IT206 Data Structures Lab with OOP

Lecture 2

Rachit Chhaya

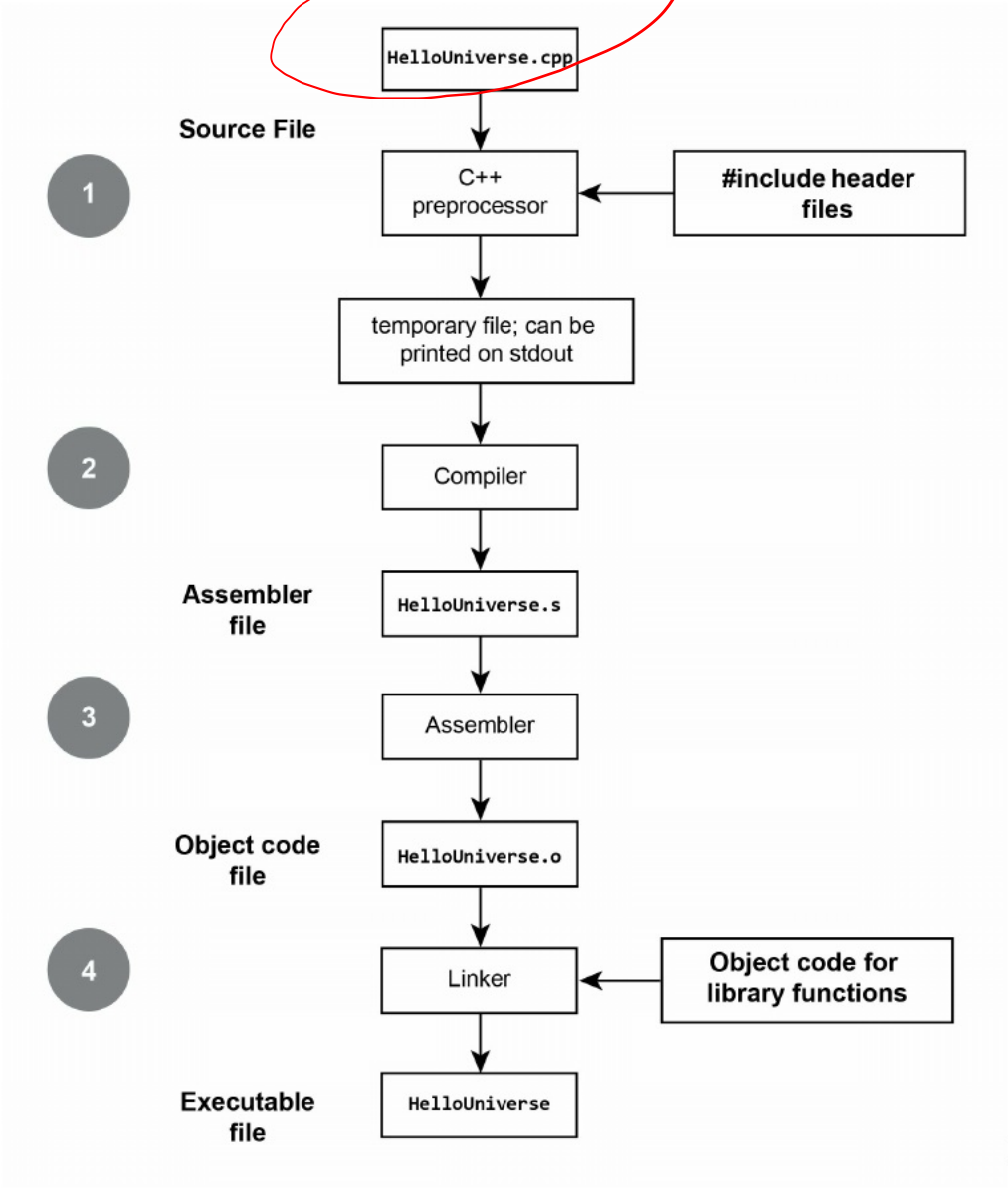
DA-IICT

April 4, 2022

Topics

- Overview of Compilation Process
- Pass By Reference
- Access Specifiers
- Friend Function
- Constructors and Destructors

The C++ compilation process



[1]

[1] https://subscription.packtpub.com/book/programming/9781789801491/1/ch01lvl1sec03/the-c-compilation-model

Pass By Reference

C++ allows the functionality to pass an argument by reference. Consider the following code

Pass By Reference

C++ allows the functionality to pass an argument by reference. Consider the following code

```
#include <iostream>
using namespace std;
void my_function(int &x) {
  x = 50;
  cout << "Value of x from my_function: " << x << endl;
}
int main() {
  int x = 10;
  my_function(x);
  cout << "Value of x from main function: " << x;
  return 0;
}
```

pass a reference

50

10

50

- In Pass by Value copy of variable is created and changes are made to the copy.

- In Pass by Value copy of variable is created and changes are made to the copy.
- Pass by reference : allows a function to modify a variable without having to create a copy of it.

- In Pass by Value copy of variable is created and changes are made to the copy.
- Pass by reference : allows a function to modify a variable without having to create a copy of it.
- Similar objective can by passing using pointers. Differences?
- Also possible to return by reference

— NULL Values

— Pointers can be reassigned

int &a

Access Specifiers/ Modifiers in C++

Visibility levels

Default

- Three kinds: 1)Public 2) Private 3) Protected

Access Specifiers/ Modifiers in C++

- Three kinds: 1)Public 2) Private 3) Protected
- **public**: members are accessible from outside the class. can be accessed by other classes and functions too and anywhere in program using dot operator.

```
class circle
{
Private
    int radius;
Public
    area
        pi
}
```

```
class myclass {
Public:
    int a;
Private
    int b;
}
myclass A;
A.a=
A.b=
```

Access Specifiers/ Modifiers in C++

DAperson ← Base class / Parent

DAFaculty — DA student

child class / Derived class

PG UG

- Three kinds: 1)Public 2) Private 3) Protected
- **public**: members are accessible from outside the class. can be accessed by other classes and functions too and anywhere in program using dot operator.
- **private**: default type. members cannot be accessed (or viewed) from outside the class. member functions or the friend functions are allowed to access
- **protected**: Similar to private. but can also be accessed by child or derived class.

Inheritance

scope

prepare result()
{ getscores()
}

ABC.getscores()
↑ private ✗

Friend Function

- A function "friendly" to some class has access to private members of that class.

int a;

main

{ int a;

a

}

sum

{ a

}

Friend Function

- A function "friendly" to some class has access to private members of that class.
- Need not be a member of the class.

```
// Taken from tutorials point
#include <iostream>

using namespace std;

class Box {
   double width;

   public:
      friend void printWidth( Box box );
      void setWidth( double wid );
};
```

Scope Resolution

Jw ay

```
// Member function definition
void Box::setWidth( double wid ) {
    width = wid;
}

// Note: printWidth() is not a member function of any class
void printWidth( Box box ) {
    /* Because printWidth() is a friend of Box, it can
    directly access any member of this class */
    cout << "Width of box : " << box.width <<endl;
}
```

```
// Main function for the program
int main() {
   Box box;

   // set box width without member function
   box.setWidth(10.0);

   // Use friend function to print the wdith.
   printWidth( box );

   return 0;
}
```

object of type box

? valid ← public

not using . operator

More about Friend functions

- Not in scope of class it is declared friend with. cannot use object to invoke.
- has to use object and dot operator to access members of class
- Usually has object as arguments

class student
{
Friend printdetails
}

class Teacher
printdetails(Teacher t, student s);
s.name , t.age
Friend printdetails
};

Constructors and Destructors

- Constructor: Special Member function with same name as the class.

Constructors and Destructors

- Constructor: Special Member function with same name as the class.
- Used to initialize class objects

student S

Constructors and Destructors

- Constructor: Special Member function with same name as the class.
- Used to initialize class objects
- Invoked whenever an object of associated class created.

Examples (Taken from Balagurusamy)

```
class integer
{
        int m, n;
    public:
        integer(void);          // constructor declared
        .....
        .....
};
integer :: integer(void)        // constructor defined
{
        m = 0; n = 0;
}
```

declaration

Same name

Parametrized Constructors

integer i1;

cout << i1.m;

Polymorphism
↓
Function overloading

```
class integer
{
        int m, n;
  public:
        integer(int x, int y);   // parameterized constructor
        .....
        .....
};
integer :: integer(int x, int y)
{
        m = x; n = y;
}
```

arguments

class integer
{
int m, n;
public
integer (void)
integer (int x, int y)
integer (int x,)

main
{
i1;
integer i2(10, 20);
integer i3 = integer (30, 40);

Implicit calling
Explicit calling
→ integer

int a = 5;
int a;
Assignment
a = 7;
i) Is it valid?
No
D = A;
operator overloading

Copy Constructor

- The copy constructor in C++ is used to copy data of one object to another.
- Initialization by copy constructor is copy initialization
- Takes reference variable as argument

int a, b
a + b = sum
a = "37"
b = "206"
a + b =

```
code(code & x)

{
        id = x.id;
}
```

```
code A(100);   // object A is created and initialized
code B(A);     // copy constructor called
code C = A;    // copy constructor called again

code D; // D is created, not initialized
D = A;         // copy constructor not called
```

Destructor

- Used to destroy objects created by constructor
- Same name as class preceded by a tilde
  ~integer(){}
- No arguments and return value

destructor

end of function

~integer ( ) {
cout<<"Destroying";
}

a.~integer()