Programming with C

Terry Marris November 2010

3 String Functions

Previously we looked at string variables, their input, output and conversion to number types. Now we look at some string handling functions.

3.1 String Length

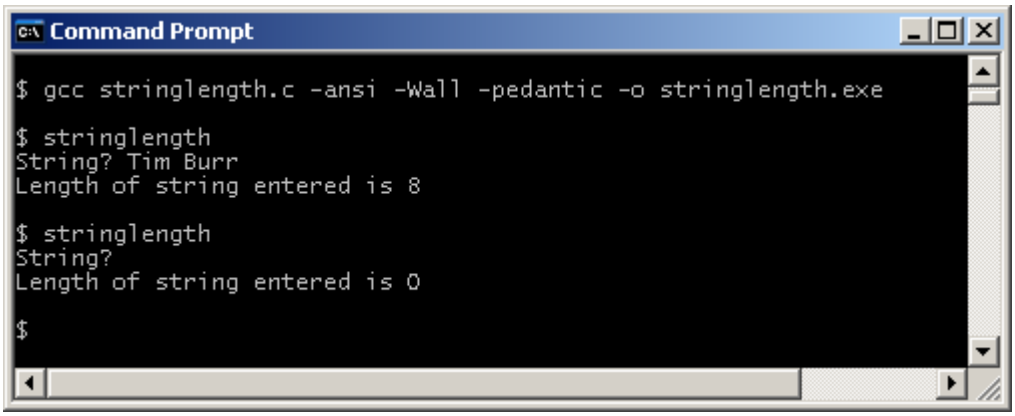
strlen() tells you how many characters there are in a string. The end-of-string character, known as *null* and denoted by *\0*, is not counted. *strlen()* is found in *string.h*.

```
/* stringlength.c: uses strlen() */

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

int main()
{
  char string[BUFSIZ];
  int length;

  printf("String? ");
  gets(string);
  length = strlen(string);
  printf("Length of string entered is %d\n", length);
  return 0;
}
```



A space is counted as a character and so *Tim Burr* has eight characters altogether. If just the return key is pressed, the number of characters in the string entered is zero.

3.2 String Copy

We look at two string copy functions.

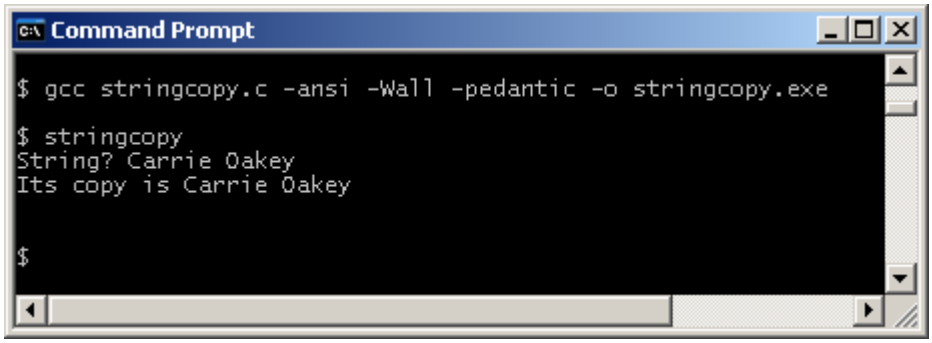
strcpy(destination, source) copies from *source* into *destination*. We need to be sure that *destination* is large enough. *strcpy()* is defined in *string.h*.

```
/* stringcopy.c: uses strcpy() */

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

int main()
{
  char original[BUFSIZ];
  char copy[BUFSIZ];

  printf("String? ");
  gets(original);
  strcpy(copy, original);
  printf("Its copy is %s\n", copy);
  printf("\n");
  return 0;
}
```



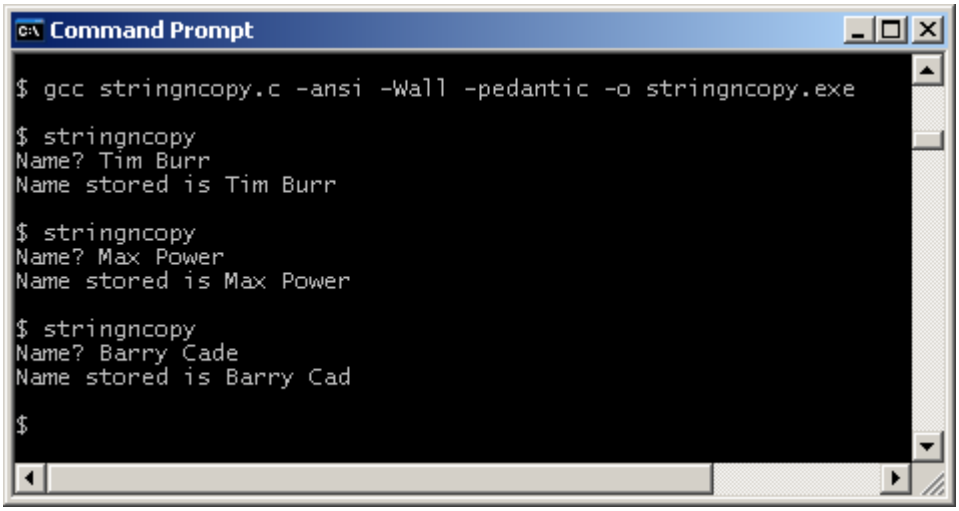
strncpy(destination, source, number) copies up to the given *number* of characters from *source* into *destination*. The characters copied may not contain the *null* character, so we always supply one. And, of course, *destination* must be large enough. *strncpy()* is defined in *string.h*.

```
/* stringncopy.c: uses strncpy() */

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

int main()
{
  char string[BUFSIZ];
  char name[10];  /* indexed 0..9 */

  printf("Name? ");
  gets(string);
  strncpy(name, string, 10);
  name[9] = '\0';  /* guarantees name is null terminated */
  printf("Name stored is %s\n", name);
  return 0;
}
```



Tim Burr has eight characters altogether and easily fits into *name*.

name

0	1	2	3	4	5	6	7	8	9
T	i	m		B	u	r	r	\0	\0

Max Power just fits, remembering that space is required for the *null* character. Many string-handling functions rely on finding this end-of-string marker.

name

0	1	2	3	4	5	6	7	8	9
M	a	x		P	o	w	e	r	\0

But *Barry Cade* comprises ten characters.

name

0	1	2	3	4	5	6	7	8	9
B	a	r	r	y		C	a	d	\0

The last character, *e*, is lost since *name* is too small to contain both *Barry Cade* and the *null* character.

3.3 String Concatenation

Concatenation involves adding one string onto the end of another. *strcat(first, second)* adds *second* onto the end of *first*. *strcat()* is defined in *string.h*.

First, we initialise two character arrays named *firstName* and *lastName*, and declare *fullName*.

```
char firstName[] = "Carrie";
char lastName[] = "Oakey";
char fullName[30];
```

Then we copy the first name into *fullName*.

```
strcpy(fullName, firstName);
```

Then we add a space.

```
strcat(fullName, " ");
```

Finally we add the last name.

```
strcat(fullName, lastName);
```

Here is the complete program and its run.

```
/* joinstrings.c: appends one string onto another */

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

int main()
{
  char firstName[] = "Carrie";
  char lastName[] = "Oakey";
  char fullName[30];

  strcpy(fullName, firstName);
  strcat(fullName, " ");
  strcat(fullName, lastName);
  printf("The full name is %s\n", fullName);
  return 0;
}
```

strcat() overwrites the *null* character in the first string when it copies the second string onto the end of the first string.

3.4 Splitting a String

In section 3.3 above we saw how to join one string onto the end of another. Now, we reverse the process. We split a name into its first and last names.

We initialise *fullName* and declare *firstName* and *lastName*.

```
char fullName[] = "Pearl Button";
char firstName[15];
char lastName[15];
```

In *Pearl Button* we see that a space separates *Pearl* from *Button*.

Then we use *strtok(fullName, " ")* to extract the text up to the first space character. That is *Pearl*.

We copy this token, a piece of text, into *firstName*.

```
strcpy(firstName, strtok(fullName, " "));
```

Then we use *strtok()* again, but this time with *NULL* instead of *fullName*, to extract the second token, *Button*, and copy it into *lastName*.

```
strcpy(lastName, strtok(NULL, " "));
```

NULL defines the null pointer. We discuss pointers later on. *NULL* is defined in *stddef.h*. *stddef* stands for standard definitions. Note that *NULL* the pointer is not the same as *null* the end-of-string character.

If there were another token to be retrieved we would use *strtok(NULL, " ")* again. *strtok()* is defined in *string.h*.

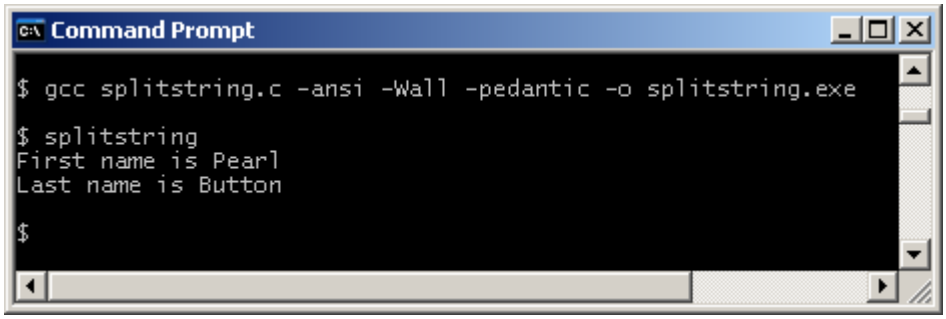
Here is the complete program and its run.

```
/* splitstring.c: splits a string into parts */

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

int main()
{
  char fullName[] = "Pearl Button";
  char firstName[15];
  char lastName[15];

  strcpy(firstName, strtok(fullName, " "));
  strcpy(lastName, strtok(NULL, " "));
  printf("First name is %s\n", firstName);
  printf("Last name is %s\n", lastName);
  return 0;
}
```



In

```
char line[] = "salt, pepper, vinegar, mustard";
```

we see that a comma followed by a space separates one word from the next. We would write *strtok(line, ", ")* to extract the first word, and *strtok(NULL, ", ")* to extract the next and

subsequent words. *strtok()* fails if there is no text to be split. We see how to deal with this in the chapter on validation.

Exercise 3.1

1. Write and test a program that inputs a string and outputs its length. By length we mean the number of characters it contains, excluding the null character.

2. Write and test a program that inputs two strings, and outputs the result of appending one string onto the end of another.

3. A postcode has two parts: the *outcode* and the *incode*. The two parts are separated by a space e.g. PO1 3AX. The outcode e.g. PO1 identifies the area and district. The incode e.g. 3AX identifies the building or group of buildings the post is destined for. Write and test a program that inputs a postcode, and outputs both the outcode and the incode. A postcode always has two, three or four characters for the outcode, and three characters for the incode.

We have seen how to use the string handling functions *strlen()*, *strcpy()*, *strncpy()*, *strcat()* and *strtok()*.

Next we see examine the numeric data type *double*.

Bibliography

Kernighan B and Ritchie D *The C Programming Language* Prentice Hall 1988
Mark Williams Company *ANSI C A Lexical Guide* Prentice Hall 1988
www.cabinetoffice.gov.uk/govtalk/schemasstandards/e-gif/datastandards/address/postcode.aspx accessed Nov 2010