Lecture 3.1

Pointers (cont.)

Pointer Dereferencing

- Dereferencing allows manipulation of the **data** contained at the memory address stored in the pointer**.**
- Remember!
  - The pointer stores a memory **address**.
  - Dereferencing allows the **data** at that memory address to be modified.
  - The unary operator "*****" is used to dereference.

Pointer Dereferencing (cont)

- Consider the following piece of code:

```
int j = 1;
pt1 = &j; //pt1 points to j
*pt1 =*pt1 + 2; //adds two to the value
                //pointed to by pt1
```

- The effect of the above statement is to add 2 to j.

Pointer Dereferencing (cont 2)

- 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 pointed to
             //by pt2;
k = *pt2; //Assigns the contents of the
          //address pointed to by pt2 to k.
```

- The value of k will now be 3.

Pointer Arithmetic

- Pointers can be incremented, decremented and manipulated using arithmetic expressions.

```
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)
```


Pointer Arithmetic (cont)

```
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 (i = 0; i < 100; i++)
{
   *pt3++ = 37.0; // This sets the entire array to 37.0
}
```


Pointer Arithmetic (cont 2)

```
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 (i=0; i < 100; i++;)
{
   *pt4 = *pt3; // The contents of the address contained
                // in pt3 are assigned to the contents of the
                // address contained in pt4
    pt4++;
    pt3++;
}
```


Relationship between Pointers and Arrays

- Suppose the following declarations are made.

```
float temperatures[31];
float *tmp;
```

- The address of the first element of the array temperatures can be assigned to temp in two ways.

```
tmp = &temperatures[0];
tmp = temperatures;
```

Relationship between Pointers and Arrays (cont)

- Values for the first element can be assigned in two ways:

  temperatures[0] = 29.3;

  ***tmp = 15.2;**

- Other elements can be updated via the pointer, as well.

  tmp = &temperatures[0];

  ***(tmp + 1) = 19.0**; /*assigns 19 to the second element of temp */

Relationship between Pointers and Arrays (cont2)

tmp = tmp + 10; /* tmp now has the address of the 11th element of the array */

***tmp = 25.0;** /* temperatures[9] = 25, remember that arrays are zero based, so the tenth element is at index 9 */

tmp++; /* tmp now points at the 12th element */

***tmp = 40.9;** /* temperatures[11] = 40.9 */

Pointers and Character Arrays

- Assigning a character literal to an array

  char str1[] = "Hello World";

- A character pointer can also be assigned the address of a string constant or of a character array.

  char *lpointer = "Hello World"; /* Assigns the address of the literal to lpointer */

  char *apointer = str1; /* Assigns the starting address of str1 to apointer */

  char *apointer = &str1[0]; /* Assigns the starting address of str1 to apointer */

Copying one array to another

- There is no direct means in the C or C++ language to copy one array to another.
- Must be done either with a standard library function or element wise in a loop.
- Example

```
#include <stdio.h>
int main()
{
   char str1[] = "Hello World";
   char str2[] = "Goodbye World";
   str2 = str1;
   return 0;
}
```