Introduction to Pointers

10-4-2002

Opening Discussion

- What did we talk about last class?
- What exactly is a variable in C? What happens when the statement num=5; gets executed?
- Limitations of pass by value:
  - You can only return one thing.
  - Possible speed and memory problems for large structures (which we haven't talked about).
- Pass by reference

Revisiting Code, Pass by Value, and the Stack

- For the 10:30 section a "bug" in the code got things a bit messed up early on so we should look now at the fixed code the look at code displaying the other things that we talked about last class.

Pointers

- Our model of a computer has some form of memory in it. We can refine that a little by saying that the memory is made of a long series of bytes, one after another that we can read from and write to if we know the "address" of them.
- A pointer in programming is such an address. Normal variables refer to what is stored. A pointer is where it is stored.

Warning!

- What we are about to talk about could be confusing because we are going to introduce two/three new uses for symbols that you have just learned other uses for. Don't be afraid, and try not to let it confuse you. The meaning should generally be clear from where it is used in a program.

Declaring Pointers

- A pointer variable is declared by putting and asterisk in front of the variable name.

```
int *n; double *val;
```

- The pointer becomes part of the type and pointers to different types are different types themselves. Since we can make a pointer to any type and a pointer type is a type we can have pointer to pointer types as well.

```
int **ptrptr,***ptrptrptr;
```

The “Address of” Operator

- C has an operator that allows us to get the address of a variable. You have seen it before because you had to use it with scanf. It is the & (which unfortunately also means bitwise and).
- Applying & to a variable/expression gives you the address it is stored at in memory. The type is a pointer to the type of the variable/expression.

```
int a;
int *b=&a;
```

The Dereference Operator

- Most operators have an inverse and the same is true for &. It’s inverse is the dereference operator, * (which unfortunately is just like a multiply).
- Applied to a pointer, it “gives you” the value that was pointed to.

```
int a=5,*b=&a;
printf(“%d\n”,*b);
```

Assignments with Pointers

- Pointers can be involved in two distinct types of assignment, but in all cases the types on both sides of the assignment operator must be the same.
- Pointer assignment:

```
int *a,*b,c;
a=&c;
b=a;
```

- Dereferenced value assignment:

```
*a=4;
```

Pointer Arithmetic

- Pointers can also be involved in simple addition and subtraction opertations.
- When this is done, the value changes by the sizeof the type the pointer points to for each unit incremented or decremented.
- For example, using **a** from the last slide, **a+1** is four larger than **a** on these machines because a points to an int.

Code Involving Pointers

- Now we are going to write some code with pointers. In the code we will also print out the addresses of pointers. It is somewhat standard to print pointers as hex values.
- We can also examine the layout of the stack and stack variables this way. This is more for your knowledge than use in programming.

Minute Essay

- What does this do?

```
int a=4,*b=&a;
printf(“%d\n”,5**b);
```

- I’m going to be gone next week. We’ll have a review session this afternoon for the test on Monday. You will have a sub-prof on Wednesday and then Friday you are off.
- Review session times: 1:30 and 4:30.