Nov 02, 2015

Computer Programming

LAB 8

ABDUL AZIZ

Lab Instructor	**Abdul Aziz**
Course	**Computer Programming Lab**
Duration	**2hrs**

Objectives:

In this lab, following topics will be covered:

- Introduction to Pointers
- Declaring Pointer variables
- Initializing Pointers
- Manipulating Data with Pointers
- Pointer has a type Too
- Dynamic Variables
- Dynamic variable of class type
- Dynamic memory allocations
  - new
  - delete
- Passing Pointers as parameters
- Array and Pointer variables
- Pointer to C++ classes

1. Pointers

Pointer is a variable that stores a memory address. Every variable is located under unique location within a computer's memory and this unique location has its own unique address, the memory address. Normally, variables hold values such as 5 or "hello" and these values are stored under specific location within computer memory. However, **pointer is a different beast, because it holds the memory address as its value and has an ability to "point" (hence pointer) to certain value within a memory, by use of its associated memory address.**

1.1 Retrieving a Variable's Memory Address

```
#include <iostream>
using namespace std;
int main()
{
                // Declare an integer variable and initialize it with 99
  int myInt = 99;
                // Print out value of myInt
  cout << myInt << endl;
                // Use address-of operator & to print out
                // a memory address of myInt
  cout << &myInt << endl;

return 0;
}
```

1.2 Assigning a Variable's Memory Address to a Pointer

```
#include <iostream>
using namespace std;
int main()
{
 // Declare and initialize a pointer.
   int * pPointer = 0;
// Declare an integer variable and initialize it with 35698
   int twoInt = 35698;
// Declare an integer variable and initialize it with 77
   int oneInt = 77;
// Use address-of operator & to assign a memory address of twoInt to a pointer
  pPointer = &twoInt;
// Pointer pPointer now holds a memory address of twoInt

// Print out associated memory addresses and its values
  cout << "pPointer's memory address:\t\t" << &pPointer << endl;
  cout << "Integer's oneInt memory address:\t" << &oneInt << "\tInteger
value:\t" << oneInt << endl;
  cout << "Integer's twoInt memory address:\t" << &twoInt << "\tInteger
value:\t" << twoInt << endl;
  cout << "pPointer is pointing to memory address:\t" << pPointer <<
"\tInteger value:\t" << *pPointer << endl;
return 0;
}
```

1.3 Initializing the Pointer via the Address-Of Operator (&)

When you declare a pointer variable, its content is not initialized. In other words, it contains an address of "somewhere", which is of course not a valid location. This is dangerous! You need to initialize a pointer by assigning it a valid address. This is normally done via the **address-of operator (&).**

The **address-of operator (&)** operates on a variable, and returns the address of the variable. For example, if number is an int variable, &number returns the address of the variable number.

You can use the address-of operator to get the address of a variable, and assign the address to a pointer variable.

For example,

```
int number = 88; // An int variable with a value
int * pNumber;   // Declare a pointer variable called pNumber pointing to an
int (or int pointer)
pNumber = &number; // Assign the address of the variable number to
pointer pNumber
int * pAnother = &number // Declare another int pointer and init to address
of the variable numbe
```

Accessing the Value at the Memory Address held by a Pointer

```
#include <iostream>
 using namespace std;
int main()
{
                // Declare an integer variable and initialize it with 99
   int myInt = 99;
               // Declare and initialize a pointer
   int * pMark = 0;
               // Print out a value of myInt
  cout << myInt << endl;
               // Use address-of operator & to assign a memory address
               of myInt to a pointer
  pMark = &myInt;
               // Dereference a pMark pointer with dereference operator
               * to access a value of myInt
  cout << *pMark << endl;
return 0;
}
```

Manipulating Data with Pointers

```
#include <iostream>
 using namespace std;
int main()
{
              // declare an integer variable and initialize it with 99
   int myInt = 99;
              // declare and initialize a pointer
   int * pMark = 0;
              // Print out a value of myInt
  cout << myInt << endl;
             // Use address-of operator & to assign memory address of
             myInt to a pointer
  pMark = &myInt;
             // dereference a pMark pointer with dereference operator * and
             set new value
  *pMark = 11;
              // show indirectly a value of pMark and directly the value of
        myInt
  cout << "*pMark:\t" << *pMark << "\nmyInt:\t" << myInt << endl;

return 0;
}
```

Pointers has a type Too

A pointer is associated with a type (of the value it points to), which is specified during declaration. A pointer can only hold an address of the declared type; it cannot hold an address of a different type.

```
int i = 88;
double d = 55.66;
int * iPtr = &i;    // int pointer pointing to an int value
double * dPtr = &d; // double pointer pointing to a double value

iPtr = &d;   // ERROR, cannot hold address of different type
dPtr = &i;   // ERROR
iPtr = i;    // ERROR, pointer holds address of an int, NOT int value

int j = 99;
iPtr = &j;  // You can change the address stored in a pointer
```

Test Example of Pointers

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

int main() {
   int number = 88;    // Declare an int variable and assign an initial value
   int * pNumber;      // Declare a pointer variable pointing to an int (or int pointer)
   pNumber = &number;  // assign the address of the variable number to pointer pNumber

   cout << pNumber << endl;  // Print content of pNumber (0x22ccf0)
   cout << &number << endl;  // Print address of number (0x22ccf0)
   cout << *pNumber << endl; // Print value pointed to by pNumber (88)
   cout << number << endl;   // Print value of number (88)

   *pNumber = 99;            // Re-assign value pointed to by pNumber
   cout << pNumber << endl;  // Print content of pNumber (0x22ccf0)
   cout << &number << endl;  // Print address of number (0x22ccf0)
   cout << *pNumber << endl; // Print value pointed to by pNumber (99)
   cout << number << endl;   // Print value of number (99)
                             // The value of number changes via pointer
    cout << &pNumber << endl; // Print the address of pointer variable pNumber (0x22ccec)
}
```

Dynamic Variables

- Variables that are created using the new operator are called **dynamically allocated variables** or simply **dynamic variables** , because they are created and destroyed while the program is running.

```
//Program to demonstrate pointers and dynamic variables.
#include <iostream>
using namespacestd;

int main( )
{
   int *p1, *p2;
   p1 = new int;
   *p1 = 42;
   p2 = p1;
   cout << "*p1 == " << *p1 << endl;
   cout << "*p2 == " << *p2 << endl;

   *p2 = 53;
   cout << "*p1 == " << *p1 << endl;
   cout << "*p2 == " << *p2 << endl;

   p1 = new int;
   *p1 = 88;
   cout << "*p1 == " << *p1 << endl;
   cout << "*p2 == " << *p2 << endl;

   cout << "Hope you got the point of this example!\n";
   return 0;
}
```

```
Output

*p1 == 42
*p2 == 42
*p1 == 53
*p2 == 53
*p1 == 88
*p2 == 53
Hope you got the point of this example!
```

Dynamic Variables of Class Type

- When the new operator is used to create a dynamic variable of a class type, a constructor for the class is invoked. If you do not specify which constructor to use,
  the default constructor is invoked.

```
SomeClass *classPtr;
classPtr = new SomeClass; //Calls default constructor
```

- If the type is a class type, the default constructor is called for the newly created dynamic variable. You can specify a different constructor by including arguments as follows:

```
   MyType *mtPtr;
mtPtr = new MyType(32.0, 17); // calls MyType(double, int);
```

Dynamic Memory Allocation

- **new**

- When the new operator is used to create a dynamic variable of a class type, a constructor for the class is invoked. If you do not specify which constructor to use,
  the default constructor is invoked.

  delete

  - The delete operator eliminates a dynamic variable and returns the memory that the dynamic variable occupied to the freestore

```
delete p;
```

Passing pointers to functions in C++

C++ allows you to pass a pointer to a function. To do so, simply declare the function parameter as a pointer type.

```
#include <iostream>
#include <ctime>
using namespace std;
void getSeconds( long *par);
int main ()
{
   long sec;
  getSeconds( &sec );
                  // print the actual value
  cout << "Number of seconds :" << sec << endl;
  return 0;
}
void getSeconds( long *par)
{
                  // get the current number of seconds
  *par = time( NULL );//here NULL is just like required parameter for time from <ctime> library
  return;
}
```

Another Example

```
#include<iostream>
using namespace std;
double CalculateNetPrice(double *);

int main()
{
   double FinalPrice;
   double Discount = 20.00;

   FinalPrice = CalculateNetPrice(&Discount);

   cout<<"After applying a 20% discount";
   cout<<"Final Price = "<<FinalPrice;

   return 0;
}

double CalculateNetPrice(double *Discount)
{
   double OrigPrice;

   cout<<"Enter the original price: ";
   cin>>OrigPrice;

   return OrigPrice - (OrigPrice * *Discount / 100);
}
```

Array and Pointer variables

The function which can accept a pointer, can also accept an array

```
#include <iostream>
using namespace std;
                // function declaration:
double getAverage(int *arr, int size);
 int main ()
{
                 // an int array with 5 elements.
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;
                // pass pointer to the array as an argument.
   avg = getAverage( balance, 5 ) ;
                // output the returned value
   cout << "Average value is: " << avg << endl;
      return 0;
}
```

```
double getAverage(int *arr, int size)
{
  int    i, sum = 0;
  double avg;

  for (i = 0; i < size; ++i)
  {
    sum += arr[i];
   }
   avg = double(sum) / size;
   return avg;
}
```

Another Example

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

int main ()
{
  // an array with 5 elements.
  array <int,5> balance = {100, 2, 3, 17, 5};
  int *p;

  p = balance.data();

  // output each array element's value
  cout << "Array values using pointer " << endl;
  for ( int i = 0; i < 5; i++ )
  {
     cout << "*(p + " << i << ") : ";
     cout << *(p + i) << endl;
  }


  return 0;
}
```

Pointer to C++ Classes

```
#include <iostream>

using namespace std;

//Box.h

class Box
{
  public:
```

```
    // Constructor definition
    Box(double l, double b, double h);
    double Volume();

      private:
    double length;     // Length of a box
    double breadth;    // Breadth of a box
    double height;     // Height of a box
};

//Box.cpp
  Box::Box(double l, double b, double h)
    {
      cout <<"Constructor called." << endl;
      length = l;
      breadth = b;
      height = h;
    }
   double Box:: Volume()
    {
      return length * breadth * height;
    }
 //Main.cpp
int main(void)
{
  Box Box1(3.3, 1.2, 1.5);    // Declare box1
  Box Box2(8.5, 6.0, 2.0);    // Declare box2
  Box *ptrBox;              // Declare pointer to a class.

  // Save the address of first object
  ptrBox = &Box1;

  // Now try to access a member using member access operator
  cout << "Volume of Box1: " << ptrBox -> Volume() << endl;

  // Save the address of first object
  ptrBox = &Box2;

  // Now try to access a member using member access operator
  cout << "Volume of Box2: " << ptrBox->Volume() << endl;

  return 0;
}
```

```
//Using Dynamic Class Variable
#include <iostream>

using namespace std;

//Box.h

class Box
{
  public:
    // Constructor definition
    Box(double l, double b, double h);
    ~Box();
    double Volume();
```

```
        private:
     double length;     // Length of a box
     double breadth;    // Breadth of a box
     double height;     // Height of a box
};

//Box.cpp
  Box::Box(double l, double b, double h)
     {
       cout <<"Constructor called." << endl;
       length = l;
       breadth = b;
       height = h;
     }
   double Box:: Volume()
     {
       return length * breadth * height;
     }
     Box::~Box(){
        cout<<"Calling Destructor... ";
          }
 //Main.cpp
int main(void)
{
  Box *ptrBox = new Box(3.3, 1.2, 1.5);              // Declare pointer to a class.
  // Now try to access a member using member access operator
  cout << "Volume of Box1: " << ptrBox -> Volume() << endl;
  delete ptrBox;
  return 0;
}
```

Exercise