C Programming

String processing

in C language

Strings

C has no special `string' type. A C string is just an array of `char`, with the understanding that last character is zero (not the character literal `'0'`, but the character with numerical value zero). C provides a familiar notation for string literals: simply enclose the string in double-quotes, and C will add the zero-terminator by itself:

```
char c[] = "abcde";

/*  c contains 6 chars, 'a', 'b', 'c', 'd', 'e', 0  */
```

Normally you do not need to worry about the zero-terminator unless you have to know exactly how much space the string will take (see below for an example).

You may use any character literal in a string literal:

```
char c[] = "Done.";
```

A string in C is simply an array of characters. The following line declares an array that can hold a string of up to 99 characters.

char str[100];

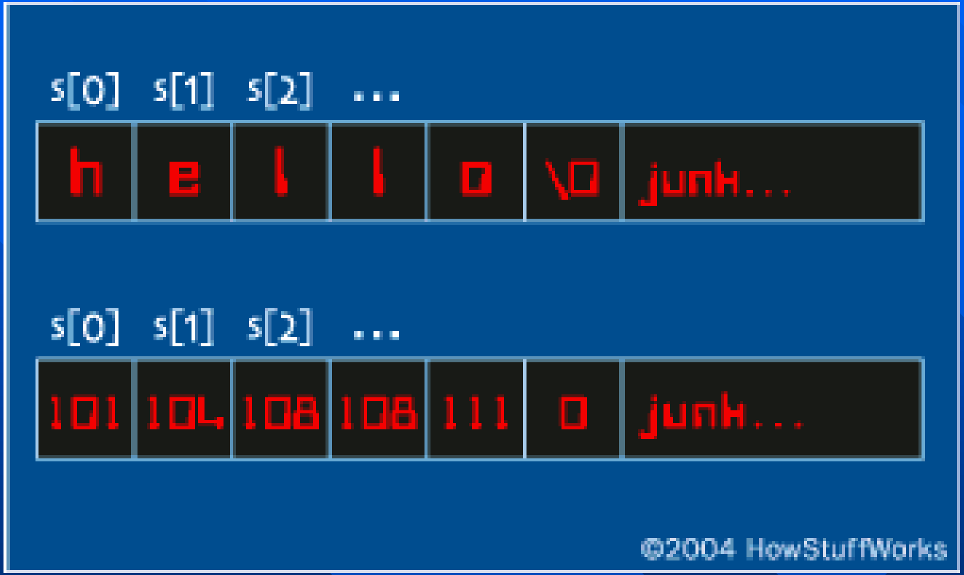
It holds characters as you would expect: **str[0]** is the first character of the string, **str[1]** is the second character, and so on. But why is a 100-element array unable to hold up to 100 characters? Because C uses *null-terminated strings*, which means that the end of any string is marked by the ASCII value 0 (the null character), which is also represented in C as **'\0'**.

Null termination is very different from the way many other languages handle strings. For example, in Pascal, each string consists of an array of characters, with a length byte that keeps count of the number of characters stored in the array. This structure gives Pascal a definite advantage when you ask for the length of a string. Pascal can simply return the length byte, whereas C has to count the characters until it finds **'\0'**. This fact makes C much slower than Pascal in certain cases, but in others it makes it faster, as we will see in the examples below.

The fact that strings are not native to C forces you to create some fairly roundabout code. For example, suppose you want to assign one string to another string; that is, you want to copy the contents of one string to another. In C, as we saw in the last article, you cannot simply assign one array to another. You have to copy it element by element. The string library (<string.h> or <strings.h>) contains a function called *strcpy* for this task. Here is an extremely common piece of code to find in a normal C program:

```
char s[100];
strcpy(s, "hello");
```

After these two lines execute, the following diagram shows the contents of s:



The top diagram shows the array with its characters. The bottom diagram shows the equivalent ASCII code values for the characters, and is how C actually thinks about the string (as an array of bytes containing integer values).

Standard Functions for String Processing

Standard C String Functions

Function	Meaning
atof	converts a string to a double
atoi	converts a string to an integer
atol	converts a string to a long
strcat	concatenates two strings
strchr	finds the first occurance of a character in a string
strcmp	compares two strings
strcoll	compares two strings in accordance to the current locale
strcpy	copies one string to another
strcspn	searches one string for any characters in another
strerror	returns a text version of a given error code

strlen	returns the length of a given string
strncat	concatenates a certain amount of characters of two strings
strncmp	compares a certain amount of characters of two strings
strncpy	copies a certain amount of characters from one string to another
strpbrk	finds the first location of any character in one string, in another string
strrchr	finds the last occurance of a character in a string
strspn	returns the length of a substring of characters of a string
strstr	finds the first occurance of a substring of characters
strtod	converts a string to a double
strtol	converts a string to a long
strtoul	converts a string to an unsigned long

The following code shows how to use *strcpy* in C:

strcpy is used whenever a string is initialized in C

```
#include <string.h>
    int main()
     {
          char s1[100],s2[100];
          strcpy(s1,"hello");
        /* copy "hello" into s1 */
          strcpy(s2,s1);
        /* copy s1 into s2 */
          return 0;
     }
```

You use the **strcmp** function in the string library to compare two strings. It returns an integer that indicates the result of the comparison. Zero means the two strings are equal, a negative value means that **s1** is less than **s2**, and a positive value means **s1** is greater than **s2**.

Example

```
#include <stdio.h>
#include <string.h>
  int main()
  {
       char s1[100],s2[100];
       gets(s1);
       gets(s2);
         if (strcmp(s1,s2)==0)
               printf("equal\n");
          else if (strcmp(s1,s2)<0)
               printf("s1 less than s2\n");
          else
            printf("s1 greater than s2\n");
       return 0;
    }
```

Other common functions in the string library include **strlen**, which returns the length of a string, and **strcat** which concatenates two strings.

Example of using *strlen*

```
  int strlen(char s[])
{
   int x;
    x=0;
    while (s[x] != '\0')
    x++;
    return(x);
}
```

Most C programmers shun this approach because it seems inefficient. Instead, they often use a pointer-based approach:

```
    int strlen(char *s)
{
        int x=0;
        while (*s != '\0')
            {
                x++;
                s++;
            }
        return(x);
}
```

You can abbreviate this code to the following:

```
int strlen(char *s)
  {
        int x=0;
        while (*s++)
        x++;
        return(x);
  }
```

Example of using *strcpy*

1 version

```
strcpy(char s1[],char s2[])
  {
        int x;
        for (x=0; x<=strlen(s2);
x++)
        s1[x]=s2[x];
  }
```

Note here that **<=** is important in the *for* loop because the code then copies the **'\0'**. Be sure to copy *'\0'*. Major bugs occur later on if you leave it out, because the string has no end and therefore an unknown length. Note also that this code is very inefficient, because strlen gets called every time through the for loop.

To solve this problem, you could use the following code:

2 version

```
strcpy(char s1[],char s2[])
  {
        int x, len;
        len=strlen(s2);
        for (x=0; x<=len; x++)
        s1[x]=s2[x];
   }
```

The pointer version is similar

3 version

```
strcpy(char *s1,char *s2)
 {
     while (*s2 != '\0')
         {
              *s1 = *s2;
               s1++;
               s2++;
           }
 }
```

You can compress this code further:

4 version

```
strcpy(char *s1,char *s2)
 {
    while (*s2)
        *s1++ = *s2++;
  }
```

If you wish, you can even say **while (*s1++ = *s2++);**. The first version of ***strcpy*** takes 415 seconds to copy a 120-character string 10,000 times, the second version takes 14.5 seconds, the third version 9.8 seconds, and the fourth 10.3 seconds. As you can see, pointers provide a significant performance boost here.

The prototype for the *strcpy* function in the string library indicates that it is designed to return a pointer to a string:

```
char *strcpy(char *s1,char *s2)
```

Most of the string functions return a string pointer as a result, and strcpy returns the value of s1 as its result.

Using pointers with strings can sometimes result in definite improvements in speed and you can take advantage of these if you think about them a little. For example, suppose you want to remove the leading blanks from a string. You might be inclined to shift characters over on top of the blanks to remove them. In C, you can avoid the movement altogether:

```
#include <stdio.h>
   #include <string.h>
      int main()
        {
              char s[100],*p;
              gets(s);
              p=s;
              while (*p==' ')
              p++;
              printf("%s\n",p);
               return 0;
         }
```

This is much faster than the movement technique, especially for long strings.

More Examples With Strings

Strings

1. A string is nothing more than a character array.
2. All strings end with the NULL character.
3. Use the %s placeholder in the printf() function to display string values.

```
#include  <stdio.h>
main  ( )
{
      char  *s1  =  "abcd";
      char  s2[]  =  "efgh";
      printf(  "%s  %16lu  \n",  s1,  s1);
      printf(  "%s  %16lu  \n",  s2,  s2);
      s1  =  s2;
      printf(  "%s  %16lu  \n",  s1,  s1);
      printf(  "%s  %16lu  \n",  s2,  s2);
}
```

output:

```
abcd 4206592
efgh 2293584
efgh 2293584
efgh 2293584
```

A string is an array of characters

```
#include  <stdio.h>

int  main()
{
        char  myname[]  =  "Dan";
        printf("%s  \n",myname);
}
```

output:

```
Dan
```

Strings always end with the NULL character

ASCII code for the NULL character is \0

```
#include  <stdio.h>

int  main(){

    char  myname[4];
        myname[0]  =  'D';
        myname[1]  =  'a';
        myname[2]  =  'n';
        myname[3]  =  '\0';

        printf("%s  \n",myname);
}
```

output:

```
Dan
```

String constants consist of text enclosed in double quotes

We must use the standard library function strcpy to copy the string constant into the variable.
To initialize the variable name to Sam.

```
#include <string.h>

    int main()
    {
        char    name[4];
        strcpy(name, "Sam");

        return (0);
    }
```

Displaying a string with a string terminator in the middle

```
#include  <stdio.h>

int  main(void)
{
  printf("The  character  \0  is  used  toterminate  a  string.");
  return  0;
}
```

output:

```
The character
```

By setting first[4] to NULL ('\0'), we can shorten the string

```
#include  <string.h>
        #include  <stdio.h>
        int  main()
        {
                char  name[30];
               strcpy(name,  "Saaaaaaaaaaam");
                printf("The  name  is  %s\n",  name);
                name[4]  =  '\0';
                printf("The  name  is  %s\n",  name);
                return  (0);
        }
```

output:

```
The name is Saaaaaaaaaaam
The name is Saaa
```

Read string from keyboard

```
#include <stdio.h>
int main()
{
    char me[20];
    printf("What is your name?");
    scanf("%s",&me);
    printf("Darn glad to meet you, %s!\n",me);
    return(0);
}
```

output:

```
What is your name?John
Darn glad to meet you, John!
```

Reading Strings

The standard function fgets can be used to read a string from the keyboard.

The general form of an fgets call is:

```
fgets(name, sizeof(name), stdin);
```

The arguments are:

name	is the name of a character array
sizeof(name)	indicates the maximum number of characters to read
stdin	is the file to read.

Read a line from the keyboard and reports its length

```
 #include  <string.h>
#include  <stdio.h>
  int  main()
{
        char  line[100];
                        /*  Line  we  are  looking  at  */
        printf("Enter  a  line:  ");
         fgets(line,  sizeof(line),  stdin);
printf("The  length  of  the  line  is:  %d\n", strlen(line));
         return  (0);
}
```

output:

```
Enter a line: string
The length of the line is: 7
```

Ask the user for his first and last name

```
 #include  <stdio.h>
#include  <string.h>
int  main()  {
            char  first[100];
            char  last[100];
            char  full[200];
            printf("Enter  first  name:  ");
            fgets(first,  sizeof(first),  stdin);
            printf("Enter  last  name:  ");
            fgets(last,  sizeof(last),  stdin);
            strcpy(full,  first);
            strcat(full,  "  ");
            strcat(full,  last);
            printf("The  name  is  %s\n",  full);
            return  (0);
          }
```

output:

```
Enter first name: string
Enter last name: last
The name is string last
```

Joining strings without using strcat

```
#include  <stdio.h>
int  main(void)
{
    char  str1[40]  =  "AAA";
    char  str2[]  =  "BBBB";
    int  str1Length  =  0;
    int  str2Length  =  0;
    while  (str1[str1Length])
        str1Length++;
    while  (str2[str2Length])
        str2Length++;
```

```
   if(sizeof str1 < str1Length + str2Length + 1)
      printf("\n str1 is too short.");
else
   {
      str2Length = 0;
      while(str2[str2Length]){
         str1[str1Length++] = str2[str2Length++];
      }
      str1[str1Length] = '\0';
      printf("\n%s\n", str1 );
   }
   return 0;
}
```

output:

```
AAABBBB
```

Take a first name and a last name and combines the two strings

```c
#include  <string.h>
#include  <stdio.h>
int  main()
{
        char  first[100];
        char  last[100];
        char  full_name[200];
        strcpy(first,  "first");
        strcpy(last,  "last");
        strcpy(full_name,  first);
        strcat(full_name,  "  ");
        strcat(full_name,  last);
        printf("The  full  name  is:  %s\n",  full_name);
        return  (0);
}
```

output:

```
The full name is: first last
```