Lecture 2

WRITING A PROGRAM IN C+ +

1- Character set:

C++ has the letters and digits, as show below:

Uppercase: A, B, C, . . ., Z

Lowercase: a, b, c, . . ., z

Digits: 0, 1, 2, . . .,9

Special Characters: All characters other than listed treated as special characters for example:

+	-	*	/	^
(	[	{	}	]
)	<	=	>	, (Comma)
(Double Conations)	. (Dot)	: (Colon)	; (Semicolon)	␣ (Blank Space)

In C++ language, upper case and lower case letters are distinct and hence there are 52 letters in all. For example **bag** is different from **Bag** which is different from **BAG**.

2- Identifiers:

An **identifier** is a name given to some program entity, such as variable, constant, array, function, structure, or class. An identifier is a sequence of alphanumeric (alphabetic and numeric) characters, the first of which must be a letter, and can't contain spaces. The length of an identifier is machine dependent. C++ allows identifiers of up to **127 characters**.

A **variable** should not begin with a digit. C++ does not set a maximum length for an identifier. Some examples of valid identifiers are as follows:

My_name (7 char.)

i (1 char.)

B (1 char.)

- Some examples of invalid identifiers are:

3ab - variable begins with a digit

a( )test - use of special characters

gross salary - space between characters of single variable

3 Keywords:

The keywords are also identifiers but cannot be user defined, since they are reserved words. All the keywords should be in lower case letters. Reserved words cannot be used as variable names or constant. The following words are reserved for use as keywords:

Some of C++ Language Reserved Words:				
break	case	char	cin	cout
delete	double	else	enum	false
float	for	goto	if	int
long	main	private	public	short
sizeof	switch	true	union	void

Example: determine the types of the words in C++ language:

Float, float, 1float, test23, hit!it, Beta, βeta, 5q,

Ans:

Word	Float	float	1float	Test32	Hit!it	Beta	βeta	5q
Type	identify	keyword	wrong	identify	wrong	identify	wrong	wrong

4) Constants

- There are three types of constants: ***string constants***, ***numeric constants*** and ***character constants***.

a) String Constants

- A String constant is a sequence of ***alphanumeric characters*** enclosed in ***double quotation*** marks .whose maximum length is, 255 characters.
- Following are the examples of valid, string constants.

1) "The result ="

2) "Rs 2000.00”

3) "This is test program by Ali"

- Following are some examples for invalid string constants.

1) "My name is %$%' - Closing double quotation mark is missing.

2) 'this is test' - Characters are not enclosed in double quotation marks.

b) Numeric Constants

- Numeric constants are positive or negative numbers. There are four types of numeric constants: ***integer constant***, ***floating point constant***, ***hex constant*** and ***octal constant.***

C) Character Constants: A character represented within single quotes denotes a character constant, for example ‘A’, ‘a’, ‘:’, ‘?’, etc… Its maximum size is 8 bit long, signed, and unsigned char are three distinct types.

Char x; char x,y,z;

The backslash (\) is used to denote non graphic characters and other special characters for a specific operations such as:

Special Escape Code:	
Escape Code	**Description**
\n	New line. Position the screen cursor to the beginning of the next line.
\t	Horizontal TAB (six spaces). Move the screen cursor to the next tab stop.
\r	Carriage return. Position the cursor to the beginning of the current line, do not advance to the next line.
\a	Alert. Produces the sound of the system bell.
\b	Back space
****	Backslash. Prints a backslash character.
\f	Form feed
\v	Vertical tab
\"	Double quote. Prints a (") character.
\o	Null character
\?	question mark
\ooo	Octal value
\xhhh	Hexadecimal value

5-C++ operators

Once we know of the existence of variables and constants, we can begin to operate with them. For that purpose, C++ integrates operators. Unlike other languages whose operators are mainly keywords, operators in C++ are mostly made of signs that are not part of the alphabet but are available in all keyboards. This makes C++ code shorter and more international, since it relies less on English words, but requires a little of learning effort in the beginning. You do not have to memorize all the content of this page. Most details are only provided to serve as a later reference in case you need it.

Assignment (=)

The assignment operator assigns a value to a variable.

a = 5;

This statement assigns the integer value 5 to the variable a. The part at the left of the assignment operator (=) is known as the *lvalue* (left value) and the right one as the *rvalue* (right value). The lvalue has to be a variable whereas the rvalue can be either a constant, a variable, the result of an operation or any combination of these.

The most important rule when assigning is the *right-to-left* rule: The assignment operation always takes place from right to left, and never the other way:

a = b;

This statement assigns to variable a (the lvalue) the value contained in variable b (the rvalue). The value that was stored until this moment in a is not considered at all in this operation, and in fact that value is lost.

Arithmetic operators (+, -, *, /, %)

The five arithmetical operations supported by the C++ language are:

+ addition

- subtraction

* multiplication

/ division

% modulo

Operations of addition, subtraction, multiplication and division literally correspond with their respective mathematical operators. The only one that you might not be so used to see is *modulo*; whose operator is the percentage sign (%). Modulo is the operation that gives the remainder of a division of two values. For example, if we write:

a = 11 % 3;

the variable a will contain the value 2, since 2 is the remainder from dividing 11 between 3.

Compound assignment

When we want to modify the value of a variable by performing an operation on the value currently stored in that variable we can use compound assignment operators:

expression is equivalent to

value += increase; value = value + increase;

a -= 5; a = a - 5;

a /= b; a = a / b;

price *= units + 1; price = price * (units + 1);

Increase and decrease (++, --)

Shortening even more some expressions, the increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to -=1, respectively. Thus:

c++;

c+=1;

c=c+1;

are all equivalent in its functionality: the three of them increase by one the value of c.

In the early C compilers, the three previous expressions probably produced different executable code depending on which one was used. Nowadays, this type of code optimization is generally done automatically by the compiler, thus the three expressions should produce exactly the same executable code. A characteristic of this operator is that it can be used both as a prefix and as a suffix. That means that it can be written either before the variable identifier (++a) or after it (a++). Although in simple expressions like a++ or ++a both have exactly the same meaning, in other expressions in which the result of the increase or decrease operation is evaluated as a value in an outer expression they may have an important difference in their meaning: In the case that the increase operator is used as a prefix (++a) the value is

increased before the result of the expression is evaluated and therefore the increased value is considered in the outer expression; in case that it is used as a suffix (a++) the value stored in a is increased after being evaluated and therefore the value stored before the increase operation is evaluated in the outer expression. Notice the difference:

Example 1

```
B=3;
A=++B;
// A contains 4, B contains 4
```

Example 2

```
 B=3;
A=B++;
// A contains 3, B contains 4
```

In Example 1, B is increased before its value is copied to A. While in Example 2, the value of B is copied to A and then B is increased.

Relational and equality operators (==, !=, >, <, >=, <=)

In order to evaluate a comparison between two expressions we can use the relational and equality operators. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result.

We may want to compare two expressions, for example, to know if they are equal or if one is greater than the other is. Here is a list of the relational and equality operators that can be used in C++:

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Here there are some examples:

```
(7 == 5)     // evaluates to false.
(5 > 4)      // evaluates to true.
(3 != 2)     // evaluates to true.
(6 >= 6)     // evaluates to true.
(5 < 5)      // evaluates to false.
```

Logical operators (!, &&, ||)

The Operator ! is the C++ operator to perform the Boolean operation NOT, it has only one operand, located at its right, and the only thing that it does is to inverse the value of it, producing false if its operand is true and true if its operand is false. Basically, it returns the opposite Boolean value of evaluating its operand. For example:

!(5 == 5) // evaluates to false because the expression at its right (5 == 5) is true.

!(6 <= 4) // evaluates to true because (6 <= 4) would be false.

!true // evaluates to false

!false // evaluates to true.

The logical operators && and || are used when evaluating two expressions to obtain a single relational result. The operator && corresponds with Boolean logical operation AND. This operation results true if both its two operands are true, and false otherwise. The following panel shows the result of operator && evaluating the expression a && b:

&& OPERATOR

A	**b**	**a && b**
true	true	true
true	false	false
false	true	false
false	false	false

The operator || corresponds with Boolean logical operation OR. This operation results true if either one of its two operands is true, thus being false only when both operands are false themselves. Here are the possible results of a || b:

A	**b**	**a || b**
True	true	true
True	false	true

False true true

false false false

For example:

((5 == 5) && (3 > 6)) // evaluates to false (true && false).

((5 == 5) || (3 > 6)) // evaluates to true (true || false).

Conditional operator (?)

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false. Its format is: condition ? result1 : result2 If condition is true the expression will return result1, if it is not it will return result2.

7==5 ? 4 : 3 // returns 3, since 7 is not equal to 5.

7==5+2 ? 4 : 3 // returns 4, since 7 is equal to 5+2.

5>3 ? a : b // returns the value of a, since 5 is greater than 3.

a>b ? a : b // returns whichever is greater, a or b.

Comma operator (,)

The comma operator (,) is used to separate two or more expressions that are included where only one expression is expected. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

For example, the following code:

a = (b=3, b+2);

6-**C++ Data Types**

C++ offer the programmer a rich assortment of built-in as well as user defined data types. Following table lists down seven basic C++ data types:

Type	**Typical Bit Width**
char	1byte

unsigned char	1byte
signed char	1byte
int	2bytes
unsigned int	2bytes
signed int	2bytes
short int	2bytes
unsigned short int	2bytes
signed short int	2bytes
long int	4bytes
signed long int	4bytes
unsigned long int	4bytes
float	4bytes
double	8bytes
long double	16bytes