C language

Imad Kissami[1]

[1]Mohammed VI Polytechnic University, Benguerir, Morocco

MOHAMMED VI
POLYTECHNIC
UNIVERSITY

Strings

Introduction

- We've used strings

```
1 printf("hello");
```

- "Hello" is string literal constant
- Array with base type char
  - One character per element
  - One extra character: '\0'
    * Called 'null character'
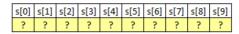   * End marker
  - Literal "Hello" stored as string

| H | e | l | l | o | \0 |

Strings

String Variable Declaration

- Array of characters:

```
1 char s[10];
```

- Declares a c-string variable to hold up to 9 characters plus one null character
- No initial value

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
?	?	?	?	?	?	?	?	?	?

Strings

String Variable

- Typically a partially filled array
  - Declare large enough to hold max-size string, including the null character.
  - Given a standard array:

  ```
  1 char s[10];
  ```

  - If s contains string "Hi Mom!", then stored as:

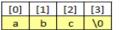 | s[0] | s[1] | s[2] | s[3] | s[4] | s[5] | s[6] | s[7] | s[8] | s[9] |
  |------|------|------|------|------|------|------|------|------|------|
  | H | i | | M | o | m | ! | \0 | ? | ? |

Strings

- Can initialize string:

```
1 char s[15] = "Hi There";
```

  - Need not fill entire array
  - Initialization places '\0' at end

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
H	i		T	h	e	r	e	\0	?	?	?	?	?	?

- Can omit array-size:
  - Automatically makes size one more than length of quoted string

```
1 char abc[] = "abc";
```

[0]	[1]	[2]	[3]
a	b	c	\0

  - NOT same as:

```
1 char abc[] = {'a', 'b', 'c'};
```

[0]	[1]	[2]
a	b	c

  - IS same as:

```
1 char abc[] = {'a', 'b', 'c', '\0'};
```

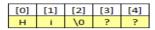| [0]	[1]	[2]	[3]
a	b	c	\0

Strings

String Indexes

- A string IS an array
- Can access indexed variables of:

```
1 char s[5] = "Hi";
```

- hi[0] is 'H'
- hi[1] is 'i'
- hi[2] is '\0'
- hi[3] is unknown
- hi[4] is unknown

[0]	[1]	[2]	[3]	[4]
H	i	\0	?	?

Strings

String Index Manipulation

- Can manipulate array elements

```
1 char s[7] = "DoBeDe";
2 s[5] = 'o';
3 s[6] = '!';
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	
D	o	B	e	D	e	\0	?
D	o	B	e	D	o	\0	?
D	o	B	e	D	o	!	?

- Be careful!
- Here, '\0' (null) was overwritten by a '!'
- If null overwritten, string no longer 'acts' like a string!
  - Unpredictable results!

Strings
String Library

- Used for string manipulations
    - Normally want to do 'fun' things with strings
    - Requires library string.h:

```
1 #include "string.h"
```

    - http://en.wikipedia.org/wiki/String.h

Strings

String Length: strlen

- Often useful to know length of string

```
1 strlen(string)
```

- Returns number of characters
  * Does not include null
  * Return type is size_t so type cast may be required

```
1 char hello_world[] = "Hello World";
2 printf("Size is %d", (int)strlen(hello_world));
```

- Output:

```
1 Size is 11
```

Strings

- Strings are not like other variables, they are arrays
  - Cannot assign:

```
1 char s[10];
2 s = "Hello"; // ILLEGAL!
```

- Must use string library function for assignment:

```
1 strcpy(destination, source)
```

  - NO checks for size – up to programmer!
  - 'Assign' value of msg to "Hello":

```
1 strcpy(msg, "Hello");
```

- Or:

```
1 strncpy(destination, source, limit)
```

  - No ending null character if limit is reached

Strings

- Cannot use operator == to compare

```
1 char c1[] = "Hello";
2 char c2[] = "GoodBye";
3
4 if (c1 == c2) // NOT ALLOWED
```

- Must use strcmp string library function to compare:

```
1 strcmp(string1, string2)
```

- Returns zero int if string1 is equal to string 2
- Returns <0 int if string1 is less than string2
- Returns >0 int if string1 is greater than string2

```
1 if (strcmp(c1, c2) == 0)
2     print("c1 equal to c2");
3 else if (strcmp(c1, c2) < 0)
4     print("c1 less than c2");
5 else:
6     print("c1 greater than c2");
```

Strings

- Appends one string onto end of another

```
1 strcat(destination, source)
```

```
1 char c1[30] = "Hello";
2 char c2[30] = "Hello";
3
4 strcat(c1, "World"); // Result "HelloWorld"
5 strcat(c2, " World"); // Result "Hello World"
```

- Be careful when concatenating words
  * msg1 is missing space after Hello
  * msg2 is correct

Strings

String Parameters to Functions

- A string is an array, so
  - String parameter is an array parameter
  - Strings passed to a function can be changed by the receiving function!
- Like all arrays, typical to send size as well
  - Function could also use '\0' to find end

```c
char msg[] = "Hello";
int msg_len = strlen(msg);

str_reverse(msg, msg_len);
```

Strings

String Input and Output

- Watch input size of string
  - Must be large enough to hold entered string!
    * + '\n' perhaps
    * + '\0'
  - C gives no warnings of input size issues!

```
1 const int MAX_INPUT_STRING = 50;
2 char input_string[MAX_INPUT_STRING + 2];
```

- Functions in stdio.h

Strings
Character Input: getchar

■ Reads one character at a time from a text stream

```
1 int getchar()
```

- Reads the next character from the standard input stream and returns its value
- Return type is int!
  * Will convert if assigned to char

```
1 char in_ch;
2 in_ch = getchar();
```

Strings

Character Output: %s and putchar

- Format string placeholder for string: %s
- putchar: Writes one character at a time

```
1 int putchar (int outChar)
```

- Writes the parameter to standard output
- If successful, returns the character written

```
1 char msg[] = "dlroW olleH";
2 int index;
3
4 // Print  dlroW olleH
5 printf("%s\n", msg);
6
7 // Print Hello World
8 for    (index = (int) strlen(msg) -1 ; index >=0; index--)
9     putchar(msg[index]);
10 printf("\n");
```

- Output:

```
1 dlroW olleH
2 Hello World
```

Strings

String Input: gets

```
1 char *gets (char *strPtr)
```

- Inputs a line (terminated by a newline) from standard input
- Converts newline to \0
- If successful, returns the string and also places it in argument
- Warning: Does not check length of input
    - gcc may produce warning message

```
1 char input_msg[100];
2 gets(input_msg);
```

```
1 $ gcc toto.c
2 toto.c: In function 'main':
3 ...
4 avertissement : the 'gets' function is dangerous and should not be used
```

Strings

String Input: fgets

```
1 char *fgets (char * strPtr, int size, FILE *fp)
```

- Inputs characters from the specified file pointer through \n or until specifed size is reached
- Puts newline (\n) in the string if size not reached!!!
- Appends \0 at the end of the string
- If successful, returns the string & places in argument

```
 1 const int MAX_LINE = 100;
 2 char line_in[MAX_LINE + 2 ];
 3 int line_len;
 4
 5 printf("Enter a string: \n");
 6 fgets(line_in, MAX_LINE, stdin);
 7
 8 // check for \n
 9 line_len = strlen(line_in);
10
11 if (line_in[line_len-1] == '\n' )
12     line_in[line_len-1] = '\0';
13
14 printf("The string is: %s\n", line_in);
```

```
1 $ ./a.out
2 Enter a string:
3 Hello World
4 The string is: Hello World
```

Strings

String Output: puts

```
1 int puts (const char *strPtr)
```

- Takes a null-terminated string from memory and writes it to standard output
- Writes \n in place of \0

```
1 char hello[] = "Hello";
2 puts(hello);
3
4 printf("-------\n");
```

```
1 $ ./a.out
2 Hello
3 -------
```

Strings

String Output: fputs

```
1 int fputs (const char *strPtr, FILE *fp)
```

- Takes a null-terminated string from memory and writes it to the specified file pointer
- Drops \0
- Programmer's responsibility: Make sure the newline is present at the appropriate place(s)

```
1 char line_out[100] = "Hello!\n";
2 fputs(hello, stdout);
```

```
1 $ ./a.out
2 Hello!
```