Digital Design Verification

Assignment # 04

Pointers

Release: 1.0

Date: 13-August-2024

NUST Chip Design Centre (NCDC), Islamabad, Pakistan

Revision History

Revision Number	Revision Date	Nature of Revision	Designed By
1.0	13/08/2024	Complete manual	Hira Sohail

TASK # 01:

A pointer is a variable that holds a memory address. Pointers are declared using the asterisk (*) operator:

```
int *p;
```

In the above example, the type of the variable p is int* (usually pronounced "int star" or "int pointer"). Like other variables, it is a fixed reference to an allocated place in memory. Unlike other variables, that memory location itself holds the address of another place in memory. That is what makes it a "pointer;" it "points" to another location.

Until it is initialized, however, it doesn't point to anything in particular. To avoid a lot of headaches later on, pointers should always be intialized when they are declared. If you don't know what it should point to immediately, you can always initialize it to a special value (zero) using the NULL constant:

```
int *p = NULL;
```

Question: How would we initialize p so that it points to x?

WARNING: If you wish to declare multiple pointers on the same line, you must include an asterisk with each of them. For instance:

```
int *p = NULL, *q = NULL;
```

In this course, we strongly recommend declaring multiple pointers on a single line.

To assign pointers so that they point to existing variables, you can use the address-of operator (&) just as we used it earlier to print the address:

```
p = &x;
```

After this line, p is now a pointer to the location that stores the value associated with the variable x. Note that x is NOT a pointer; it is just a variable name.

You can now print the value of p (the pointer) and verify that it is the same as the address of x:

```
printf("Pointer value of p is %p\n", p);
```

Dereferencing pointers

To retrieve the value that a pointer points to, use the "dereference" operator (*). Yes, it's an asterisk, the same as we used to declare the pointer.

In the running example, we know that p is an int* (an "int-pointer" or a "pointer to an int"). We have also initialized it to point to the location of variable x. Thus, we can print the integer that p points to using the following code:

```
printf("Dereferenced value of p is %d, and x = %d\n", *p, x);
```

Question: What would happen if we passed p instead of *p to printf?

Question: What would happen if we dereferenced a pointer that had the value of NULL?

Exercise:

What does the following code print? Step through it as if you were the machine. This is sometimes referred to as "tracing" the code. You may find it helpful to draw diagrams with boxes for memory locations and arrows for pointers.

```
int a = 42;
int b = 7;
int c = 999;
int *t = &a;
int *u = NULL;
printf("%d %d\n", a, *t);
c = b;
u = t;
printf("%d %d\n", c, *u);
a = 8;
b = 8;
printf("%d %d %d %d\n", b, c, *t, *u);

*t = 123;
printf("%d %d %d %d %d\n", a, b, c, *t, *u);
```

DO THIS: Copy the previous code into your file and test it. Then, add more lines of code to do the following:

1. Update t to point to c. Use a pointer dereference to change the value of c to 555. Verify that it worked by adding a printout. Does this change any of the other values?

2. Change the value of c again using a direct assignment. Verify that the pointer t still points to the value by printing the result of dereferencing it.

Question: Would happen if you tried to execute the following code? How could you fix it?

```
int *v = &t;
printf("%d\n", *v);
```

This illustrates an important concept: pointers can point to almost anything, even other pointers!

Submission:

Please submit all the answers in hard copy.