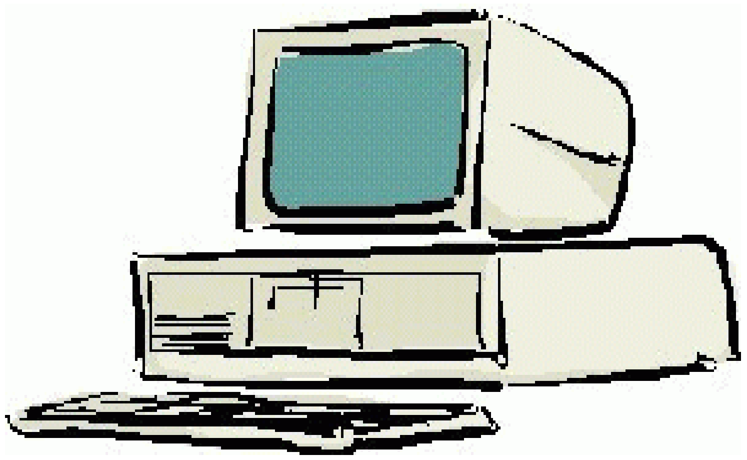
Programming in C



Session 7

Seema Sirpal
Delhi University Computer Centre

Relationship between Pointers & Arrays

In some cases, a pointer can be used as a convenient way to access or manipulate the data in an array.

Suppose the following declarations are made.

```
float temperatures[31];
   /* An array of 31 float values, the daily temperatures in a month */


float *temp;    /* A pointer to type float */
```

Since temp is a float pointer, it can hold the address of a float variable.

Relationship between Pointers & Arrays

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

```
temp = &temperatures[0];
temp = temperatures;
    /* This is an alternate notation for the first
    element of the array. Same as temperatures = &temperatures[0]. */
```

The temperature of the first day can be assigned in two ways.

```
temperatures[0] = 29.3;
*temp = 15.2;
```

Relationship between Pointers & Arrays

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

```
temp = &temperatures[0];

*(temp + 1) = 19.0;
   /* Assigns 19.0 to the second element of temperatures */

temp = temp + 9;
   /* temp now has the address of the 10th element of the array */

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

temp++;    /* temp now points at the 11th element */

*temp = 40.9;    /* temperatures[10] = 40.9 */
```

Relationship between Pointers & Arrays

Pointers are particularly useful for manipulating strings, which are stored as null terminated character arrays in C

Character Arrays

Strings are stored in C as character arrays terminated by the null character, '\0'.

The array length must be at least one greater than the length of the string to allow storage of the terminator.

String constants or literals are stored internally as a null character terminated character array.

Assigning a character literal to an array is done as follows.

```
char str1[] = "Hello World";
char str2[] = "Goodbye World";
```

Character Arrays

The compiler automatically sizes the arrays correctly. For this example, str1 is of length 12, str2 is of length 14. These lengths include space for the null character that is added at the end of the string.

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 */
```

Character Arrays

There is no direct means in the C language to copy one array to another, or one string to another. It must be done either with a standard library function or element wise in a loop. Let's try to copy one string to another.

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

   str2 = str1;

   return 0;
}
```

Can you see what's wrong with this code. As stated, there is no operation to assign one array to another in C. This code produced this compiler error.

```
error: '=' : cannot convert from 'char [12]' to 'char [14]'
There is no context in which this conversion is possible.
```

Character Arrays

Now let's make another attempt using character pointers.

```
#include <stdio.h>
int main()
{
    char str1[] = "Hello World";
    char str2[] = "Goodbye World";
    char *cpt1;
    char *cpt2;

    cpt1 = &str1[0];
    cpt2 = &str2[0];

    printf("str1 is %s\n",str1);
    printf("str2 is %s\n",str2);
    printf("cpt1 is %s\n",cpt1);
    printf("cpt2 is %s\n",cpt2);

    cpt2 = cpt1;

    printf("str1 is %s\n",str1);
    printf("str2 is %s\n",str2);
    printf("cpt1 is %s\n",cpt1);
    printf("cpt2 is %s\n",cpt2);
    return 0;
}
```

Character Arrays

Results:

```
str1 is Hello World
str2 is Goodbye World
cpt1 is Hello World
cpt2 is Goodbye World
str1 is Hello World
str2 is Goodbye World
cpt1 is Hello World
cpt2 is Hello World
```

As can be seen from the results, all that happened is that the pointer cpt2 was assigned the value of cpt1, that is, the address of str1. The contents of the array str2 were not changed. The only way to copy a string or any array in C is element by element.

Character Arrays

Here is a program that correctly copies one string to another.

```
#include <stdio.h>
int main()
{
    int i;
    char str1[] = "Hello World";
    char str2[] = "Goodbye World";
    printf("str1 is %s\n",str1);
    printf("str2 is %s\n",str2);
    i = 0;
    while ((str2[i] = str1[i]) != '\0') {
        i++;
    }
    printf("str1 is %s\n",str1);
    printf("str2 is %s\n",str2);
    return 0;
}
```

Character Arrays

Results:

```
str1 is Hello World
str2 is Goodbye World
str1 is Hello World
str2 is Hello World
```

Practice Problem

Try re-implementing the above program using pointers in the copy loop.

Hints:

```
cpt1 = &str1[0];
cpt2 = &str2[0];
```

Use these pointers in the while loop, remember to dereference.

Solution

```
#include <stdio.h>
int main()
{
    char str1[] = "Hello World";
    char str2[] = "Goodbye World";
    char *cpt1;
    char *cpt2;
    cpt1 = &str1[0];
    cpt2 = &str2[0];
    printf("str1 is %s\n",str1);
    printf("str2 is %s\n",str2);
    while ((*cpt2 = *cpt1) != '\0') {
        cpt2++;
        cpt1++;
    }
    printf("str1 is %s\n",str1);
    printf("str2 is %s\n",str2);
    return 0;
}
```

Strings

Stings in C are stored as null character, '\0', terminated character arrays.

This means that the length of a string is the number of characters it contains plus one to store the null character.

Common string operations include finding lengths, copying, searching, replacing and counting the occurrences of specific characters and words.

Strings

Here is a simple way to determine the length of a string.

```
#include <stdio.h>
int main()
{
    char sentence[] = "Hello World";
    int count = 0;
    int i;

    for (i = 0; sentence[i] != '\0'; i++)
    {
        count++;
    }
    printf("The string %s has %d characters ",
        sentence,count);
    printf("and is stored in %d bytes\n",
        count+1);
    return 0;
}
```

String

Each character within the array sentence is compared with the null character terminator until the end of the string is encountered.

This technique can be generalized to search for any character.

Sample Program

Here is a program that counts the occurrences of each lowercase letter of the alphabet in a string.

```
#include <stdio.h>
int main()
{
   int i,j;
   char buffer[120];    /* Holds input strings */
   char *pt;    /* A pointer to data type char */
   char alphabet[27] = "abcdefghijklmnopqrstuvwxyz";
   int alphaCount[26];    /* an array of counts */
   /* Initialize counts */
   for (i = 0; i < 26; i++)
   {
      alphaCount[i] = 0;
   }
   while ((pt = gets(buffer)) != NULL)
   {
      for (i = 0; i < 120 && buffer[i] != '\0'; i++)
      {
         for (j = 0; j < 26; j++)
         {
            if (buffer[i] == alphabet[j])
            {
               alphaCount[j]++;
            }
         }
      }
   }
   for (i = 0; i < 26; i++)
   {
      printf("Found %d occurrences of %c\n",
         alphaCount[i],alphabet[i]);
   }
   return 0;
}
```

Sample

gets reads a line of input into a character array. Its prototype is:
char *gets(char *buffer);

It returns a pointer to the character string if successful, or NULL if end of file is reached or if an error has occurred. The string is also read into the character array specified as an argument. The character string will be terminated be a "\0", which is standard for C.

The first step in this program is to initialize the count array. When you use an automatic variable, you must initialize it. Otherwise, it contains whatever data happened to be at the memory location it is stored in.

The while loop in this program reads lines of input until an end of file is encountered. If you run this program, remember that an end of file, EOF, is entered as a control-d or a control-z from you keyboard. This is, after you type the last line of text, type control-z or control-d to enter the End of File character.

For each line read in, each character in the line is compared against each letter in the alphabet. If a match is found the appropriate count is incremented. The last step in this program writes out the results.

Practice

1) Type in the above program. Compile and run.

2) Modify the above program to count the occurrences of the digits 0-9, rather than letters

Solution

```
#include <stdio.h>
int main()
{
    int i,j;
    char buffer[120];    /* Holds input strings */
    char *pt;    /* A pointer to data type char */
    char digits[11] = "0123456789";
    int digitCount[10];    /* an array of counts */
    /*Initialize counts */
    for (i = 0; i < 10; i++)
    {
        digitCount[i] = 0;
    }
    while ((pt = gets(buffer)) != NULL)
    {
        for (i = 0; i < 120 && buffer[i] != '\0'; i++)
        {
            for (j = 0; j < 10; j++)
            {
                if (buffer[i] == digits[j])
                {
                    digitCount[j]++;
                }
            }
        }
    }
    for (i = 0; i < 10; i++)
    {
        printf("Found %d occurences of %c\n",
            digitCount[i],digits[i]);
    }
    return 0;
}
```

Library Functions for String Operation

To use any of these functions in your code, the header file "strings.h" must be included.

strcpy
strcpy copies a string, including the null character terminator from the source string to the destination. This function returns a pointer to the destination string, or a NULL pointer on error. Its prototype is:

```
char *strcpy(char *dst, const char *src);
```

strncpy
strncpy is similar to strcpy, but it allows the number of characters to be copied to be specified. If the source is shorter than the destination, than the destination is padded with null characters up to the length specified. This function returns a pointer to the destination string, or a NULL pointer on error. Its prototype is:

```
char *strncpy(char *dst, const char *src, size_t len);
```

Library Functions for String Operation

To use any of these functions in your code, the header file "strings.h" must be included.

strcat
This function appends a source string to the end of a destination string. This function returns a pointer to the destination string, or a NULL pointer on error. Its prototype is:

```
char *strcat(char *dst, const char *src);
```

strncat
This function appends at most N characters from the source string to the end of the destination string. This function returns a pointer to the destination string, or a NULL pointer on error. Its prototype is:

```
char *strncat(char *dst, const char *src, size_t N);
```

Library Functions for String Operation

To use any of these functions in your code, the header file "strings.h" must be included.

strcmp

This function compares two strings. If the first string is greater than the second, it returns a number greater than zero. If the second string is greater, it returns a number less than zero. If the strings are equal, it returns 0. Its prototype is:

```
int strcmp(const char *first, const char *second);
```

strncmp

This function compares the first N characters of each string. If the first string is greater than the second, it returns a number greater than zero. If the second string is greater, it returns a number less than zero. If the strings are equal, it returns 0. Its prototype is:

```
int strncmp(const char *first, const char *second, size_t N);
```

Library Functions for String Operation

To use any of these functions in your code, the header file "strings.h" must be included.

strlen
This function returns the length of a string, not counting the null character at the end. That is, it returns the character count of the string, without the terminator. Its prototype is:

```
size_t strlen(const char *str);
```

Sample Program

```
#include <stdio.h>
#include <string.h>

int main()
{
   char str1[] = "Hello";
   char str2[] = "World";
   char str3[20];
   char *cpt;
   int length;
   int rc;

   /* Find length of str1 */
   length = strlen(str1);
   printf("String 1, %s, is of length %d\n",str1,length);
```

Sample Program

```
/* Copy str1 to str3, print both */
  printf("\nCopying str1 to str3.\n");
  if ((cpt = strcpy(str3,str1)) == NULL)
  {
      printf("Error on strcpy");
      return 1;
  }
  printf("String 1 is %s\n",str1);
  printf("String 3 is %s\n",str3);

  /* Clear str3 */
  memset(str3,'\0',20);
  printf("\nstr3 is cleared\n");
```

Sample Program

```
/* Copy first 2 characters of str1 to str3 */

 printf("Copying the first two characters of str1 to str3\n");
 strncpy(str3,str1,2);
 printf("String 1 is %s\n",str1);
 printf("String 3 is %s\n",str3);
```

Sample Program

```
/* Compare the first 2 characters of str1, str3 */

 printf("\nComparing the first two characters of str1 and str3\n");
 if ((rc = strncmp(str1,str3,2)) == 0)
 {
    printf("First two characters of str1 and str3 match\n");
 }
 else
 {
    printf("First two characters of str1 and str3 don't match\n");
 }
```

Sample Program

```
/* Compare all characters of str1 and str3 */
 printf("\nComparing all characters of str1 and str3\n");
 rc = strcmp(str1,str3);
 if (rc == 0)
 {
    printf("str1 equals str3\n");
 }
 else if (rc > 0)
 {
    printf("str1 is greater\n");
 }
 else /* rc < 0 */
 {
    printf("str3 is greater\n");
 }
```

Sample Program

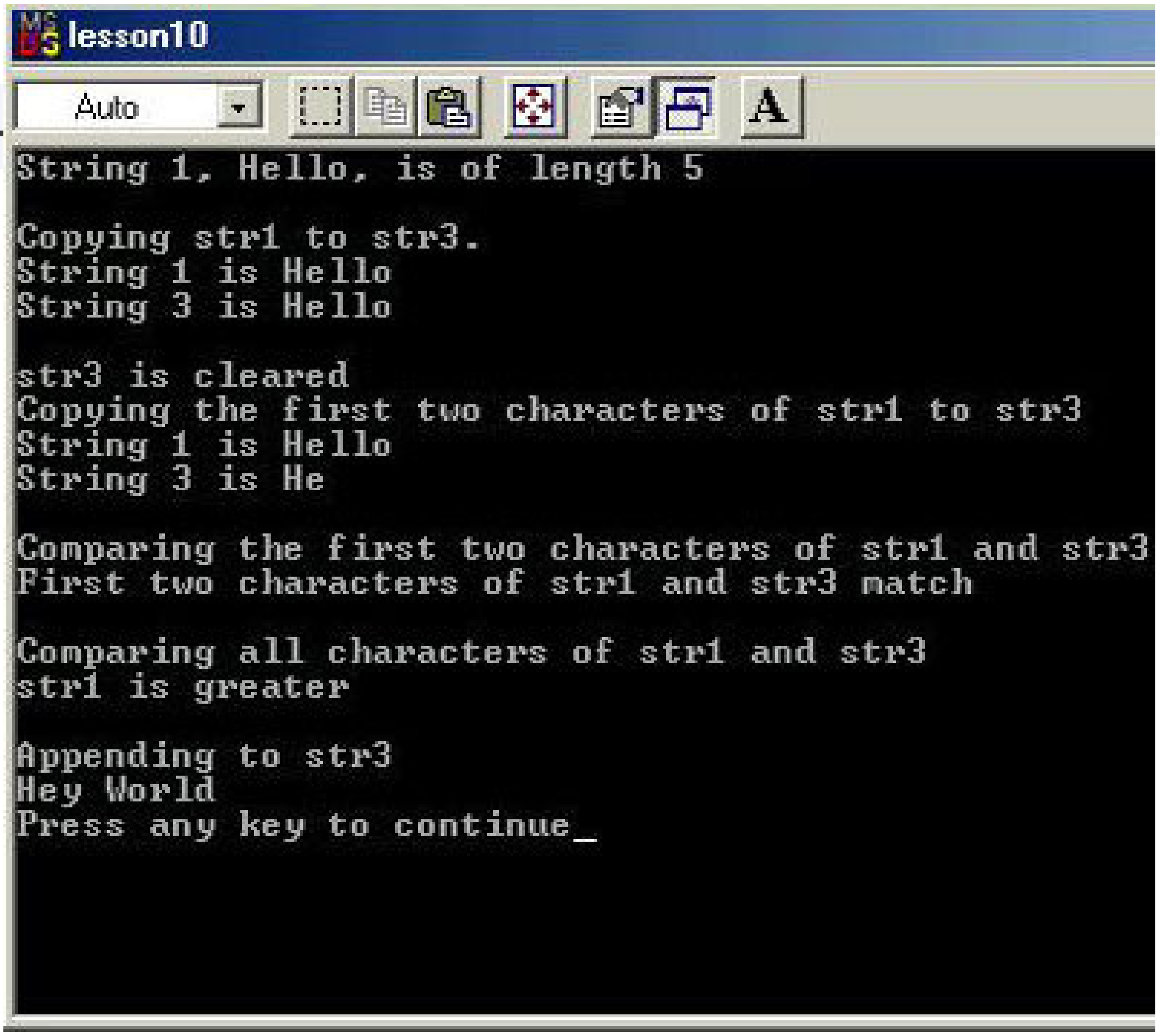
```
/* Append to "he" in str3 */

 printf("\nAppending to str3\n");
 str3[2] = 'y';
 str3[3] = ' ';
 strcat(str3,str2);
 printf("%s\n",str3);

 return 0;
}
```

Sample Program

Here are the results.



```
String 1, Hello, is of length 5

Copying str1 to str3.
String 1 is Hello
String 3 is Hello

str3 is cleared
Copying the first two characters of str1 to str3
String 1 is Hello
String 3 is He

Comparing the first two characters of str1 and str3
First two characters of str1 and str3 match

Comparing all characters of str1 and str3
str1 is greater

Appending to str3
Hey World
Press any key to continue_
```