# C language

Imad Kissami[1]
[1]Mohammed VI Polytechnic University, Benguerir, Morocco

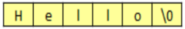
# Strings

## Introduction

- We've used strings

```
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



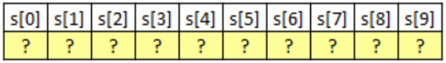
## Strings

### String Variable Declaration

- Array of characters:

```
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] |
|---|---|---|---|---|---|---|---|---|---|
| ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |

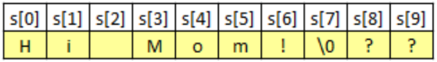
# Strings

## String Variable

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

```c
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 | ? | ? |

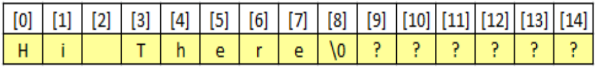
# Strings

## String Variable Initialization

- Can initialize string:

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

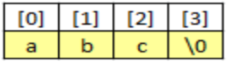
  - Need not fill entire array
  - Initialization places '\0' at end



| [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

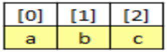
```c
char abc[] = "abc";
```



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

  - NOT same as:

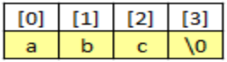
```c
char abc[] = {’a’, ’b’, ’c’};
```



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

  - IS same as:

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



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

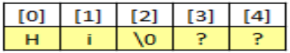
# Strings

## String Indexes

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

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

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



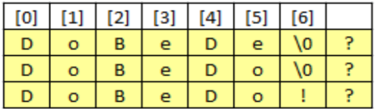
| [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|
| H | i | \0 | ? | ? |

# Strings

## String Index Manipulation

- Can manipulate array elements

```
char s[7] = "DoBeDe";
s[5] = 'o';
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:

```c
#include "string.h"
```

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

# Strings

## String Length: strlen

- Often useful to know length of string

```
strlen(string)
```

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

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

  - Output:

```
Size is 11
```

## Strings

### = with strings

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

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

- Must use string library function for assignment:

```c
strcpy(destination, source)
```

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

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

- Or:

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

  - No ending null character if limit is reached

## Strings

### == with strings

- Cannot use operator == to compare

```
char c1[] = "Hello";
char c2[] = "GoodBye";

if (c1 == c2) // NOT ALLOWED
```

- Must use strcmp string library function to compare:

```
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

```
if (strcmp(c1, c2) == 0)
    print("c1 equal to c2");
else if (strcmp(c1, c2) < 0)
    print("c1 less than c2");
else:
    print("c1 greater than c2");
```

## Strings

### String Concatenate: strcat

- Appends one string onto end of another

```c
strcat(destination, source)
```

```c
char c1[30] = "Hello";
char c2[30] = "Hello";

strcat(c1, "World"); // Result "HelloWorld"
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!

```
const int MAX_INPUT_STRING = 50;
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

```
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

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

# Strings

## Character Output: %s and putchar

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

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

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

```
char msg[] = "dlroW olleH";
int index;

// Print  dlroW olleH
printf("%s\n", msg);

// Print Hello World
for (index = (int) strlen(msg) -1 ; index >=0; index--)
    putchar(msg[index]);
printf("\n");
```

  - Output:

```
dlroW olleH
Hello World
```

## Strings

### String Input: gets

```c
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

```c
char input_msg[100];
gets(input_msg);
```

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

## Strings

### String Input: fgets

```c
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

```c
const int MAX_LINE = 100;
char line_in[MAX_LINE + 2 ];
int line_len;

printf("Enter a string: \n");
fgets(line_in, MAX_LINE, stdin);

// check for \n
line_len = strlen(line_in);

if (line_in[line_len-1] == '\n' )
    line_in[line_len-1] = '\0';

printf("The string is: %s\n", line_in);
```

```
$ ./a.out
Enter a string:
Hello World
The string is: Hello World
```

## Strings

### String Output: puts

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

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

```
char hello[] = "Hello";
puts(hello);

printf("-------\n");
```

```
$ ./a.out
Hello
-------
```

# Strings

## String Output: fputs

```c
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)

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

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