Introduction to Strings

Programming is about more than just numbers & arithmetic! Words, sentences, paragraphs: in computer-speak, represented with "strings". You've already seen the use of **`char`**: a single character.

A string is an array of characters ending with the NULL character

-- the NULL character is written **`'\0'`** and has ASCII code 0

-- note difference between NULL and the number **0**, which has ASCII code 48

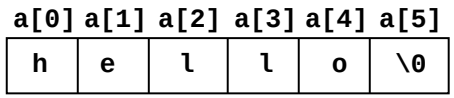
A **string constant** is enclosed in double quotes, e.g. **`"hello"`**, **`"I like C"`**

-- so far, all we've done is print them out, e.g. **`printf("hello");`**

To declare and initialise a string:

```
char a[6] = {'h','e','l','l','o','\0'};
```

alternatively, `char a[6] = "hello"; /* more natural way */`



Introduction to Strings (contd.)

Don't have to specify size of array, if string is initialised when it's declared:

```
char a[] = "hello"; /* same effect as before */
```

What if array size is larger than current string value?

```
char a[10] = "hello";
printf("string is: %s\n", a); /* use %s for string */
printf("values in char array are: ");
for (i=0;i<10;i++) {printf("%c, ASCII value %d\n",a[i],(int)a[i]);}
```

Produces the screen output:

```
string is: hello
values in char array are: h, ASCII value 104
e, ASCII value 101
l, ASCII value 108
l, ASCII value 108
o, ASCII value 111
, ASCII value 0
, ASCII value 0
, ASCII value 0
, ASCII value 0
, ASCII value 0
```

String-terminating NULL

Here, all remaining array elements have also been set to NULL. In general, such values are system-dependent.

Strings and pointers

Since a string is stored as an array of characters, the name of the string is treated by the C compiler as a **fixed** pointer-to-**`char`** whose value is the address of the first array element.

We can also declare a variable of type pointer-to-**`char`** and assign it the address of the first element of a character array (i.e. a string). The difference between this pointer and the "name of string" pointer is that the latter cannot be reassigned a different value.

```
char a[] = "hello";
char *ptr1 = &a[0];
char *ptr2 = a; /* a == &a[0] */
printf("string is: %s\n",a);
printf("string is stored at location %u\n",a);
printf("string is also: %s\n",ptr1);
printf("string is also also: %s\n",ptr2);
ptr1 = "hi there"; /* pointer a can't be reassigned like this */
printf("string is still: %s\n",a);
printf("but now ptr1 points to the string: %s\n",ptr1);
```

used %s, so print out string contents

used %u, so print out location of string (i.e. address of first element)

Produces the screen output:

```
string is: hello
string is stored at location 30296
string is also: hello
string is also also: hello
string is still: hello
but now ptr1 points to the string: hi there
```


String input

We can use **`%s`** to read in a string from the keyboard:

```
char message1[80], message2[80];
printf("enter first string: ");
scanf("%s", message1); /* don't need & */
printf("first string is: %s\n", message1);
printf("enter second string: ");
scanf("%s", message2);
printf("second string is: %s\n", message2);
```

Produces the screen output:

```
enter first string: hello
first string is: hello
enter second string: hi there
second string is: hi
```

Problem: **`scanf()`** stops reading input when it reaches a whitespace character, such as the blank space between "hi" and "there". In this case, **`scanf()`** only read in the string "hi"; a second **`scanf()`** statement would read in the string "there".

This is a problem if we want to read in a complete line of text, since in general we don't know in advance how many individual strings are contained in the line…

String input (contd.)

Solution for reading in a complete line of text: read input character-by-character, stopping only when the newline character **'\n'** is encountered.

```
int i=0;
char message[80];
printf("enter string: ");
scanf("%c",&message[i]); /* read in first input character */
while (message[i]!='\n'){
  i++;
  scanf("%c",&message[i]); /* read in next input character */
}
message[i]='\0'; /* terminate string with NULL */
printf("string is: %s\n", message);
```

Produces the screen output:

```
enter string: hi there
string is: hi there
```

This solution is a bit complicated (!). There is an easier way – ***C has special functions for inputting and outputting a single character:***

```
int getchar(void); /* returns ASCII value of next input character */
int putchar(int c); /* outputs character whose ASCII value is c */
```

Example: input a single character and immediately echo it to the screen

```
char ch = getchar(); /* read in a character from the keyboard */
putchar(ch); /* then output this character to the screen */
```


String input (contd.)

So another solution for reading in a complete line of text is:

```
int i=0;
char message[80];
printf("enter string: ");
while ((message[i]=getchar())!='\n'){
  i++;
}
message[i]='\0'; /* terminate string with NULL */
printf("string is: %s\n", message);
```

Produces the screen output:

```
enter string: hi there
string is: hi there
```

Q: what happens if you forget to NULL-terminate the string?

Strings and functions

Since the name of the string is treated as a pointer to the first element of the string, function calls involving strings use **call-by-address** (just like arrays):

```
#include "stdio.h"
void main(void){
 char message1[80]="hello";
 char *pmsg2="I like C";
 int strlen1(char str[]); /* function prototype */
 printf("\"%s\" has length %d\n",message1,strlen1(message1));
 printf("\"%s\" has length %d\n",pmsg2,strlen1(pmsg2));
}
int strlen1(char s[]){ /* no need to pass in size of array */
 int count=0;
 while (s[count]!='\0'){
   count++; /* increase by 1 as long as NULL not reached */
 }
 return count;
}
```

s ***is the local name in*** `strlen1()` ***for the string in*** `main()` ***given as the actual parameter in the function call***

Produces the screen output:

```
"hello" has length 5
"I like C" has length 8
```


Strings and functions (contd.)

Instead of the called function using a character array, it could use a pointer variable:

```
#include "stdio.h"
void main(void){
 char message1[80]="hello";
 char *pmsg2="I like C";
 int strlen2(char *str); /* function prototype */
 printf("\"%s\" has length %d\n",message1,strlen2(message1));
 printf("\"%s\" has length %d\n",pmsg2,strlen2(pmsg2));
}
int strlen2(char *ps){ /* no need to pass in size of array */
 int count=0;
 while (*ps!='\0'){
   count++; /* increase by 1 as long as NULL not reached */
   ps++; /* move pointer on to next element */
 }
 return count;
}
```

ps ***is a local pointer in*** `strlen2()` ***pointing to the string in*** `main()` ***given as the actual parameter in the function call***

Produces the screen output:

```
"hello" has length 5
"I like C" has length 8
```

Strings and functions (contd.)

In either case, since the location of the string is passed by the calling function, the called function can change the contents of the string which is given as the actual parameter:

```
#include "stdio.h"
void main(void){
 char message[80]="hello";
 void bye(char *str); /* function prototype */
 printf("string is \"%s\"\n",message);
 bye(message); /* function call */
 printf("now string is \"%s\"\n",message);
}
void bye(char *p){
 *p='g';*(p+1)='o';*(p+2)='o';*(p+3)='d';
 *(p+4)='b';*(p+5)='y';*(p+6)='e';*(p+7)='\0';
}
```

Produces the screen output:

```
string is "hello"
now string is "goodbye"
```


Library string functions

The standard C library contains several useful functions for manipulating and examining strings. Must **`#include <string.h>`** to get them to work.

Suppose **`ps`** is a pointer to the string **`s`**, and **`pt`** is a pointer to the string **`t`**

```
strlen(ps)
 /* returns integer == length of s (not counting NULL) */

strcmp(ps,pt)
 /* returns negative integer if s<t (alphabetically), returns */
 /* 0 if s==t, and returns positive integer if s>t            */
```

[Alphabetically: a<b, an<at, at = = at, z>x, fire>field, etc.]

```
strcpy(ps,pt)
 /* copies contents of t into s, overwriting previous contents */
 /* of s, and returns pointer to new 1st character of s       */
```

Use these string library functions whenever possible! There are other string library functions you may find useful; see any C textbook (or the C standard library) for details…

Library string functions: Example

```
#include <stdio.h>
#include <string.h>
void main(void){
  int i, j;
  char *p_result;
  char s[]="C is easy to learn", t[]="C is hard to learn";
  char *ps = s, *pt = t;
  printf("length of s is %d\n",strlen(ps));
  i=strlen(pt);
  printf("length of t is %d\n",i);
  j=strcmp(ps,pt);
  if (j<0) {printf("string s comes before string t\n");}
    else if (j==0) {printf("strings equal\n");}
      else {printf("string s comes after string t\n");}
  printf("s is \"%s\"\n",s);
  p_result=strcpy(ps,pt);
  printf("now s is \"%s\"\n",s);
  printf("also: s is \"%s\"\n",p_result);
}
```

Produces the screen output:

```
length of s is 18
length of t is 18
string s comes before string t
s is "C is easy to learn"
now s is "C is hard to learn"
also: s is "C is hard to learn"
```


String application: “uppercaseify”

Problem: write a C program which changes every lowercase letter in a string into uppercase.

```
#include <stdio.h>
void main(void){
 int i;
 char s[]="This is just a Test";
 printf("string is \"%s\"\n",s);
 for (i=0;s[i]!='\0';i++){ /* keep going until NULL reached */
   if ((s[i]>='a') && (s[i]<='z')){ /* lowercase letter only */
     s[i] = s[i] + 'A' - 'a'; /* uses ASCII values… */
   }
 }
 printf("now string is \"%s\"\n",s);
}
```

Produces the screen output:

```
string is "This is just a Test"
now string is "THIS IS JUST A TEST"
```

Reminder: this works because in ASCII, a-z and A-Z follow each other in sequence (so do 0-9).
If s[i] is ‘a’: s[i]+’A’-’a’ = = ‘a’+’A’-’a’ = = ‘A’.
If s[i] is ‘b’: s[i]+’A’-’a’ = = ‘b’+’A’-’a’ = = ‘a’+1+’A’-’a’ = = ‘A’+1 = = ‘B’.
If s[i] is ‘c’: s[i]+’A’-’a’ = = ‘c’+’A’-’a’ = = ‘a’+2+’A’-’a’ = = ‘A’+2 = = ‘C’, etc.