", na::num1); //OK: 10
 return 0;
}
```

- Example 2:

```
namespace na{
 int num1 = 10;
}
namespace nb{
 int num1 = 20;
}
int main( void ){
 printf("Num1 : %d\n", na::num1); //OK: 10
 printf("Num1 : %d\n", nb::num1); //OK: 20
 return 0;
}
```

- Example 3:

```
namespace na{
 int num1 = 10;
}
namespace na{
 int num2 = 20;
}
int main( void ){
 printf("Num1 : %d\n", na::num1); //OK: 10
 printf("Num1 : %d\n", na::num2); //OK: 20
 return 0;
}
```

- Example 4:

```
namespace na{
 int num1 = 10;
 int num2 = 20;
}
namespace nb{
 int num1 = 30;
 int num3 = 40;
}
```

```
int main( void ){
 printf("Num1 : %d\n", na::num1); //OK: 10
 printf("Num2 : %d\n", na::num2); //OK: 20
 printf("Num1 : %d\n", nb::num1); //OK: 30
 printf("Num3 : %d\n", nb::num3); //OK: 40
 return 0;
}
```

- Example 5:

```
namespace na{
 int num1 = 10;
 int num2 = 20;
}
namespace na{
 //int num1 = 30; //error: redefinition of 'num1'
 int num3 = 30;
}
int main( void ){
 printf("Num1 : %d\n", na::num1); //OK: 10
 printf("Num2 : %d\n", na::num2); //OK: 20
 printf("Num3 : %d\n", na::num3); //OK: 30
 return 0;
}
```

- We can not define namespace inside block scope / function scope or class scope. Namespace definition must appear in either namespace scope or file/program scope.

```
int main( void ){
 namespace na{ //error: namespaces can only be defined in global or
namespace scope
 int num1 = 10;
 }
 return 0;
}
```

- Example 6:

```
int num1 = 10;
//File Scope
namespace na{
 int num2 = 20;
 //Namespace scope
 namespace nb{ //Nested namespace
 int num3 = 30;
 }
}
```

```
int main( void ){
 printf("Num1 : %d\n", ::num1); //10
 printf("Num2 : %d\n", na::num2); //20
 printf("Num3 : %d\n", na::nb::num3); //30
 return 0;
}
```

- If we define variable/function/class without namespace globally then it is considered as a member of global namespace.
- If we dont want to use namespace name and :: operator every time then we should use using directive.
- Example 7:

```
namespace na{
 int num1 = 10;
}
int main( void ){
 using namespace na;
 printf("Num1 : %d\n", num1 );
 return 0;
}
```

- Example 8:

```
namespace na{
 int num1 = 10;
}
int main( void ){
 int num1 = 20;
 using namespace na;
 printf("Num1 : %d\n", num1 ); //20
 printf("Num1 : %d\n", na::num1 ); //10
 return 0;
}
```

- Example 9:

```
namespace na{
 int num1 = 10;
}
namespace nb{
 int num1 = 20;
}
int main( void ){
 using namespace na;
 printf("Num1 : %d\n", num1 ); //10
 using namespace nb;
 //printf("Num1 : %d\n", num1 ); //error: reference to 'num1' is ambiguous
```

```
 printf("Num1 : %d\n", nb::num1 ); //10
 return 0;
}
```

- xample 10:

```
namespace na{
 int num1 = 10;
}
void show_record( ){
 printf("Num1 : %d\n", na::num1);
}
void print_record( ){
 printf("Num1 : %d\n", na::num1);
}
void display_record( ){
 printf("Num1 : %d\n", na::num1);
}
int main( void ){
 ::show_record( );
 ::print_record( );
 ::display_record( );
 return 0;
}
```

- Example 11:

```
namespace na{
 int num1 = 10;
}
using namespace na;
void show_record( ){
 printf("Num1 : %d\n", num1);
}
void print_record( ){
 printf("Num1 : %d\n", num1);
}
void display_record( ){
 printf("Num1 : %d\n", num1);
}
int main( void ){
 ::show_record( );
 ::print_record( );
 ::display_record( );
 return 0;
}
```

- Except main function, we can declare any member inside namespace.

- Example 12:

```
namespace na{
int num1 = 10;
}
using namespace na;
namespace nb{
 void show_record( ){
 printf("Num1 : %d\n", num1);
 }
 void print_record( ){
 printf("Num1 : %d\n", num1);
 }
 void display_record( ){
 printf("Num1 : %d\n", num1);
 }
}
int main( void ){
 nb::show_record( );
 nb::print_record( );
 nb::display_record( );
 return 0;
}
```

- Example 13:

```
namespace na{
 int num1 = 10;
}
int main( void ){
 printf("Num1 : %d\n", na::num1);
 namespace nb = na; //Alias
 printf("Num1 : %d\n", nb::num1);
 return 0;
}
```

To be disscussed tommorow (08-06-2024)

Stream concept

- Standard stream objects associated with console.
- std namespace
- cin, cout, cerr and clog objects

Object Oriented Concepts

- class and object concepts
- characteristics of object.

- default argument
- extern "C"
- enum demo
- Getter and Setter function
- Constructor and its type
- Constructor member initializer list
- Aggregate Type and aggregate initialization
- Array of objects
- Constant variable, data member and member function
- mutable data member.
- Reference
- Call by value vs call by address vs call by reference
- Difference between pointer and reference
- Exception handling in C++