Lecture 12.1

Topics

1. Pointers – Introduction

---

1. Pointers – Introduction

A pointer variable is a **derived variable** that holds the address of a stored value (data) with a given (or known) data type.

With only one exception of a **`void`** pointer, a pointer variable would only hold address of one and only one type.

The syntax of a declaration is given as follows,

```
Type* ptrVariableName;
```

where

- `Type` is any valid type such as `int`, `double`, `char`, etc.
- **`*`** is the indirect operator.
- `ptrVariableName` is any valid variable name.

For examples,

```
int* iPtr{nullptr};
double* dPtr{nullptr};
```

In these declarations, `iPtr` is a pointer variable that points to an `int` (that means a value of type `int`), and `dPtr` is a pointer variable that points to a `double`.

The values that can be assigned to `iPtr` are addresses. Each of these addresses must be the address of a memory block that was reserved for (i.e., to hold) a value of type `int`.

1.1 Initializing a Pointer Variable

Frome here on, the words of "a pointer variable" may be shortened as just "a pointer or pointer".

The initialization of a pointer may be done at the time of declaration. This may be done in several forms as given in the following examples.

Example 1

```
/**
 * Program Name: cis25L1211.cpp
 * Discussion:   Pointer Initialization
 */

// Include/Header file(s)
#include <iostream>
using namespace std;

// Application Driver
int main() {
  int iVar{5};
  int* iPtr1{nullptr};   /* Initializing with predefined value of
```

```
                                nullptr*/
  int* iPtr2{iPtr1};  /* Initializing with a known value from another
                          variable*/
  int* iPtr3{&iVar};  /* Initializing with a known value based on type
                         definition*/

  cout << "Value of iPtr1 : " << iPtr1
       << "\nValue of iPtr2 : " << iPtr2
       << "\nValue of iPtr3 : " << iPtr3 << endl;

  cout << "\nAddress of iVar : " << &iVar
       << "\nAddress of iPtr1 : " << &iPtr1
       << "\nAddress of iPtr2 : " << &iPtr2
       << "\nAddress of iPtr3 : " << &iPtr3 << endl;

  return 0;
}
```

OUTPUT – From Compiler #1

```
Value of iPtr1 : 00000000
Value of iPtr2 : 00000000
Value of iPtr3 : 0012FF7C

Address of iVar : 0012FF7C
Address of iPtr1 : 0012FF78
Address of iPtr2 : 0012FF74
Address of iPtr3 : 0012FF70
```

OUTPUT – From Compiler #2

```
Value of iPtr1 : 00000000
Value of iPtr2 : 00000000
Value of iPtr3 : 0012FF60

Address of iVar : 0012FF60
Address of iPtr1 : 0012FF54
Address of iPtr2 : 0012FF48
Address of iPtr3 : 0012FF3C
```

The above example shows three different ways to initialize a pointer variable.

Note!

Actual address values will be obtained during program execution. The system that runs the program will assign the required memory block(s) for each individual program; this memory block will have the same size (depend on the actual data being created) but the starting (memory) address of this memory block may be different for each execution.

1.2 Assigning Value to Pointer Variable

Assigning values to pointer variables (or, just pointers) can also be done.

Of course, values being assigned must be addresses. These address values must be valid and properly obtained while being used.

Example 2

```
/**
 * Program Name: cis25L121212.cpp
 * Discussion:   Pointer Accessing
 */
```

```
// Include/Header file(s)
#include <iostream>
using namespace std;

/*Function prototypes*/
void printClassInfo(void);

//Application Driver
int main() {
  int iVar{5};
  int* iPtr1{nullptr};   /* Initializing with predefined value of
                            nullptr*/
  int* iPtr2{iPtr1};  /* Initializing with a known value from another
                         variable*/
  int* iPtr3{&iVar};  /* Initializing with a known value based on type
                         definition*/

  printClassInfo();

  cout << "\nValue of iPtr1 : " << iPtr1
       << "\nValue of iPtr2 : " << iPtr2
       << "\nValue of iPtr3 : " << iPtr3 << endl;

  cout << "\nAddress of iVar : " << &iVar
       << "\nAddress of iPtr1 : " << &iPtr1
       << "\nAddress of iPtr2 : " << &iPtr2
       << "\nAddress of iPtr3 : " << &iPtr3 << endl;

  iPtr1 = iPtr3; /* Assigning value in iPtr3 to iPtr1. The value
                    in iPtr3 is an address, which is the memory
                    location of iVar. That means iPtr1 also has
                    the address of iVar*/
  iPtr2 = iPtr1; /* Assigning value in iPtr1 to iPtr2. The value
                    in iPtr1 is an address, which is the memory
                    location of iVar.*/
  iPtr3 = nullptr; /* Assigning nullptr to iPtr3. Note that this
                      is the only constant value that one should
                      use to assign to a pointer; other constant
                      values should not be used in this way of
                      setting for any pointer.*/
  cout << "\nValue pointed to by iPtr1 : " << *iPtr1
       << "\nValue pointed to by iPtr2 : " << *iPtr2 << endl;

  return 0;
}

// Function Definitions

/**
 * Function Name: printClassInfo()
 * Description:   To print class information
 * Pre:           Nothing (no argument required)
```

```
 * Post:      :   Displaying class information on screen
 */
void printClassInfo() {
  cout << "Class Information --"
       << "\n  CIS 25 -- C++ Programming"
       << "\n  Laney College" << endl;
  return;
}
```

OUTPUT – From Compiler #1

```
Class Information --
  CIS 25 -- C++ Programming
  Laney College

Value of iPtr1 : 00000000
Value of iPtr2 : 00000000
Value of iPtr3 : 0012FF7C

Address of iVar : 0012FF7C
Address of iPtr1 : 0012FF78
Address of iPtr2 : 0012FF74
Address of iPtr3 : 0012FF70

Value pointed to by iPtr1 : 5
Value pointed to by iPtr2 : 5
```

OUTPUT – From Compiler #1

```
Class Information --
  CIS 25 -- C++ Programming
  Laney College

Value of iPtr1 : 00000000
Value of iPtr2 : 00000000
Value of iPtr3 : 0012FF60

Address of iVar : 0012FF60
Address of iPtr1 : 0012FF54
Address of iPtr2 : 0012FF48
Address of iPtr3 : 0012FF3C

Value pointed to by iPtr1 : 5
Value pointed to by iPtr2 : 5
```

In the above example, the constant value of `nullptr` was used in the assignment to `iPtr3`.

- This is the only acceptable **constant** value for the assignment coded this way.
- It then says that the pointer is being nullified, and there is no data value being pointed to by the pointer.
- And again, `nullptr` is a pointer constant, which has value **0** and a given type.

During an execution of any program, there are other pointer constants. These are the addresses of declared variables being used in the program.

1) Then, pointer constants refer to these declared variables may not be valid during different runs as different address values being assigned to the memory blocks.
2) Thus, assigning these kinds of pointer constants are not recommended and should be avoided.

The following example shows several assignment statements that need to be discussed. Pay attention to the addresses of the declared variables.

Example

```
iPtr1 = static_cast<int *> (0x0012ff78); // BAD
iPtr2 = 0x0012ff74; // BAD
```

1.3 Retrieving Value from a Pointer Variable

- Retrieving value through a pointer means that the value is obtained at the given memory location currently stored in the pointer.
- The retrieved value is obtained through the **dereference operator * (**also a value is to be stored indirectly through this pointer – Hence, **indirection operator)**.

Consider the following statements.

```
int iVar1 = 5;
int iVar2;
int* iPtr4;

iPtr4 = &iVar1;
iVar2 = *iPtr4;
*iPtr4 = 100;
iVar2 = iVar1;
```

How should the above statements be understood? What would be the problem for the following statements?

```
int* iPtr4;

iPtr4 = NULL;
cout << *iPtr4;
```

We will discuss the above questions in class!

And, you just have fun with the above! BUT, please never write code as above again!!!???

What and why for the following implementation?

```
void printClassInfo() {
  cout << "Class Information --"
       "\n  CIS 25 -- C++ Programming"
       "\n  Laney College" << endl;

  return;
}

void printClassInfo() {
  cout << "Class Information --"
       "\n  CIS 25 -- C++ Programming"
       "\n  Laney College" << endl;
}
int iVar{5};
int* iPtr1{nullptr};   /* Initializing with predefined value of
                          nullptr*/
```

```
int* iPtr2{iPtr1};  /* Initializing with a known value from another
                       variable*/
int* iPtr3{&iVar};  /* Initializing with a known value based on type
                       definition*/
```