Lecture 3

Comments. Characters. Strings. Names (Identifies) in C/C++

Comments.

A comment is a piece of descriptive text which explains some aspect of a program. Program comments are totally ignored by the compiler and are only intended for human readers. C/C++ provides two types of comment delimiters:

- Anything after // (until the end of the line on which it appears) is considered a comment.
- Anything enclosed by the pair /* and */ is considered a comment.

Listing illustrates the use of both forms.

Listing

```
#include <stdio.h>

/* This program calculates the weekly gross pay for a worker,
   based on the total number of hours worked and the hourly pay
   rate. */
 int main (     )
{
     int      workDays = 5 ;              // Number of work days per week
     float    workHours = 7.5;            // Number of work hours per day
     float    payRate = 33.50;            // Hourly pay rate
     float    weeklyPay;                  // Gross weekly pay
     weeklyPay = workDays * workHours * payRate;
     printf("weeklyPay =  %.2f " , weeklyPay) ;
     return 0;
}
```

Comments should be used to enhance (not to hinder) the readability of a program. The following two points, in particular, should be noted:

- A comment should be easier to read and understand than the code which it tries to explain. A confusing or unnecessarily-complex comment is worse than no comment at all.
- Over-use of comments can lead to even less readability. A program which contains so much comment that you can hardly see the code can by no means be considered readable.
- Use of descriptive names for variables and other entities in a program, and proper indentation of the code can reduce the need for using comments.

The best guideline for how to use comments is to simply apply common sense.

Characters.

A **character variable** is defined to be of type char. A character variable occupies a single byte which contains the *code* for the character. This code is a numeric value and depends on the *character coding system* being used (is machine-dependent). The most common system is ASCII (American Standard Code for Information Interchange). For example, the character *A* has the ASCII code 65, and the character *a* has the ASCII code 97.

```
char        ch = 'A';
```

A **literal character** is written by enclosing the character between a pair of single quotes (e.g., 'A'). Nonprintable characters are represented using escape sequences. For example:

```
’\a’        // Alarm (computer bell)
'\n'        // new line
'\r'        // carriage return
'\t'        // horizontal tab
'\v'        // vertical tab
'\b'        // backspace
'\f'        // formfeed
```

Single and double quotes and the backslash character can also use the escape notation:

```
'\’'        // single quote (’')
'\”'        // double quote (")
'\\'        // backslash (\)
’\?’        //  question mark(?)
```

Literal characters may also be specified using their numeric code value. Two general escape sequences \ooo (a backslash followed by up to three octal digits) or \xhh (a backslash followed by two hexadecimal digits) used for this purpose. For example (assuming ASCII):

```
'\12'       // newline (decimal code = 10)
'\11'       // horizontal tab (decimal code = 9)
‘\101’      //  ‘A‘ (decimal code = 65)
'\0'        // null byte, null character (decimal code = 0
```

Strings

In C/C++ a **string** is a set (sequence) of characters which are terminated by a null character. A **literal string** is written by enclosing its characters between a pair of double quotes (e.g., “HELLO”). The compiler always appends a null character to a literal string to mark its end. The characters of a string may be specified using any of the notations for specifying literal characters. For example:

```
"Name\tAddress\tTelephone"    // tab-separated words
"ASCII character 65: \101"    // 'A' specified as '101'
```

In program a long string may extend beyond a single line, in which case each of the preceding lines should be terminated by a backslash. For example:

```
"Example to show \
the use of backslash for \
writing a long string"
```

The backslash in this context means that the rest of the string is continued on the next line.
The above string is equivalent to the single line string:

"Example to show the use of backslash for writing a long string"

A common programming error results from confusing a single-character string (e.g., "A") with a single character (e.g., 'A'). These two are *not* equivalent. The first consists of two bytes (the character 'A' followed by the character '\0'), whereas the latter consists of a single byte.

The shortest possible string is the null string ("") which simply consists of the null character.

Names (Identifies). Keywords.

Programming languages use names to refer to the various entities that make up a program. We have already seen examples of an important category of such names (variable names). Other categories include: function names, type names, and macro names, which will be described later.

C/C++ languages have the following rules for creating valid names (also called **identifiers**). A name should consist of one or more characters, each of which may be a letter (i.e., 'A'-'Z' and 'a'-'z'), a digit (i.e., '0'-'9'), or an underscore character ('_'), except that the first character may not be a digit. Upper and lower case letters are distinct. For example:

```
salary      // valid identifier
salary2     // valid identifier
2salary     // invalid identifier (begins with a digit)
_salary     // valid identifier
Salary      // valid but distinct from salary
```

Certain words are reserved by C++ for specific purposes and may not be used as identifiers. These are called **reserved words** or **keywords** and are summarized in the table:

C/C++ keywords.

asm	continue	float	new	signed	try
auto	default	for	operator	sizeof	typedef
break	delete	friend	private	static	union
case	do	goto	protected	struct	unsigned
catch	double	if	public	switch	virtual
char	else	inline	register	template	void
class	enum	int	return	this	volatile
const	extern	long	short	throw	while

Exercises

1 Write a program which inputs a temperature reading expressed in Fahrenheit and outputs its equivalent in Celsius, using the formula:

$$^{\circ}C = \frac{5}{9}(^{\circ}F - 32)$$

Compile and run the program. Its behavior should resemble this:

```
Enter temperature in Fahrenheit: 41

        Result:
```

```
41 degrees Fahrenheit = 5 degrees Celsius
```

2 Which of the following represent valid variable definitions?

```
int n = -100;
unsigned int i = -100;
signed int = 2.9;
long m = 2, p = 4;
int 2k;
double x = 2 * m;
float y = y * 2;
unsigned double z = 0.0;
double d = 0.67F;
float f = 0.52L;
signed char = -1786;
char c = '$' + 2;
sign char h = '\111' ;
```

3 Which of the following represent valid identifiers?

```
identifier
seven 11
_unique__
gross-income
gross$income
2by2
default
average_weight_of_a_large_pizza
variable
object.oriented
```

4 Define data type of variables to represent the following entities:

- Age of a person.
- Income of an employee.
- Number of words in a dictionary.
- A letter of the alphabet.
- A greeting message.

Lecturer: Kulev Mihail, associate professor