



# PROGRAMMING METHODOLOGY (PHƯƠNG PHÁP LẬP TRÌNH)

---

## UNIT 16: Characters and Strings

# Unit 16: Characters and Strings

## Objectives:

- Declare and manipulate data of `char` data type
- Learn fundamental operations on strings
- Write string processing programs

## References:

- Lesson 1.4.1 Characters and Symbols
- Chapter 7: Strings and Pointers

# Unit 16: Characters and Strings (1/2)

## 1. Motivation

## 2. Characters

2.1 ASCII Table

2.2 Demo #1: Using Characters

2.3 Demo #2: Character I/O

2.4 Demo #3: Character Functions

2.5 Common Error

## 3. Strings

3.1 Basics

3.2 String I/O

3.3 Demo #4: String I/O

3.4 Demo #5: Remove Vowels

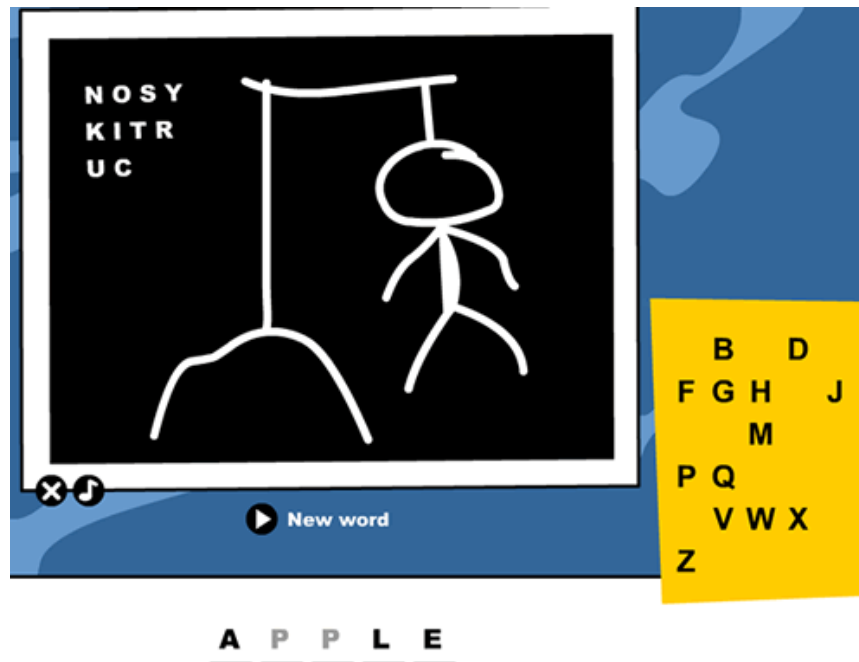
3.5 Demo #6: Character Array without terminating '\0'

# Unit 16: Characters and Strings (2/2)

4. String Functions
5. Pointer to String
6. Array of Strings
7. Demo #7: Using String Functions
8. Strings and Pointers

# 1. Motivation

- Why study characters and strings?
- **Hangman** game – Player tries to guess a word by filling in the blanks. Each incorrect guess brings the player closer to being “hanged”
- Let's play! <http://www.hangman.no/>



## 2. Characters

- In C, single **characters** are represented using the data type **char**
- **Character constants** are written as symbols enclosed in single quotes
  - Examples: **'g'**, **'8'**, **'\*'**, **' '**, **'\n'**, **'\0'**
  - Recall: Week 3 Exercise #7 NRIC Check Code
- Characters are stored in one byte, and are encoded as numbers using the **ASCII** scheme.
- *ASCII (American Standard Code for Information Interchange)*, is one of the document coding schemes widely used today.
- *Unicode* is another commonly used standard for multi-language texts.

## 2.1 Characters: ASCII Table

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
10	lf	vt	ff	cr	so	si	dle	dcl	dc2	dc3
20	cd4	nak	syn	etb	can	em	sub	esc	fs	gs
30	rs	us	sp	!	"	#	\$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	}		~	del		

For example,  
character 'O' is  
79 (row value  
70 + col value 9  
= 79).

## 2.2 Demo #1: Using Characters (1/2)

```
// Unit16_CharacterDemo1.c
#include <stdio.h>
```

```
int main(void) {
```

Declaring and initialising  
char variables.

```
    char grade = 'A', newgrade, ch;
    int value;
```

Using %c

```
    printf("grade = %c\n", grade);
    newgrade = grade + 2;
    printf("newgrade = %c\n", newgrade);
    printf("newgrade = %d\n", newgrade);
```

```
    value = 65;
    printf("value = %d\n", value);
    printf("value = %c\n", value);
```

Relationship between  
character and integer.

Unit16\_CharacterDemo1.c

grade = A  
newgrade = C  
newgrade = 67

value = 65  
value = A



## 2.2 Demo #1: Using Characters (2/2)

Comparing characters.

```
if ('A' < 'c')
    printf("'A' is less than 'c'\n");
else
    printf("'A' is not less than 'c'\n");

for (ch = 'p'; ch <= 't'; ch++)
    printf("ch = %c\n", ch);

return 0;
}
```

Using character variable  
as a loop variable.

'A' is less than 'c'

ch = p  
ch = q  
ch = r  
ch = s  
ch = t

ASCII value of 'A'  
is 65. ASCII  
value of 'c' is 99.

## 2.3 Demo #2: Character I/O

- Besides `scanf()` and `printf()`, we can also use `getchar()` and `putchar()`. Note how they are used below:

Unit16\_CharacterDemo2.c

```
// Unit16_CharacterDemo2.c
#include <stdio.h>
```

```
int main(void) {
    char ch;
```

Read a character  
from stdin.

```
    printf("Enter a character: ");
    ch = getchar();
```

Enter a character: **W**  
Character entered is **W**

```
    printf("The character entered is ");
    putchar(ch);
    putchar('\n');
```

```
    return 0;
```

```
}
```

Print a character  
to stdout.

## 2.4 Demo #3: Character Functions

- Must include `<ctype.h>` to use these functions.

Unit16\_CharacterDemo3.c

```
// Unit16_CharacterDemo3.c
#include <stdio.h>
#include <ctype.h>
int main(void) {
    char ch;

    printf("Enter a character: ");
    ch = getchar();
    if (isalpha(ch)) {
        if (isupper(ch)) {
            printf("'%c' is a uppercase-letter.\n", ch);
            printf("Converted to lowercase: %c\n", tolower(ch));
        }
        if (islower(ch)) {
            printf("'%c' is a lowercase-letter.\n", ch);
            printf("Converted to uppercase: %c\n", toupper(ch));
        }
    }
    if (isdigit(ch)) printf("'%c' is a digit character.\n", ch);
    if (isalnum(ch)) printf("'%c' is an alphanumeric character.\n", ch);
    if (isspace(ch)) printf("'%c' is a whitespace character.\n", ch);
    if (ispunct(ch)) printf("'%c' is a punctuation character.\n", ch);
    return 0;
}
```


Download this program and test it out.  
For a complete list of character functions,  
refer to the Internet (eg:  
<http://www.csd.uwo.ca/staff/magi/175/refs/char-funcs.html>)

Note that  
`tolower(ch)` and  
`toupper(ch)` do  
NOT change `ch`!


## 2.5 Characters: Common Error

- A character variable named **z** does not means it is equivalent to 'z' or it contains 'z'!


```
char A, B, C, D, E;  
  
if (marks >= 80)  
    return A;  
else if (marks >= 70)  
    return B;  
else if (marks >= 60)  
    return C;  
. . .
```



```
if (marks >= 80)  
    return 'A';  
else if (marks >= 70)  
    return 'B';  
else if (marks >= 60)  
    return 'C';  
. . .
```



```
char grade;  
if (marks >= 80)  
    grade = 'A';  
else if (marks >= 70)  
    grade = 'B';  
else if (marks >= 60)  
    grade = 'C';  
. . .  
return grade;
```



### 3. Strings

- We have seen arrays of numeric values (types `int`, `float`, `double`)
- We have seen `string constants`
  - `printf("Average = %.2f", avg);`
  - `#define ERROR "*****Error -"`
- A `string` is an array of characters, terminated by a null character `\0` (which has ASCII value of zero)

c	s	1	0	1	0	\0
---	---	---	---	---	---	----

## 3.1 Strings: Basics

### ■ Declaration an array of characters

- `char str[6];`

### ■ Assigning character to an element of an array of characters

- `str[0] = 'e';`
- `str[1] = 'g';`
- `str[2] = 'g';`
- `str[3] = '\0';`



Without '\0', it is just an array of character, not a string.

Do not need '\0' as it is automatically added.

### ■ Initializer for string

#### ■ Two ways:

- `char fruit_name[] = "apple";`
- `char fruit_name[] = {'a', 'p', 'p', 'l', 'e', '\0'};`

## 3.2 Strings: I/O (1/2)

### ■ Read string from stdin

- `fgets(str, size, stdin)` // reads size - 1 char,  
// or until newline
- `scanf("%s", str);` // reads until white space

### ■ Print string to stdout

```
puts(str); // terminates with newline  
printf("%s\n", str);
```

Note: There is another function `gets(str)` to read a string interactively. However, due to security reason, we avoid it and use `fgets()` function instead.

## 3.2 Strings: I/O (2/2)

- `fgets()`
  - On interactive input, `fgets()` also reads in the newline character

User input: **eat**

e	a	t	\n	\0	?	?
---	---	---	----	----	---	---

- Hence, we may need to replace it with `'\0'` if necessary

```
fgets(str, size, stdin);  
len = strlen(str);  
if (str[len - 1] == '\n')  
    str[len - 1] = '\0';
```



## 3.3 Demo #4: String I/O

Unit16\_StringIO1.c

```
#include <stdio.h>
#define LENGTH 10

int main(void) {
    char str[LENGTH];
```

Test out the programs with this input:  
**My book**

```
    printf("Enter string (at most %d characters): ", LENGTH-1);
    scanf("%s", str);
    printf("str = %s\n", str);
    return 0;
}
```

Output:  
**str = My**

```
#include <stdio.h>
#define LENGTH 10

int main(void) {
    char str[LENGTH];
```

Output:  
**str = My book**

Unit16\_StringIO2.c

```
    printf("Enter string (at most %d characters): ", LENGTH-1);
    fgets(str, LENGTH, stdin);
    printf("str = ");
    puts(str);
    return 0;
}
```

Note that puts(str) adds  
a newline automatically.

## 3.4 Demo #5: Remove Vowels (1/2)

- Write a program `Unit16_RemoveVowels.c` to remove all vowels in a given input string.
- Assume the input string has at most 100 characters.
- Sample run:

```
Enter a string: How HAVE you been, James?  
Changed string: Hw HV y bn, Jms?
```

### 3.4 Demo #5: Remove Vowels (2/2)

Unit16\_RemoveVowels.c

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main(void) {
    int i, len, count = 0;
    char str[101], newstr[101];

    printf("Enter a string (at most 100 characters): ");
    fgets(str, 101, stdin); //what happens if you use scanf() here?
    len = strlen(str); // strlen() returns number of char in string
    if (str[len - 1] == '\n')
        str[len - 1] = '\0';
    len = strlen(str); // check length again

    for (i=0; i<len; i++) {
        switch (toupper(str[i])) {
            case 'A': case 'E':
            case 'I': case 'O': case 'U': break;
            default: newstr[count++] = str[i];
        }
    }
    newstr[count] = '\0';
    printf("New string: %s\n", newstr);
    return 0;
}
```

Need to include `<string.h>`  
to use string functions such  
as `strlen()`.

## 3.5 Demo #6: Character Array without terminating '\0'

- What is the output of this code?

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

```
int main(void) {
    char str[10];
```

```
    str[0] = 'a';
    str[1] = 'p';
    str[2] = 'p';
    str[3] = 'l';
    str[4] = 'e';
```

```
    printf("Length = %d\n", strlen(str));
    printf("str = %s\n", str);
```

```
    return 0;
```

```
}
```

Unit16\_without\_null\_char.c

One possible output:

**Length = 8**  
**str = apple;<**

Compare the output if you add:

**str[5] = '\0';**

or, you have:

**char str[10] = "apple";**

printf() will print %s from the starting address of str until it encounters the '\0' character.



%s and string functions work only on "true" strings. Without the terminating null character '\0', string functions will not work properly.

## 4. String Functions (1/3)

- C provides a library of string functions
  - Must include `<string.h>`
  - Table 7.3 (pg 509 – 514)
  - [http://www.edcc.edu/faculty/paul.bladek/c\\_string\\_functions.htm](http://www.edcc.edu/faculty/paul.bladek/c_string_functions.htm)
  - <http://www.cs.cf.ac.uk/Dave/C/node19.html>
  - and other links you can find on the Internet
- **strcmp(s1, s2)**
  - Compare the ASCII values of the corresponding characters in strings s1 and s2.
  - Return
    - a negative integer if s1 is lexicographically less than s2, or
    - a positive integer if s1 is lexicographically greater than s2, or
    - 0 if s1 and s2 are equal.
- **strncmp(s1, s2, n)**
  - Compare first n characters of s1 and s2.

## 4. String Functions (2/3)

### ■ strcpy(s1, s2)

- Copy the string pointed to by s2 into array pointed to by s1.
- Function returns s1.
- Example:

```
char name[10];  
strcpy(name, "Matthew");
```

M	a	t	t	h	e	w	\0	?	?
---	---	---	---	---	---	---	----	---	---

- The following assignment statement does not work:

```
name = "Matthew";
```

- What happens when string to be copied is too long?

```
strcpy(name, "A very long name");
```

A		v	e	r	y		l	o	n	g		n	a	m	e	\0
---	--	---	---	---	---	--	---	---	---	---	--	---	---	---	---	----

### ■ strncpy(s1, s2, n)

- Copy first n characters of string pointed to by s2 to s1.

## 4. String Functions (3/3)

- **strstr(s1, s2)**
  - Returns a pointer to the first instance of string s2 in s1.
  - Returns a NULL pointer if s2 is not found in s1,
- We will use the functions above in Demo #7.
- Read up on the above functions (Table 7.3 [pg 405 – 411] and Table 7.4 [pg 412 – 413])
  - We will do some more exercises on them next week
- Other functions (atoi, strcat, strchr, strtok, etc.)
  - We will explore these in your discussion session

## 5. Pointer to String (1/2)

```
#include <stdio.h>
#include <string.h>
int main(void) {
    char name[12] = "Chan Tan";
    char *namePtr = "Chan Tan";

    printf("name = %s\n", name);
    printf("namePtr = %s\n", namePtr);
    printf("Address of 1st array element for name = %p\n", name);
    printf("Address of 1st array element for namePtr = %p\n", namePtr);

    strcpy(name, "Lee Hsu");
    namePtr = "Lee Hsu";

    printf("name = %s\n", name);
    printf("namePtr = %s\n", namePtr);
    printf("Address of 1st array element for name = %p\n", name);
    printf("Address of 1st array element for namePtr = %p\n", namePtr);
}
```

`name` is a character array of 12 elements.

`namePtr` is a pointer to a character.

Both have strings assigned.

**Difference** is `name` sets aside space for 12 characters, but `namePtr` is a char pointer variable that is initialized to point to a string constant of 9 characters.

`name` updated using `strcpy()`.

`namePtr` assigned to another string using `=`.

Address of first array element for `name` remains constant, string assigned to `namePtr` changes on new assignment.

Unit16\_StringPointer.c



## 5. Pointer to String (2/2)

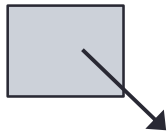
- Comparison

```
char name[12] = "Chan Tan";
```

name[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	h	a	n		T	a	n	\0	\0	\0	\0

```
char *namePtr = "Chan Tan";
```

namePtr



C	h	a	n		T	a	n	\0
---	---	---	---	--	---	---	---	----

## 6. Array of Strings

- Declaration

```
char fruits[MAXNUM][STRSIZE];  
// where MAXNUM is the maximum number of names  
// and STRSIZE is the size of each name
```

- Initialization

```
char fruits[][6] = {"apple", "mango", "pear"};
```

or

```
char fruits[3][6] = {"apple", "mango", "pear"};
```

- Output

```
printf("fruits: %s %s\n", fruits[0], fruits[1]);  
printf("character: %c\n", fruits[2][1]);
```

```
fruits: apple mango  
character: e
```

## 7. Demo #7: Using String Functions

Unit16\_StringFunctions.c

```
#include <stdio.h>
#include <string.h>
#define MAX_LEN 10
int main(void) {
    char s1[MAX_LEN + 1], s2[MAX_LEN + 1], *p;
    int len;

    printf("Enter string (at most %d characters) for s1: ", MAX_LEN);
    fgets(s1, MAX_LEN+1, stdin);
    len = strlen(s1);
    if (s1[len - 1] == '\n') s1[len - 1] = '\0';

    printf("Enter string (at most %d characters) for s2: ", MAX_LEN);
    fgets(s2, MAX_LEN+1, stdin);
    len = strlen(s2);
    if (s2[len - 1] == '\n') s2[len - 1] = '\0';

    printf("strcmp(s1,s2) = %d\n", strcmp(s1,s2));

    p = strstr(s1,s2);
    if (p != NULL) printf("strstr(s1,s2) returns %s\n", p);
    else printf("strstr(s1,s2) returns NULL\n");

    strcpy(s1,s2);
    printf("After strcpy(s1,s2), s1 = %s\n", s1);
    return 0;
}
```

## 8. Strings and Pointers (1/4)

- We discussed in [Unit #9 Section 4](#) that an array name is a pointer (that points to the first array element)
- Likewise, since a string is physically an array of characters, the name of a string is also a pointer (that points to the first character of the string)

Unit16\_String\_vs\_Pointer.c

```
char str[] = "apple";

printf("1st character: %c\n", str[0]);
printf("1st character: %c\n", *str);

printf("5th character: %c\n", str[4]);
printf("5th character: %c\n", *(str+4));
```

1st character: a  
1st character: a  
5th character: e  
5th character: e

## 8. Strings and Pointers (2/4)

- `Unit16_strlen.c` shows how we could compute the length of a string if we are not using `strlen()`
- See full program on CS1010 website

Unit16\_strlen.c

```
int mystrlen(char *p) {  
    int count = 0;  
  
    while (*p != '\0') {  
        count++;  
        p++;  
    }  
  
    return count;  
}
```

## 8. Strings and Pointers (3/4)

- Since ASCII value of null character `'\0'` is zero, the condition in the while loop is equivalent to `(*p != 0)` and that can be further simplified to just `(*p)` (see left box)
- We can combine `*p` with `p++` (see right box) (why?)

```
int mystrlen(char *p) {  
    int count = 0;  
  
    while (*p) {  
        count++;  
        p++;  
    }  
  
    return count;  
}
```

```
int mystrlen(char *p) {  
    int count = 0;  
  
    while (*p++) {  
        count++;  
    }  
  
    return count;  
}
```

Unit16\_strlen\_v2.c

## 8. Strings and Pointers (4/4)

- How to interpret the following?

**while (\*p++)**

Check whether **\*p** is 0  
(that is, whether **\*p** is the  
null character '\0')...

Then, increment **p** by 1  
(so that **p** points to the  
next character).  
Not increment **\*p** by 1!

**(\*p++)** is not the same as **(\*p)++**  
**(\*p)++** is to increment **\*p** (the  
character that **p** points to) by 1. (Hence, if  
**p** is pointing to character 'a', that character  
becomes 'b'.)

# Extra topics

- 2 additional topics that are not in the syllabus are in the file [Unit16\\_Extra.pptx](#)
  - Array of Pointers to Strings
  - Command-line arguments



# Summary

- In this unit, you have learned about
  - **Characters**
    - Declaring and using characters
    - Characters I/O
    - Character functions
  - **Strings**
    - Declaring and initialising strings
    - String I/O
    - String functions
    - Array of strings