Some Important String Functions

The *strlen()* function

- get string length

The *strcat()* function

- concatenate two strings

The *strcpy()* function

- copy a string from another string.

The *strcmp()* function

- compare two strings

The *strncmp()* function

- compare parts of two strings

The *strncpy()* function

- copy part of a string

The *strchr()* function

- Find a character in a string

```
// crt_strchr.c
//
// This program illustrates searching for a character
// with strchr (search forward) or strrchr (search backward).
//

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

int  ch = 'r';
char string[] = "The quick brown dog jumps over the lazy fox";

int main( void )
{
   char *pdest;
   int result;

   printf( "String to be searched:\n      %s\n", string );
   printf( "Search char:   %c\n", ch );

   // Search forward.
   pdest = strchr( string, ch );
   result = (int)(pdest - string + 1);
   if ( pdest != NULL )
```

```
      printf( "Result:   first %c found at position %d\n",
              ch, result );
   else
      printf( "Result:   %c not found\n" );

   // Search backward.
   pdest = strrchr( string, ch );
   result = (int)(pdest - string + 1);
   if ( pdest != NULL )
      printf( "Result:   last %c found at position %d\n", ch, result );
   else
      printf( "Result:\t%c not found\n", ch );
}
```

The *strstr()* function

Returns a pointer to the first occurrence of a search string in a string.

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

char str[] =    "lazy";
char string[] = "The quick brown dog jumps over the lazy fox";

int main( void )
{
   char *pdest;
   int  result;
   pdest = strstr( string, str );
   result = (int)(pdest - string + 1);
   if ( pdest != NULL )
      printf( "%s found at position %d\n", str, result );
   else
      printf( "%s not found\n", str );
}
```

The *strtok()* funciton

The strtok function in the string.h library can be very useful, but there are a few caveats that need to be borne in mind. Some avoid it, some find it indespensible.

Introduction

Breaking a string down into it's component parts is one of those useful data processing tasks that is almost always required when reading and writing information in a program. It can also be tough to implement, so most string libraries provide a tokenizing function, and C is no different.

An Example - Processing CSV

Before we look at why this could be, we should first see what strtok actually does. In essence, it exists to tokenize a string - turn it into a set of sub-strings, based on the processing of 'separators'.

A typical separator is the comma. Indeed, if a file is exported from a spreadsheet application as CSV, each field is delimited by a comma, and each line delimited by a carriage return.

To turn a line of text into a set of fields, two operations are therefore required:

1. Swap the final carriage return for a null character
2. Translate the comma delimited fields into a set of sub-strings

The first is required because we need to lose the carriage return because it does not form part of the data of the last field, and we need to pass strtok a null-terminated string. The code to do this might look like:

```
int nLength = strlen(szString);
if (szString[nLength]-1 == '\n') {
szString[nLength]-1 = '\0';
}
```

Of course, one might be tempted to also test for '\r' on the basis that each line could be terminated with a LF/CR combination, but that is system dependent and slightly out of scope. The tokenizing process itself, might look like this:

```
char * tok = strtok(szString, ",");
while (tok != NULL) {
// Do something with the tok
tok = strtok(NULL,",");
}
```

The above will move through szString, dividing it into tokens, each delimited by a comma. We use NULL as the first argument to strtok in the loop because otherwise, the function will replace the string with szString each time, since it keeps a global copy in memory for the duration.

A side effect of this is that the string kept in memory might become corrupt, or remain allocated long after the program has finished with it, since there is no guarantee that all the fields will be split. This makes strtok less than perfect, earning it a slightly dubious repulation.

Changing the Separators

Another curio is that we can actually change the second argument between calls - in effect changing the separator or separators that we wish to use. In the above example, we could use this technique to absorb the rest of the line up to the carriage return by changing the separator once the loop has completed; rather than setting the final character to a null.

To do this, of course, we would need to know the exact number of fields to be converted, so as to be able to stop at the appropriate point. It is, after all only an example, and there are probably far better reasons to want to change separators mid-processing.

Safe Use of strtok

To safeguard our use of strtok, there are two things we can do:

1. Check for null strings before the first call
2. Check the string is empty after the last call

The second is advisable, while the first is required, as strtok does not generally deal with null pointers very elegantly.

Conclusion & Further Reading

So, strtok should come with a health warning, but it is not quite the beast that it can often be made out to be. Quite the reverse - with careful handling it is a very useful piece of functionality. The programmer just needs to be sure that they really need it...

The *memcpy()* function

- Copies a block of “n” bytes from source to destination. It can be used to copy from one integer array to another.

```
#include <string.h>
#include <stdio.h>
int main( void )
{
   char buffer[] = "This is a test.", temp[100]=”12”;
   int a[5]={1,2,3,4,5}, b[5];

   printf( "Before: %s\n", temp );    //output: 12
   memcpy( buffer, temp, sizeof(buffer) );
   memcpy( a, b, sizeof(a));//now “a” and “b” array contains same data
   printf( "After:  %s\n", temp );    //output: This is a test.
}
```

The *memcmp()* function

- Compares the first **n** bytes from string **S1** to String **S2**

The *memset()* function

Sets buffers to a specified character.

```
#include <string.h>
#include <stdio.h>
int main( void )
{
   char buffer[] = "This is a test.";
   int a[20];

   printf( "Before: %s\n", buffer );  //output: This is a test.
   memset( buffer, '*', 4 );
   memset( a, '0', sizeof(a));        //initializes "a" array to 0
   printf( "After:  %s\n", buffer );  //output: **** is a test.
}
```

The *sprintf()* function

It can be used to convert integer or any data to character array. Function ***itoa()*** is not supported by ANSI C. So sprintf() function can be used instead of it.

```
int a=12;
char str[20];
sprintf(str,"%d",a);   //str[0]='1', str[1]='2', str[2]='\0';
printf("%s",str);
```

The *sscanf()* function

It can be used to take input from string other than standard input.

```
int a, b;
char str[20]="342 543";
sscanf(str,"%d %d",&a, &b);   //a=342, b=543
printf("%d %d",a,b);
```

-- x --