I) make a header file of ext .h

II)make a cpp file and include the header file made using " ".

III)if required make another cpp file and include the header file using " ".

IV)make object file of both the cpp files and then link them together and then compie

C++ Programming

Trainer : Rajiv K

Email: rajiv.kamune@sunbeaminfo.com

Agenda

- static data member
- static member function
- constant data member
- constant (data member, member function, object)
- mutable keyword
- modular Approach
- Composition

static data member

- Only **one copy** of that **member** is **created** for the **entire class** and is shared by **all the objects** of that class, no matter how many objects are created. shared among all object of a class
- It is **initialized before any object** of this class is being created, **even before main() starts.**
- It is **visible only within the class**, but its **lifetime is the entire program**
- **Syntax:**
- static data-type data-member-name; static int num; //declaration
- **Example:**
- static int num1; only declared and not defined but we have to give global initialization using scope resolution optr.
- *static members are **only declared in a class declaration**, **not defined.***
- *They must be **explicitly defined outside the class** using the scope resolution operator*.
- Initialized static and global variable get space on **Data segment**. //Data Section

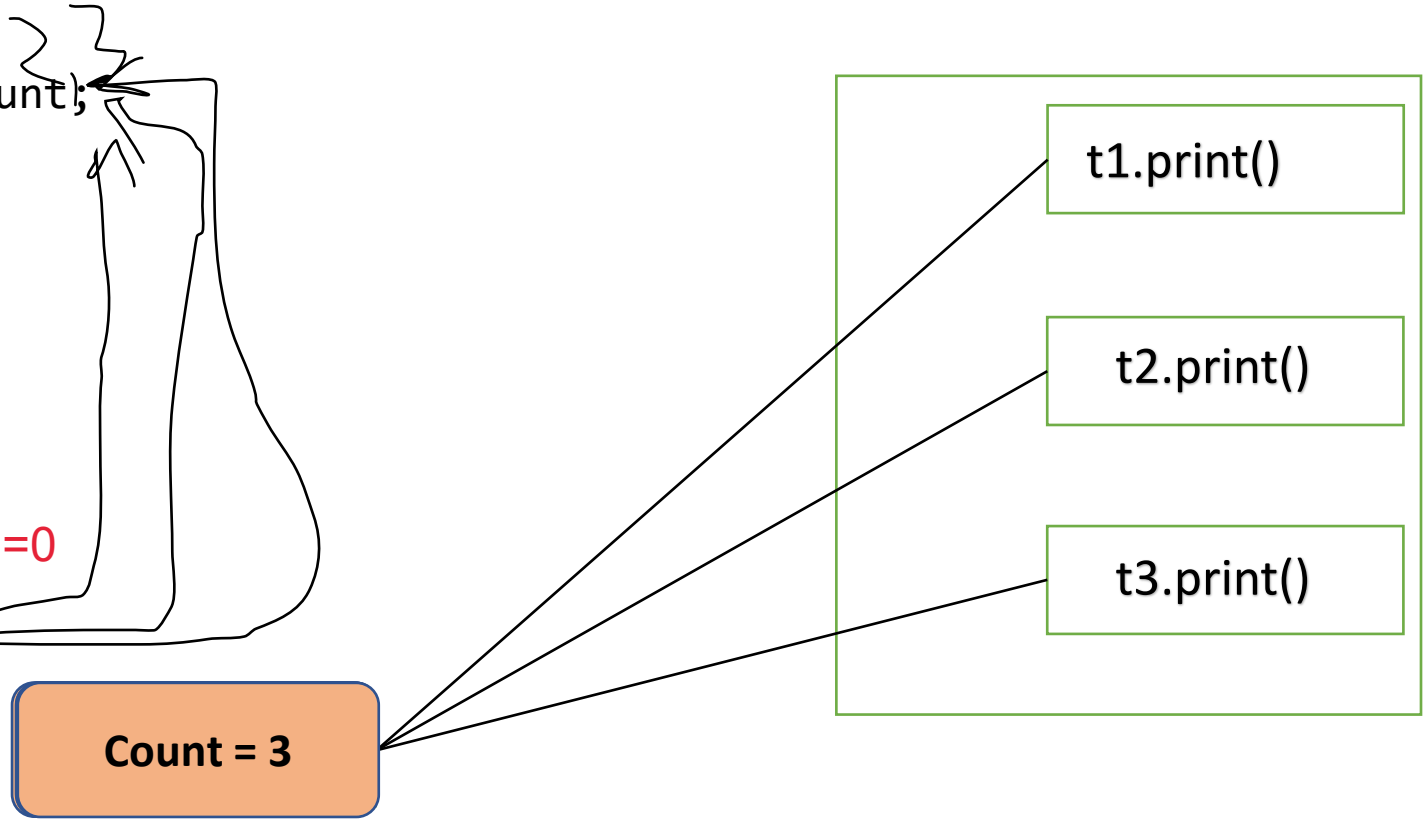
int Test::num=0; //global initialisation //global definition

static int num=0; //not allowed

Static data member

```
1.  class Test
2. {    static int count;
3.      public:
4.              void print()
5.              {
6.                  cout<<"Count:"<<++count;
7.              }
8. };
9. int Test::count=0;
10. int main()
11. {
12.     Test t1,t2,t3;
13.     cout<<"count:"<<Test::count<<endl; =0
14.     t1.print(); =1
15.     t2.print(); =2
16.     t3.print(); =3
17.    // cout<<"Count "<<Test::count;
18.     return 0;
19. }
```

static member = SHARED with every object
non static member= NOT SHARED with every object



static member function of class

- Just like the static data members or static variables inside the class, static member functions also **does not depend on object of class**.
- Static **member functions** are **allowed** to **access** only the **static data members** or other **static member functions**.
- We are allowed to invoke a static member function using the object and the '.' operator but it is **recommended to invoke** the static members using the class name and the **scope resolution operator(::).**

*can only access
1.STATIC DATA MEMBER
2.STATIC MEMBER FUNCTION

- **Syntax:**
- static return-type member-function-name
- { //Function Body};
- **Example:**
- static void print()
- {
- cout<<"Inside print function";
- }

```
static int fun()
{

}
```

<CLASS NAME>: : <STATIC DATA MEMBER>
<CLASS NAME>: : <STATIC MEMBER FUNCTION>

no need of instantization.

static member function of class

- Static member function do not get this pointer.
- You can not declare static member function as virtual.
- You can not declare the static member function as const.

```
class Test
{   static int count;
    public:
        static void print()
        { cout<<"count: "<<++Test::count;}
};
int Test::count=0;
int main()
{
    Test::print();
    return 0;
};
```

constant data member

- If we **do not want to modify value** of the **variable** then const keyword is used.
- constant variable is also called as **read only variable.**

const int x;int y;
inside constructor:
this->x=10//not allowed
this->y=20//allowed

- The value of such variable should be known **at compile time.**

should be initialized at declaration

- In C++ , **Initializing constant variable is mandatory**

- To initialize the const data member of a class we need to use **the member initializer list.**
  - const int num=10; //allowed in C++
  - const int num; // not allowed in C++

constructor() : x(10) //allowed

constant member function of class

- A constant member function of a class that never changes any class data members, and it also does not call any non-const function.
- It is also known as the **read-only function.**
- We can create a **constant member function of a class by adding the const keyword after the name of the member function.**
- We can not declare following function constant:
  1. Global Function
  2. Static Member Function
  3. Constructor
  4. Destructor
- **Syntax**

1)it cannot change value of data member
2)it doesnt call any non const function.

int fun() const

when we declare a function as constant
we use the keyword const at the end
<return type><name>() const

```
1. return_type fuction_name () const  //constant function declaration
2. {
       //Function Body
3. }
```

non constant functions:
++this->x //not allowed
++this->y //allowed

constant function:
++this->x //not allowed
++this->y //not allowed

constant object

- **objects** that need to be **unmodifiable** can be defined as constant objects.
- Member function that operates on **const objects**, **without modifying them**, must be **declared** with the **const** key word.
- The const key word must be specified after the parameter list in function prototype and function definition.
- The **constructor and destructor** of const object **do not need the const** key word.
- When a class contain the const data member it can't be initialized inside the body of constructor, because a const member by definition can not be assigned a new value and C++ dose not allow the initialization in the class definition.
- **Syntax:**
- const class-name object-name;

  const CAR c1;

  c1.print()//allowed if print is const

if we make an OBJECT constant then we can only call CONSTANT FUCTIONS by it.

we can not call non const member function of class

mutable keyword

- **mutable keyword:** mutable is storage **class specifier**
- In case of constant member function you can't change the value of data member through it , it is not allowed in C++.
- Hence mutable keyword comes in picture, **mutable key word allows** to modify the state of **data member through constant member function**.
- **Syntax:**
- mutable data_type variable_name;
- **Example:**
- mutable int num=10;

it is used to change the VALUE of NON CONST DATA MEMBER within CONSTANT MEMBER FUNCTION.

```
mutable int x;

constant function:
    ++this->x//now it is allowed
```

Modular Programming

- If a function or group of function belonging together are put in separate module/source file, there **modular programming** is realized.
- Modules can be **compiled and tested separately** and they can be **imported in to other projects as well.**
- By using modular programming we can **achieve code reusability.**
- If we include header file in **angular bracket** (e.g #include<filename.h>) then **preprocessor try to locate and load header file from standard directory** only(/usr/include).
- If we include header file in **double quotes** (e.g #include“filename.h") then 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.

Modular Programming

- **Advantages Of Modular Programming:**
  - **Ease of Use**
  - **Reusability**
  - **Ease of maintenance**

I) make a header file of ext .h

II)make a cpp file and include the header file made using " ".

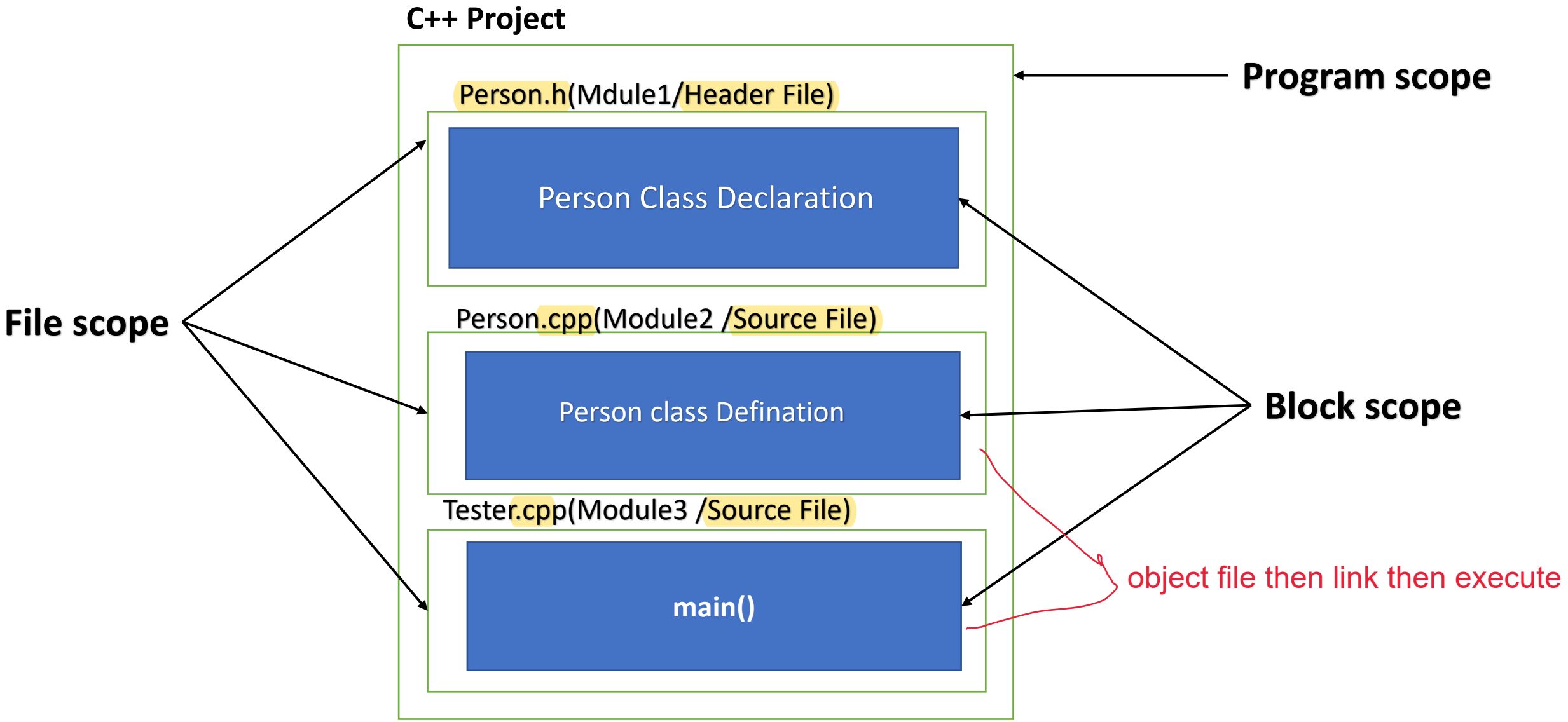
III)if required make another cpp file and include the header file using " ".

IV)make object file of both the cpp files and then link them together and then ~~compile~~ execute

- Header Guard

```
#ifndef HEADER_FILE_NAME_H_
#define HEADER_FILE_NAME_H_
        //TODO : Type declaration here(Class declaration)
#endif
```

Modular Programming Block Diagram



Advantages Of Modular Programming:

- **Ease of Use:** This approach allow simplicity, as rather than focusing on the entire thousand and millions of lines code in one go we can access it in the form of module. This allows ease in debugging the code and prone to less error.
- **Reusability**: It allows the user to reuse the functionality with different interface without typing the whole program again.
- **Ease of maintenance:** It helps in less collision at the time of working on module. Helps the team to work with proper collaboration while working on large application.

Association

- If there is **has-a** relation between the classes, then we should use association.

- If object is a component of another object then it is called as association.

- Example:
  - Person has-a date-of-birth
  - Mobile has-a screen
  - Car has-a engine
- Type of Association
- 1. Aggregation
- 2. Composition

Aggregation

- It is a special form of association whole-part relationship
- If dependency object can exist without dependent object, then it is called as aggregation.
- Consider the following example:
- Department has-a faculty.
- Dependent Object: Department
- Dependency Object: Faculty

Composition

- It is also special form of association
- If dependency object dose not exist without dependent object then it is called as composition

- **Example:**

- Human has-a heart
- Dependent object: Human
- Dependency object: Heart

Thank You