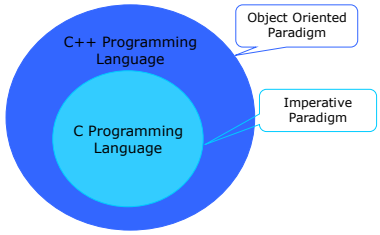
Preview

- Object Oriented Program with C++
- Class Declarations in C++
- Class Members
- Inline Function vs. Regular Function
- Constructor & Destructor
- Assessing a Class Member
- Initializing class Objects with constructor

Object Orient Program with C++



C++ is a super set of C

Object Orient Program with C++

Imperative Programming

- **action oriented**: a computation is viewed as a sequence of actions (instructions). – In this paradigm, a program is assumed to be a sequence of detailed statements (commands), instructing the computer exactly how a task is to be accomplished.
- **The word imperative means command**. - Sequential imperatives effect changes to the contents of internal storage locations by assignments.
- This paradigm promotes **sharp distinctions between program and data**. – They are heavily concerned with variables tied to main storage location and with assignment of values to these variables.
- Supported languages - Fortran (scientific calculation), Algol60, Pascal (teaching language), C (UNIX), COBOL.

Object Orient Program with C++

Object-Oriented Programming

- Attempt of grouping variables (data) and procedures (operations) together - class
- Individual data objects are defined by modules of code, called **class**, which encapsulate data declarations and functionalities (algorithms) for the operation.
- Supported languages – Simila, C++, Java, C#

Object Orient Program with C++

Goal : Kick a soccer ball with procedural programming

1. call a function called "leg" to kick the soccer ball
2. Kick the soccer ball

Object Orient Program with C++

Goal : Kick a soccer ball with object-oriented programming

1. Create a human object to kick the ball
2. Kick the soccer ball

Object Orient Program with C++

The characteristics of OO programming

1. Information hiding (encapsulation)
2. Data abstraction
3. Message passing (polymorphism)
4. Dynamic binding
5. Inheritance

Class Declarations in C++

- Classes in C++ are a generalization of record, called structures (struct) in C and C++.
- A class in C++ is an encapsulation of data members and functions that manipulate the data. .

Class Declarations in C++

```
class Time
Private:
    int hour;
    int minutes;
    int seconds;
Public:
    Time (int, int, int)
    ~Time();
    void SetTime(int, int, int);
    void PrintTime();
```


Class Declarations in C++

(Members in a Class)

Types of Member in a Class

- **Public member** – can be accessed by any function in the program.
- **Private member** – can be accessed only by member functions (and friends)
- **Protected member** – can be accessed by member functions and to derived class – inheritance

Class Declarations in C++

(Public, Private, and Protected Member)

```
class Time{
private:
      int hour;
      int min;
      int sec;
public:
      Time(){hour=min=sec=0;};
      void SetTime(int, int, int);
      void Printtime(Time &);
      ~Time(){;};
};
```


Class Declarations in C++

- **Data Members** – The data members can be of
  - a primitive data type ,
  - a pointer type,
  - a structured data type or
  - a class type.

Class Declarations in C++

Function Members – Function can be defined inside the class or outside the class.

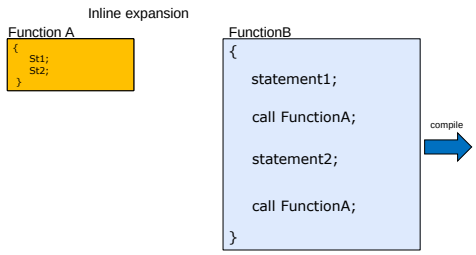
- If a function is defined outside a class structure, the function prototype must be in the class.
- If a function is defined inside the class, system consider it as a **inline function**.
- We can define a inline function outside the class with inline

Class Declarations in C++
(Inline Function)

What is inline function?

- An **inline function** is a programming language construct used to suggest to a compiler that a particular function be subjected to in-line expansion.
- A in-line expansion is that the compiler insert the complete body of the function in every context where that function is called.

Class Declarations in C++
(Inline Function)



Class Declarations in C++
(Inline Function)

Do inline-function improve performance?

- **inline functions might make it *faster*:** Procedural integration by in-line function might remove a bunch of unnecessary function calls
- **functions might make it *slower*:** Too much in-line expansion might cause code bloat, which might cause "thrashing" on demand-paged virtual-memory systems.

Class Declarations in C++
(Define a Member Function)

A Member Function Defined in a Class

```
class Time{
private:
        int hour;
        int min;
        int sec;
public:
        Time(){hour=min=sec=0;};
        void SetTime(int h int m int s)
        {
           hour = h;
           min = m;
           sec = s;
        };
        void Printtime(Time &);
        ~Time(){;};
};
```


Class Declarations in C++
(Define a Member Function)

A Member Function Defined outside a Class

```
Return_Type Class_Name:: FunctionName (Parameter_List)
{
     Statements
}
```

Class Declarations in C++
(Define a Member Function)

A Member Function Defined outside a Class

```
class Time{
private:
        int hour;
        int min;
        int sec;
public:
        Time();
        void SetTime(int, int, int);
        void Printtime(Time &);
        ~Time(){;};
};
void Time::SetTime(int h, int m int s)
{
        hour = h;
        min = m;
        sec = s;
}
```

```
Time::Time()
{
    hour=min=sec=0;
}

void Time::printtime(Time &t)
{
    cout << t.hour << ":" << t.min << ":" << t.sec;
}
```


Constructor and Destructor in a Class

Constructor

- A **Constructor** is a special function which existed in every class without any return type.
- It is used for memory allocation for each object.
- A constructor has same name as a class name.
- A constructor is called automatically when an instance of a class is created.
- If a constructor is not defined by a programmer, C++ invokes a default constructor, which allocates memory for the object, but without any initialization.

Constructor and Destructor in a Class

Destructor

- A destructor is usually used to deallocate memory space and do other cleanup for a class object and its class members when the object is destroyed.
- A destructor has same name as a class preceded by ~.
- A destructor is called automatically to clean up just before the lifetime of an object end if the object is created by declaration.
- If an object is created dynamically during runtime, to remove object from memory, need call delete() function to call destructor.
- If no user-defined destructor exists for a class, the compiler implicitly declares a destructor.

Accessing a Class Member

Accessing Class Member

- There are two ways to access member of object.
  - Through dot operator

```
Time Now(); //create a instance of the class Time
Now.SetTime(1, 2, 3); // call public member function
```

  - Through pointer

```
Time Now(); //create a instance of the class Time
Time *TPtr; //create a pointer to a Time object
TPtr = Now; //Assign pointer to the reference to Now
Tptr->SetTime(1, 2, 3); // call public member function
```


Accessing a Class Member

```
#include <iostearm>
using namespace std;
int main ()
{
  Time Now;
  int h, m, s;
  cout << "What is Current time? (hour min sec)";
  cin >> h >> m >>s;
  cout <<endl;
  Now.SetTime(h,m,s);
  Now.Printtime(Now);

  return 0;
}
```


Accessing a Class Member

```
//What is wrong with following program
#include <iostearm>
using namespace std;

int main ()
{
  Time Now;
  cout << "What is Current time? (hour min sec)";
  cin >> Now.hour >> Now.min >>Now.sec;
  cout <<endl;
  Now.Printtime(Now);

  return 0;
}
```

Initializing class objects with constructor

- Initialize a class by a constructor without parameter
- Initialize a class by a constructor with parameter
  - No default argument with constructor
  - Using default arguments with constructor

Initializing class objects with constructor
(without default arguments)

```
//initiate object with constructor
#include <iostream>

using namespace std;

class Two_Num{
public:
    int one;
    int two;
    int add(int x, int y) {return (x+y);};
    Two_Num(int a, int b) {one = a; two = b;};
};


int main()
{
    Two_Num x(2,3);

    cout << x.one << x.two<<endl;
    return 0;
}
```


Initializing class objects with constructor
(without default arguments)

```
//initiate object with constructor
#include <iostream>
using namespace std;

class Two_Num{
public:
    int one;
    int two;
    int add(int x, int y) {return (x+y);};
    Two_Num(int = 0, int =0);
};
Two_Num::Two_Num(int a, int b)
{
    one = a;
    two = b;
}

int main()
{
    Two_Num x(2,3);
    Two_Num y;
    cout << x.one << x.two<<endl;
    cout << y.one << y.two<<endl;
   return 0;
}
```


Initializing class objects with constructor
(With Default Arguments)

```
// time2.h
// Declaration of the Time class.
// Member functions are defined in time2.cpp

#ifndef TIME2_H
#define TIME2_H

// Time abstract data type definition
class Time {
public:
   Time( int = 0, int = 0, int = 0 );  // default arguments
   void setTime( int, int, int ); // set hour, minute, second
   void printMilitary();          // print military time format
   void printStandard();          // print standard time format
private:
   int hour;     // 0 - 23
   int minute;   // 0 - 59
   int second;   // 0 - 59
};

#endif
```



```
// time2.cpp
// Member function definitions for Time class.
#include <iostream>

using namespace std;

#include "time2.h"

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time( int hr, int min, int sec )
   { setTime( hr, min, sec ); }

// Set a new Time value using military time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
{
   hour   = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

// Print Time in military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
}

// Print Time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
}
```



```
// Time.cpp
// Demonstrating a default constructor
// function for class Time.
#include <iostream>
#include "time2.h"
using namespace std;

int main()
{
   Time t1,            // all arguments defaulted
        t2(2),         // minute and second defaulted
        t3(21, 34),    // second defaulted
        t4(12, 25, 42), // all values specified
        t5(27, 74, 99); // all bad values specified

   t1.printMilitary();
   cout << endl;
   t1.printStandard();

   t2.printMilitary();
   cout << endl;
   t2.printStandard();

   t3.printMilitary();
   cout << endl;
   t3.printStandard();

   t4.printMilitary();
   cout << endl;
   t4.printStandard();

   t5.printMilitary();
   cout << endl;
   t5.printStandard();
   cout << endl;

   return 0;
}
```