Preview

- Scope of Member
- Utility Functions
- When Constructor and Destructor is called?
- Insecure Public Functions
- Assessing a Class Member
- Initializing class Objects with constructor

Scope of Members

```
// Time abstract data type definition
class Time {
public:
   Time();                           // constructor
   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
};
```

- Member function printMilitary and printStandard takes no arguments. Why?
  - Since the function and variables are members in the same scope

Utility Functions

- Usually public functions in a class are worked for interface of the class
- But, not all member functions need to be made public.
- We can define private functions to serve as utility functions to other function in the class.

Utility Functions

```
// salesp.h
// SalesPerson class definition
// Member functions defined in salesp.cpp
#ifndef SALESP_H
#define SALESP_H

class SalesPerson {
public:
   SalesPerson();                // constructor
   void getSalesFromUser(); // get sales figures from keyboard
   void setSales( int, double ); // User supplies one month's
                                 // sales figures.
   void printAnnualSales();

private:
   double totalAnnualSales();       // utility function
   double sales[ 12 ];           // 12 monthly sales figures
};

#endif
```


Utility Functions

```
// salesp.cpp
// Member functions for class SalesPerson
#include <iostream>
#include <iomanip>
#include "salesp.h"
using namespace std;

// Constructor function initializes array
SalesPerson::SalesPerson()
{
   for ( int i = 0; i < 12; i++ )
      sales[ i ] = 0.0;
}

// Function to get 12 sales figures from the user
// at the keyboard
void SalesPerson::getSalesFromUser()
{
   double salesFigure;

   for ( int i = 1; i <= 12; i++ ) {
      cout << "Enter sales amount for month " << i << ": ";
      cin >> salesFigure;
      setSales( i, salesFigure );
   }
}
```

```
// Print the total annual sales
void SalesPerson::printAnnualSales()
{
   cout << setprecision( 2 )
        << "The total annual sales are: $"
        << totalAnnualSales() << endl;
}

// Private utility function to total annual sales
double SalesPerson::totalAnnualSales()
{
   double total = 0.0;

   for ( int i = 0; i < 12; i++ )
      total += sales[ i ];

   return total;
}
```


Utility Functions

```
// Salesperson.cpp
// Demonstrating a utility function
// Compile with salesp.cpp, and salesp.h
#include "salesp.h"

int main()
{
   SalesPerson s;          // create SalesPerson object s

   s.getSalesFromUser();  // note simple sequential code
   s.printAnnualSales();  // no control structures in main
   return 0;
}
```

When Constructor and Destructor are called

- A constructor is called for a class object when object is created for memory space.
- A destructor is called for a class object when that object passes out of scope or is explicitly deleted.
- A destructor is a member function with the same name as its class prefixed by a ~ (tilde).

When Constructor and Destructor are called

```
// pro1.h
// compile with pro1.cpp and pro11.cpp
#ifndef PRO_H
#define PRO_H
//function prototypes for function f1 and f2
void f1();
void f2();

// class test definded
class test {
public:
        int a;
        test(int);
        ~test();
};

#endif
```


When Constructor and Destructor are called

```
//define member functions for class test
// in pro.h
#include <iostream>
#include "pro.h"
using namespace std;
test::test(int x) // constructor
{
        a = x;
        cout <<"test object "<< a<< " is created "<< endl;
}
test::~test() // destructor
{
        cout <<"Test object " << a << " is destroyed"<<endl;
}
```


When Constructor and Destructor are called

```
// pro1.cpp
// Example shows when a constructor and destructor is called

#include <iostream>
#include "pro1.h"
using namespace std;

int main()
{
        f1();
        f2();
        return 0;
}

// function f1 create a test object and call function f2
void f1()
{
        test w(1);
        f2();
}
//function f2 create a test object
void f2()
{
        test aa(2);
}
```


Insecure Public Function

- Programmer must not define a public function which returns a reference to a private member in the class.
- It create a "security hole".
- If a public member function returns a reference to a private member, a private member can be modified from the outside of the class

Insecure Public Function

```
// time.h
// Declaration of the Time class.
// Member functions defined in time4.cpp

#ifndef TIME4_H
#define TIME4_H

class Time {
public:
   Time( int = 0, int = 0, int = 0 );
   void setTime( int, int, int );
   int getHour();
   int &badSetHour( int );  // A insecure public function
private:
   int hour;
   int minute;
   int second;
};

#endif
```

Insecure Public Function

```
// time.cpp
// Member function definitions for Time class.
#include "time.h"

// Constructor function to initialize private data.
// Calls member function setTime to set variables.
// Default values are 0 (see class definition).
Time::Time( int hr, int min, int sec )
   { setTime( hr, min, sec ); }

// Set the values of hour, minute, and second.
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;
}

// Get the hour value
int Time::getHour() { return hour; }

// Insecure public function
// Returning a reference to a private data member.
int &Time::badSetHour( int hh )
{
   hour = ( hh >= 0 && hh < 24 ) ? hh : 0;

   return hour;  // DANGEROUS reference return
}
```


Insecure Public Function

```
// insecure.cpp (with time.cpp and time.h)
// Demonstrating a public member function that
// returns a reference to a private data member.
// Time class has been trimmed for this example.
#include <iostream>
#include "time4.h"
using namespace std;
int main()
{
   Time t;
   int *hourRef;
   hourRef = &t.badSetHour( 20 );

   cout << "Hour before modification: " << t.getHour() <<endl;
   // modification private value result of a insecure function
   *hourRef = 30;
   cout << "Hour after modification: " << t.getHour()<<endl;

   return 0;
}
```


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
(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;
}
```