CS1010 http://www.comp.nus.edu.sg/~cs1010/

Programming Methodology

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

Reference:

- Chapter 8: Strings

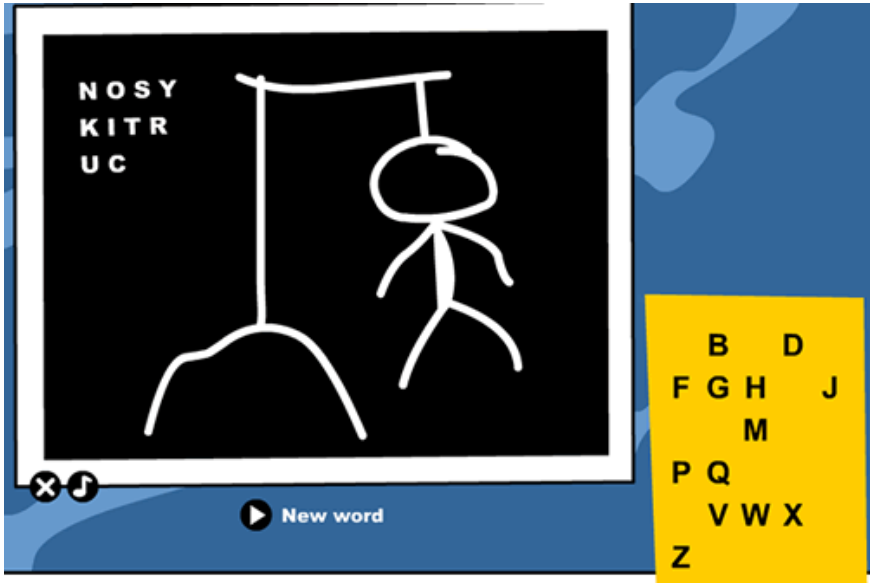
Unit 16: Characters and Strings (1/2)

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
9. String Function – strtok( )
10. Extra topics

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: Practice S02P03 - 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	dc1	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

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

int main(void) {

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

   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);
```

Declaring and initialising char variables.

Using %c

Relationship between character and integer.

```
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;

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

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

   return 0;
}
```

Read a character from stdin.

Print a character to stdout.

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

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: https://www.tutorialspoint.com/c_standard_library/ctype_h.htm)

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

2.5 Ex #1: Summing Digit Characters (1/4)

- Write a program Unit16_SumDigits.c to read characters on a line, and sum the digit characters, ignoring the non-digit ones and everything after the first white space.
- Use the appropriate functions introduced in Demos #2 and #3.
- Two sample runs:

```
Enter input: v7o/K3-968+?.2@+
Sum = 35
```

```
Enter input: ^71()-2%:46" 9W35j
Sum = 20
```

2.5 Ex #1: Summing Digit Characters (2/4)

- Refer to this web page:
- https://www.tutorialspoint.com/c_standard_library/ctype_h.htm
- What is the input function needed if we do not want to use scanf())? getchar()
- What are the character functions needed?

  isdigit()

  isspace()

- What header file to include besides <stdio.h>?

  <ctype.h>

2.5 Ex #1: Summing Digit Characters (3/4)

- How do we obtain an integer value from a digit character (let ch be the character variable)?
- i.e.: '0' → 0. '1' → 1, …, '9' → 9

Hint: ASCII value

- What is the ASCII value of character '0'? 48
- What is the ASCII value of character '1'? 49
- …
- What is the ASCII value of character '9'? 57

ch – 48 *or* ch – '0'

2.5 Ex #1: Summing Digit Characters (4/4)

```c
#include <stdio.h>
#include <ctype.h>

int main(void) {
   char ch;
   int sum = 0;

   printf("Enter input: ");
   while (!isspace(ch = getchar()))
      if (isdigit(ch))
         sum += ch - '0';

   printf("Sum = %d\n", sum);

   return 0;
}
```

2.6 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, F;

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;
```

Quick Quiz

1. Are 'A' and "A" the same thing?

No

2. Can you do this?
   - char ch = 'at';

No

3. Can char be used in a switch statement? How about a string?

char – yes
string – no

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 of 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.

- Initializer for string
  - Two ways:
    - `char fruit_name[] = "apple";`
    - `char fruit_name[] = {'a','p','p','l','e','\0'};`

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

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];

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

Test out the programs with this input:
My book

Output:
str = My

Unit16_StringIO2.c

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

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

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

Output:
str = My book

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?

Unit16_without_null_char.c

```
#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;
}
```

One possible output:

```
Length = 8
str = apple¿ø<
```

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 8.1 (pg 483)
  - http://faculty.edcc.edu/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
- 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 =.

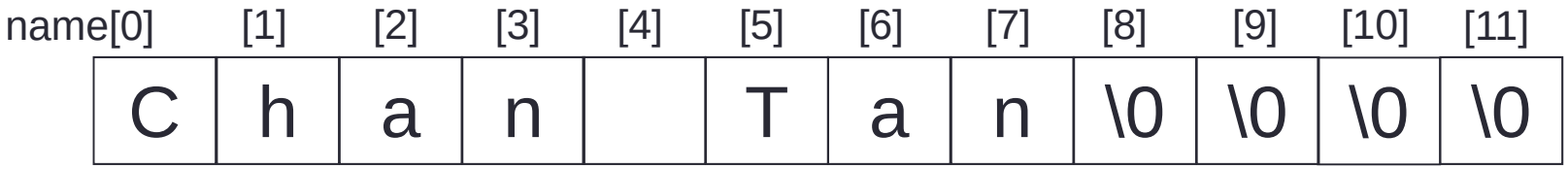
Address of first array element for name remains constant, string assigned to namePtr changes on new assignment.

Unit16_StringPointer.c

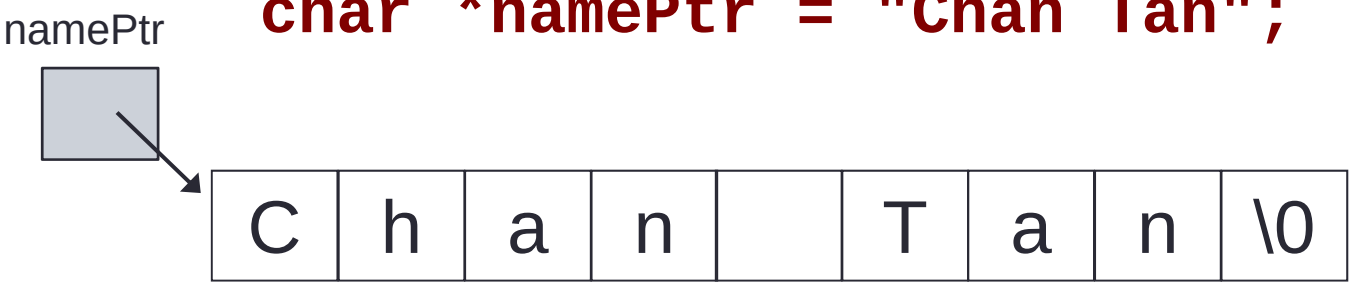
5. Pointer to String (2/2)

- Comparison

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




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



6. Array of Strings

Unit16_ArrayOfStrings.c

- 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 #8 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'.)

9. String function: strtok() (1/2)

- To break a string into a series of tokens using some specified delimiter(s).

```
char *strtok(char *str, const char *delim)
```

- Read the following site:
  - http://www.tutorialspoint.com/c_standard_library/c_function_strtok.htm
- The first time you call strtok() you pass it: (1) the string you want to tokenise, and (2) a delimiter string.
- For subsequent calls, you pass it: (1) NULL as the first paramater to tokenise the same string, and (2) a delimiter string.

9. String function: strtok() (2/2)

Unit16_strtok.c

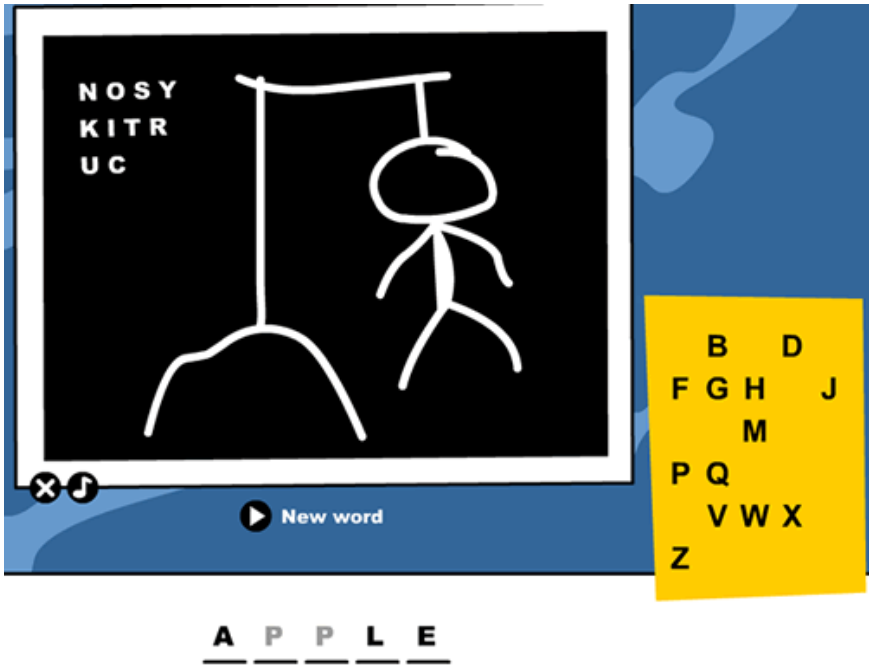
```
#include <string.h>
#include <stdio.h>

int main(void) {
  char str[80] = "This is - www.tutorialspoint.com - website";
  char s[2] = "-";
  char *token;

  /* get the first token */
  token = strtok(str, s);

  /* walk through other tokens */
  while (token != NULL) {
    printf("%s\n", token);
    token = strtok(NULL, s);
  }
  return 0;
}
```

Output:

```
This is
 www.tutorialspoint.com
 website
```

Example: Hangman Game

- http://www.hangman.no/



Let’s play!

Hangman Game version 1 (1/5)

- Unit16_Hangman_v1.c
  - Assume that a player is given 5 lives.
  - Each incorrect guess → reduce the number of lives.
  - Each correct guess → display the letter in the word.

Hangman Game version 1 (2/5)

- Sample run #1:

```
Number of lives: 5
Guess a letter in the word _ _ _ _ _
h
Number of lives: 4
Guess a letter in the word _ _ _ _ _
p
Number of lives: 4
Guess a letter in the word _ p p _ _
b
Number of lives: 3
Guess a letter in the word _ p p _ _
m
Number of lives: 2
Guess a letter in the word _ p p _ _
x
Number of lives: 1
Guess a letter in the word _ p p _ _
i
Sorry, you're hanged! The word is "apple".
```

- Sample run #2:

```
Number of lives: 5
Guess a letter in the word _ _ _ _ _
p
Number of lives: 5
Guess a letter in the word _ p p _ _
e
Number of lives: 5
Guess a letter in the word _ p p _ e
o
Number of lives: 4
Guess a letter in the word _ p p _ e
a
Number of lives: 4
Guess a letter in the word a p p _ e
l
Congratulations! The word is "apple".
```

Hangman Game version 1 (3/5)

Unit16_Hangman_v1.c

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

int has_letter(char [], char);

int main(void) {
   char input;
   char word[] = "apple";
   char temp[] = "_____";
   int i, count = 0;
   int num_lives = 5;
   int length = strlen(word);
```

Hangman Game version 1 (4/5)

Unit16_Hangman_v1.c

```
   do {
      printf("Number of lives: %d\n", num_lives);
      printf("Guess a letter in the word ");
      puts(temp);
      scanf(" %c", &input);

      if (has_letter(word, input)) {
         for (i=0; i<length; i++)
            if ((input == word[i]) && (temp[i] == '_')) {
               temp[i] = input;
               count++;
            }
      }
      else num_lives--;
   } while ((num_lives != 0) && (count != length));

   if (num_lives == 0)
      printf("Sorry, you're hanged! The word is \"%s\"\n", word);
   else
      printf("Congratulations! The word is \"%s\"\n", word);
   return 0;
}
```

Hangman Game version 1 (5/5)

Unit16_Hangman_v1.c

```
// Check whether word contains ch
int has_letter(char word[], char ch) {
   int j;
   int length = strlen(word);

   for (j=0; j<length; j++) {
      if (ch == word[j])
         return 1;
   }

   return 0;   // ch does not occur in word
}
```

Note: It is better to call strlen(word) just once and save the length in a variable, instead of calling strlen(word) multiple times as a condition in the 'for' loop.

10. Extra topics

- 2 additional topics that are not in the syllabus:
  - Array of Pointers to Strings
  - Command-line arguments

1. Array of Pointers to Strings (1/2)

- Declaration

```
char *fruits[3];
```

- Assignment

```
fruits[0] = "apple";
fruits[1] = "banana";
fruits[2] = "cherry";
```

- Declare and initialize

```
char *fruits[] = {"apple", "banana", "cherry"};
fruits[0] = "pear";     // new assignment
```

- Output

```
for (i=0; i<3; i++)
    printf("%s\n", fruits[i]);
```

```
pear
banana
cherry
```

1. Array of Pointers to Strings (2/2)

Unit16_ArrayOfPointersToStrings.c

```
#include <stdio.h>

int main(void) {
  char *fruits[] = {"apple", "banana", "cherry"
};
  int i;

  fruits[0] = "pear";
  for (i=0; i<3; i++) {
    printf("%s\n", fruits[i]);
  }

  return 0;
}
```

2. Command-line Arguments (1/2)

- So far, our main function header looks like this:

```
int main(void)
```

- We can pass arguments to a program when we run it:

```
a.out water "ice cream" 34+7
```

- Add two parameters in the main function header:

```
int main(int argc, char *argv[])
```

  - Parameter argc stands for “argument count”
  - Parameter argv stands for “argument vector”. It is an array of pointers to strings.
  - argv[0] is the name of the executable file (sometimes also called the command)
  - You can name them anything, but the names argc and argv are commonly used.

2. Command-line Arguments (2/2)

Unit16_CommandLineArgs.c

```
#include <stdio.h>
int main(int argc, char *argv[]) {
   int count;

   printf ("This program was called with \"%s\"\n", argv[0]);
   if (argc > 1)
      for (count = 1; count < argc; count++)
         printf("argv[%d] = %s\n", count, argv[count]);
   else
      printf("The command had no argument.\n");

   return 0;
}
```

```
gcc Unit16_CommandLineArgs.c
a.out water "ice cream" 34+7
This program was called with "a.out"
argv[1] = water
argv[2] = ice cream
argv[3] = 34+7
```

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

End of File