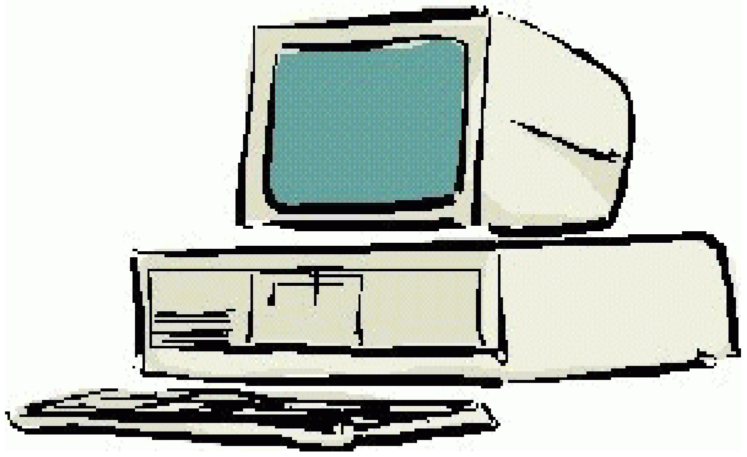
Programming in C



Session 5

Seema Sirpal
Delhi University Computer Centre

Introduction to Pointers

Pointers are variables that hold addresses in C and C++.

They provide much power and utility for the programmer to access and manipulate data in ways not seen in some other languages.

They are also useful for passing parameters into functions in a manner that allows a function to modify and return values to the calling routine.

When used incorrectly, they also are a frequent source of both program bugs and programmer frustration.

Pointers

As a program is executing all variables are stored in memory, each at its own unique address or location.

Typically, a variable and its associated memory address contain data values. For instance, when you declare:

```
int count = 5;
```

The value "5" is stored in memory and can be accessed by using the variable "count".

A pointer is a special type of variable that contains a memory address rather than a data value.

Just as data is modified when a normal variable is used, the value of the address stored in a pointer is modified as a pointer variable is manipulated.

Pointers

Usually, the address stored in the pointer is the address of some other variable.

```
int *ptr;
```

ptr = &count ***/* Stores the address of count in ptr */***
    /* The unary operator & returns the address of a variable */

Pointers

To get the value that is stored at the memory location in the pointer it is necessary to dereference the pointer. Dereferencing is done with the unary operator "*".

```
int total;

total = *ptr;
   /* The value in the address stored in ptr is assigned to total */
```

Declaration & Initialization

```
int main()
{
    int j;
    int k;
    int l;
    int *pt1;    /* Declares an integer pointer */
    int *pt2;    /* Declares an integer pointer */
    float values[100];
    float results[100];
    float *pt3;    /* Declares a float pointer */
    float *pt4;    /* Declares a float pointer */
    j = 1;
    k = 2;
    pt1 = &j;    /* pt1 contains the address of the variable j */
    pt2 = &k;    /* pt2 contains the address of variable k */
    pt3 = values;
        /* pt3 contains the address of the first element of values */
    pt3 = &values[0];
        /* This is the equivalent of the above statement */
    return 0;
}
```

Pointer Dereferencing/ Value Assignment

Dereferencing allows manipulation of the data contained at the memory address stored in the pointer.

The pointer stores a memory address.

Dereferencing allows the data at that memory address to be modified.

The unary operator "*" is used to dereference.

For instance:

```
*pt1 =*pt1 + 2;
```

This adds two to the value "pointer to" by pt1. That is, this statement adds 2 to the contents of the memory address contained in the pointer pt1. So, from the main program, pt1 contains the address of j. The variable "j" was initialized to 1. The effect of the above statement is to add 2 to j.

Pointer Dereferencing/ Value Assignment

The contents of the address contained in a pointer may be assigned to another pointer or to a variable.

```
*pt2 = *pt1;
   /* assigns the contents of the memory pointed to by pt1 */
   /* to the contents of the memory pointer to by pt2; */

k = *pt2;
   /* assigns the contents of the address pointer to by pt2 to k. */
```

Pointer Arithmetic

Part of the power of pointers comes from the ability to perform arithmetic on the pointers themselves. Pointers can be incremented, decremented and manipulated using arithmetic expressions. Recall the float pointer "pt3" and the float array "values" declared above in the main program.

```
pt3 = &values[0];            /* The address of the first element of "values" is stored in pt3*/
pt3++;                         /* pt3 now contains the address of the second element of values */
*pt3 = 3.1415927;             /* The second element of values now has pie (actually pi)*/
pt3 += 25;                    /* pt3 now points to the 27th element of values */
*pt3 = 2.22222;               /* The 27th element of values is now 2.22222 */
pt3 = values;                 /*pt3 points to the start of values, now */
for (ii = 0; ii < 100; ii++)
{
    *pt3++ = 37.0;                    /* This sets the entire array to 37.0 */
}
pt3 = &values[0];                     /* pt3 contains the address of the first element of values */
pt4 = &results[0];                    /* pt4 contains the address of the first element of results */
for (ii=0; ii < 100; ii++)
{
   *pt4 = *pt3;                 /* The contents of the address contained in pt3 are assigned to
                                       the contents of the address contained in pt4 */

    pt4++;
    pt3++;
}
```