Lecture 12.2

Topics

1. Pointer Variables – Introduction

____

1. Pointer Variables – Introduction

- Prior to this pointer discussion, all variables are used to hold data but not the locations (or addresses of the memory locations) where data are located at (except for array names).
- As we are writing code, we are mostly interested in data values, therefore the functions and operations were designed around the direct access to these values.
- What if we have the knowledge of the locations of the data, which means we have the address of the memory location where data value is stored?
- Of course, we can do a few more things, but how?

Let's look at a new type of variable called **pointer**,

1.1 Definition

A pointer variable is a derived variable that holds the address of a given type. With only one exception, a pointer variable can 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, float, char`, etc.
- **`*`** is the indirection operator.
- **`ptrVariableName`** is any valid variable name.

For examples,

```
int* iPtr;
double* dPtr;
```

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; this must be the address of a memory block that was reserved for a value of type `int`.

Declaring a pointer variable allows this variable to be used. Using a variable means both storing (initializing and assigning) and retrieving values to and from this variable.

1.2 Initializing a Pointer Variable

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

Example 1

```
/**
 *Program Name: cis26L1221.c
 *Discussion:   Pointer Initialization
 */
```

```
#include <stdio.h>

int main( void ) {
  int iVar = 5;

  int* iPtr1 = NULL;   /*Initializing with predefined
                         value of NULL (which has value 0
                         defined in stdio.h)*/
  int* iPtr2 = iPtr1;  /*Initializing with a known value
                         from another
                         variable*/
  int* iPtr3 = &iVar;  /*Initializing with a known value
                         based on type definition*/

  printf( "\nValue of iPtr1 is %p\n", iPtr1 );
  printf( "\nValue of iPtr2 is %p\n", iPtr2 );
  printf( "\nValue of iPtr3 is %p\n", iPtr3 );

  return 0;
}
```

OUTPUT – From Compiler #1

```
Value of iPtr1 is 00000000

Value of iPtr2 is 00000000

Value of iPtr3 is 0012FF60
```

OUTPUT – From Compiler #2

```
Value of iPtr1 is 00000000

Value of iPtr2 is 00000000

Value of iPtr3 is 0012FF7C
```

The above example shows three different ways to initialize a pointer variable. Note that the address values will be obtained while running the program. The system will assign memory block(s) for each individual program and this memory block may 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.

1.3 Assigning a Pointer Variable

Assigning values to pointer variables (or, just pointers) can also be done. The values being assigned must be addresses and these values must be properly obtained and used (that means the values must be addresses of valid memory blocks or declared variables).

Example 2

```
/**
 *Program Name: cis26L1222.c
 *Discussion:   Assigning values to pointers
 */
#include <stdio.h>
```

```
int main( void ) {
  int iVar = 5;

  int* iPtr1 = NULL;   /*Initializing with predefined
                         value of NULL (which has value 0
                         defined in stdio.h)*/
  int* iPtr2 = iPtr1;  /*Initializing with a known value
                         from another variable*/
  int* iPtr3 = &iVar;  /*Initializing with a known value
                         based on type definition*/

  printf( "\nValue of iPtr1 is %p\n", iPtr1 );
  printf( "\nValue of iPtr2 is %p\n", iPtr2 );
  printf( "\nValue of iPtr3 is %p\n", iPtr3 );

  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 = NULL; /*Assigning NULL to iPtr3. The new value
                  stored in iPtr3 is NULL, which is 0.
                  Note that this is the only constant value
                  that one can use to assign to a pointer;
                  other constant values should not be used
                  in this way of initializing of any pointer.*/
  printf( "\nValue of iPtr1 is %p\n", iPtr1 );
  printf( "\nValue of iPtr2 is %p\n", iPtr2 );
  printf( "\nValue of iPtr3 is %p\n", iPtr3 );

  return 0;
}
```

OUTPUT – From Compiler #1

```
Value of iPtr1 is 00000000

Value of iPtr2 is 00000000

Value of iPtr3 is 0012FF60

Value of iPtr1 is 0012FF60

Value of iPtr2 is 0012FF60

Value of iPtr3 is 00000000
```

OUTPUT – From Compiler #2

```
Value of iPtr1 is 00000000

Value of iPtr2 is 00000000
Value of iPtr3 is 0012FF7C

Value of iPtr1 is 0012FF7C

Value of iPtr2 is 0012FF7C

Value of iPtr3 is 00000000
```

In the above example, the constant value **`NULL`** was used in the assignment to `iPtr3`. This is the only acceptable value for the assignment (if it is to be coded this way). It then says (or should be read as) that the pointer is being nullified and there is no data value being pointed to by the pointer.

`NULL` is also called a pointer constant, which has value **`ZERO (0)`**. This is the only pointer constant that can be used at any time to nullify a pointer variable.