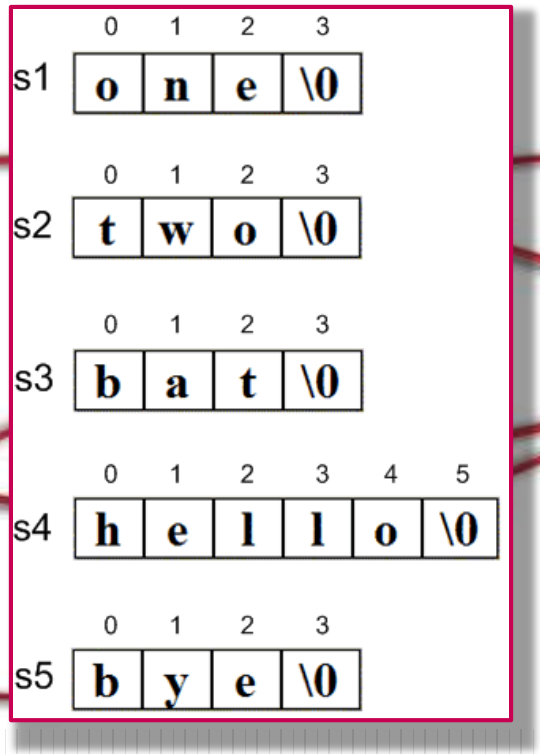
Programming in C

Chapter 9
Strings



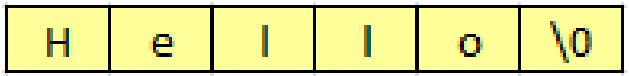
Strings

- 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



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]
?	?	?	?	?	?	?	?	?	?

String Variable

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

```
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	?	?

String Variable Initialization

- Can initialize string:

```
char message[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	?	?	?	?	?	?

String Variable Initialization

- Can omit array-size:

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

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

  - Automatically makes size one more than length of quoted string
  - NOT same as:

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

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

  - IS same as:

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

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

String Indexes

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

```
char hi[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	?	?

String Index Manipulation

- Can manipulate array elements

```
char dobedo[7] = "DoBeDe";
dobedo[5] = 'o';
dobedo[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!

String Library

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

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

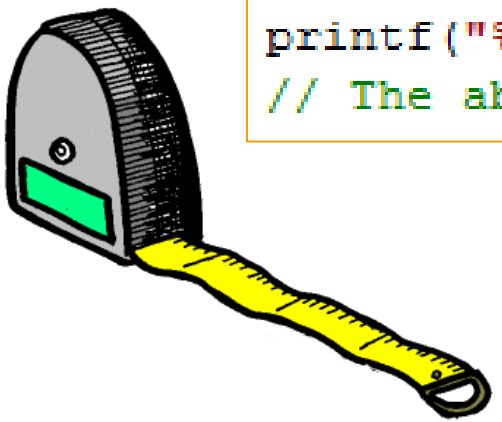
http://en.wikipedia.org/wiki/String.h

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("%d", (int) strlen(hello_world));
// The above will print number 11
```

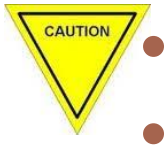
= with strings

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

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

- Must use string library function for assignment:

`strcpy(destination, source)`



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

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

- Or **`strncpy(destination, source, limit)`**
  - No ending null character if limit is reached

== with strings

- Cannot use operator == to compare

```
char hello[] = "Hello";
char goodbye[] = "Goodbye";
if (hello == goodbye)   // 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(hello, goodbye) == 0)
   printf("Hello equal to Goodbye");
else if (strcmp(hello, goodbye) < 0)
   printf("Hello less than Goodbye");
else
   printf("Hello greater than Goodbye");
```

String Concatenate: strcat

- Appends one string onto end of another

`strcat(destination, source)`

```
char msg1[30] = "Hello";
char msg2[30] = "Hello";
strcat(msg1, "World");  // Result "HelloWorld"
strcat(msg2, " World"); // Result "Hello World"
```

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

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

```
char msg[] = "BOB";
int msg_len = strlen(msg);
str_reverse(msg, msg_len);
```

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

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();
```

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 ndx;
// Print dlroW olleH
printf("%s\n", msg);
// Print Hello World
for (ndx = (int) strlen(msg) - 1; ndx >= 0; ndx--)
   putchar(msg[ndx]);
printf("\n");
```

String Input: gets

String variable

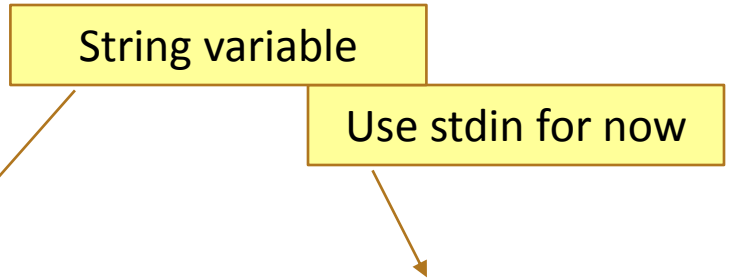
```
char *gets (char *strPtr)
```

- 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

```
char input_string[100];
gets(input_string);
```

```
warning: the `gets' function is dangerous and should not be used.
```

String Input: fgets




```
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

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

String Output: puts

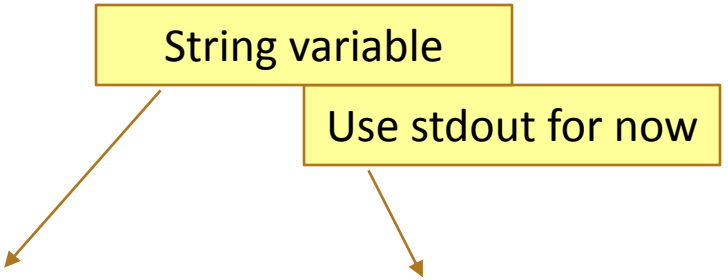
String variable or constant

```
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");
/*
    Prints:
       hello
       -----
*/
```

String Output: fputs




```
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)

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

Programming in C

Chapter 9
Strings

THE END