



CH3: ALGORITHMS, FLOWCHART AND OPERATORS

Ms. SAFA ANMAR ALBARWARY

Computer Programming And Algorithm

1st Semester

Date: 11/ 01/ 2024

Outline

- **Algorithms**
- **Flowchart**
- **Operators**

Objectives

- The students will learn how to:
 1. Track the process of the program from the starting point till the end using Algorithms and flowcharts.
 2. How to use different types of operators with C++ Language.

Algorithm and Flow Chart

- Algorithm and flowchart are two types of tools to explain the process of a program.
- This class extends the differences between an algorithm and a flowchart, and how to create a flowchart to explain an algorithm in a visual way.
 - An algorithm is a step-by- step analysis of the process, while a flowchart explains the steps of a program in a graphical .
- To write a logical step-by-step method to solve the problem is called **algorithm**, in other words, an algorithm is a procedure for solving problems. In order to solve a mathematical or computer problem, this is the first step of the procedure.
- An algorithm includes calculations, reasoning and data processing. Algorithms can be presented by natural languages, flowcharts

THE ALGORITHM,

- The ordered set of instructions required to solve a problem is known as an *algorithm*.
- Algorithm is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. No matter what the input values may be, an algorithm terminates after executing a finite number of instructions.
- The characteristics of a good algorithm are:
 1. **Precision** – the steps are precisely stated (defined).
 2. **Uniqueness** – results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
 3. **Finiteness** – the algorithm stops after a finite number of instructions are executed.
 4. **Input** – the algorithm receives input.
 5. **Output** – the algorithm produces output.
 6. **Generality** – the algorithm applies to a set of inputs.

Example, Write a algorithm to find out number is odd or even?

Ans.

step 1 : start

step 2 : input number

step 3 : Num=number / 2

step 4 : if Num=0 then

 print "number even"

 else








 print "number odd"

step 5 : stop

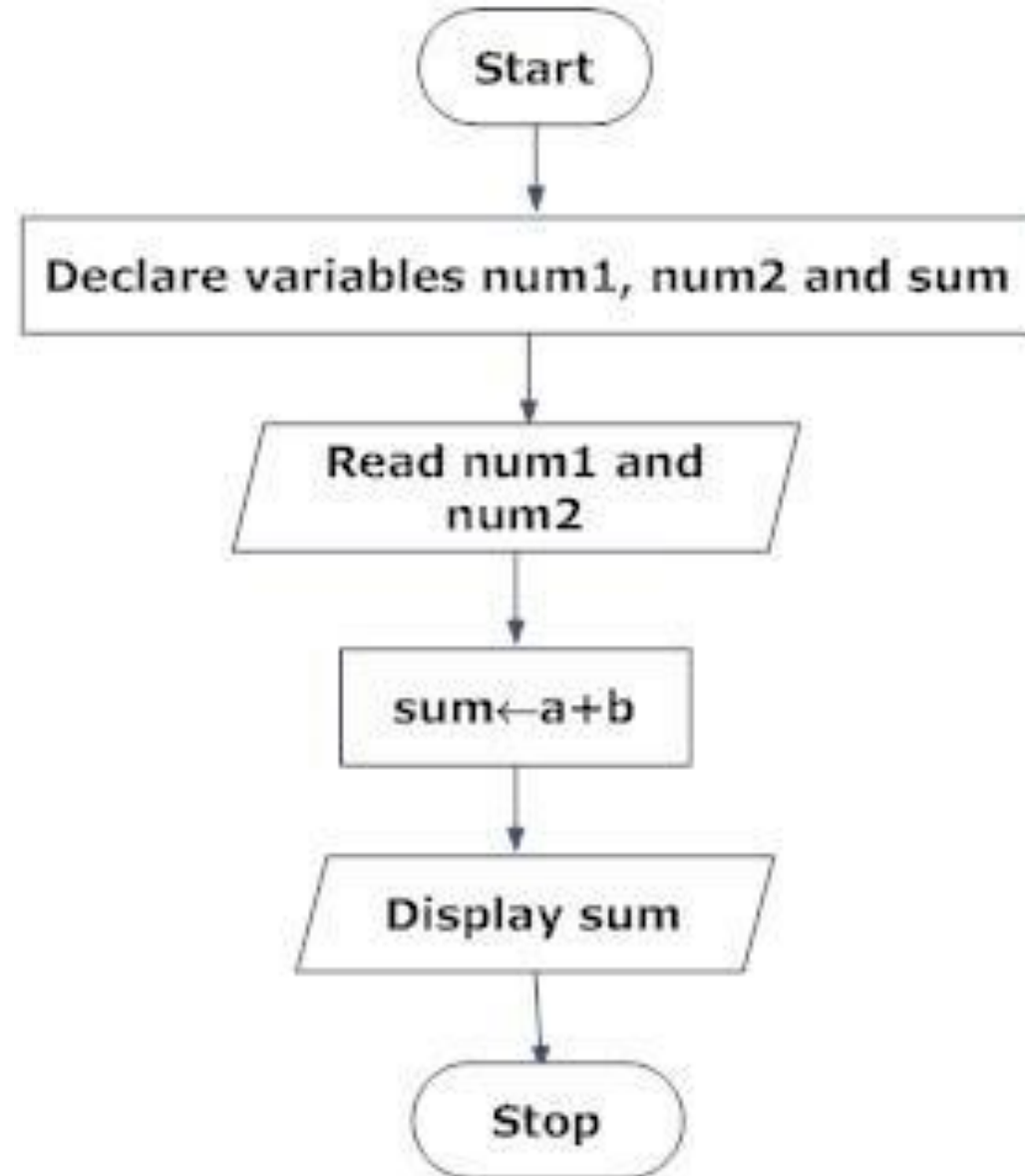
THE FLOWCHART,

- **Flowchart**, is a diagrammatic representation of an algorithm. A graphical representation of the sequence of operations in an information system or program.
- Flowchart is very helpful in writing program and explaining program to others. Information system flowcharts show how data flows from source documents through the computer to final distribution to users.
- Program flowcharts show the sequence of instructions in a single program or subroutine.
- **Symbols Used In Flowchart**, Different symbols are used for different states in flowchart,
 - For example: Input / Output and decision making has different symbols.

The table below describes all the symbols that are used in making flowchart:

Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Used to represent start and end of flowchart.
	Input/Output	Used for input and output operation.
	Processing	Used for airthmetic operations and data-manipulations.
	Desicion	Used to represent the operation in which there are two alternatives, true and false.
	On-page Connector	Used to join different flowline
	Off-page Connector	Used to connect flowchart portion on different page.

Examples, Draw a flowchart to add two numbers entered by user.



OPERATORS AND EXPRESSIONS

- An operator, in computer programming, is a symbol that usually represents an action or process. These symbols were adapted from mathematics and logic. An operator is capable of manipulating a certain value or operand.
- C++ language offers many types of operators, including:
 1. Arithmetic Operators
 2. Assignment Operators
 3. Relational Operators
 4. Logical Operators

Arithmetic Operators,

CONCEPT: There are many operators for manipulating numeric values and performing arithmetic operations.

C++ offers a multitude of operators for manipulating data. Generally, there are three types of operators: *unary*, *binary*, and *ternary*. These terms reflect the number of operands an operator requires.

1. **Unary operators**, only require a single operand.
2. **Binary operators**, work with two operands. The assignment operator is in this category.
3. **Ternary operators**, as you may have guessed, require three operands.

- Arithmetic operations are very common in programming. The table below shows the common arithmetic operators in C++ :

Table : Fundamental Arithmetic Operators

Operator	Meaning	Type	Example
+	Addition	Binary	<code>total = cost + tax;</code>
-	Subtraction	Binary	<code>cost = total - tax;</code>
*	Multiplication	Binary	<code>tax = cost * rate;</code>
/	Division	Binary	<code>salePrice = original / 2;</code>
%	Modulus	Binary	<code>remainder = value % 3;</code>

- **Increment and Decrement Operators:** C++ also provides increment and decrement operators: **++** and **--** respectively.
 - ✓ ++ increases the value of the operand by 1
 - ✓ -- decreases it by 1

C++ Assignment Operators,

CONCEPT: In C++, assignment operators are used to assign values to variables. For example,

// assign 15 to a

a = 15;

Here, we have assigned a value of 15 to the variable a in different ways:

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Relational Operators,

CONCEPT: Relational operators allow you to compare numeric and char values and determine whether one is greater than, less than, equal to, or not equal to another.

- So far, the programs you have written follow this simple scheme:
 - Gather input from the user.
 - Perform one or more calculations.
 - Display the results on the screen.
- Computers are good at performing calculations, but they are also quite adept at comparing values to determine if one is greater than, less than, or equal to the other. These types of operations are valuable for tasks such as examining sales figures, determining profit and loss, checking a number to ensure it is within an acceptable range, and validating the input given by a user.

Relational Operators

CONCEPT: A relational operator is used to check the relationship between two operands.

- For example, checks if a is greater than b $a > b;$ Here, $>$ is a relational operator. It checks if a is greater than b or not.
- If the relation is **true**, It returns **1** whereas if the relation is **false**, it returns **0**. Numeric data is compared in C++ by using **relational operators**. Each relational operator determines whether a specific relationship exists between two values.
- In the shown table below, lists all of C++'s relational operators.

Relational Operators	Meaning
$>$	Greater than
$<$	Less than
$>=$	Greater than or equal to
$<=$	Less than or equal to
$==$	Equal to
$!=$	Not equal to

Note: Relational operators are used in decision-making and loops.

Relational Operators

- All of the relational operators are binary, which means they use two operands. Here is an example of an expression using the greater-than operator: $x > y$

- This expression is called a *relational expression*. It is used to determine whether x is greater than y .

- The following expression determines whether x is less than y : $x < y$

- The table shows examples of several relational expressions that compare the variables x and y :

Expression	What the Expression Means
$x > y$	Is x greater than y ?
$x < y$	Is x less than y ?
$x >= y$	Is x greater than or equal to y ?
$x <= y$	Is x less than or equal to y ?
$x == y$	Is x equal to y ?
$x != y$	Is x equal to y ?

NOTE: All the relational operators have left-to-right associativity. Recall that associativity is the order in which an operator works with its operands.

Class Activity,

A couple of the relational operators actually test for two relationships. The \geq operator determines whether the operand on its left is greater than or equal to the operand on the right. Assuming that *a* is 4, *b* is 6, and *c* is 4. State the following expression if its *True* or *False*.

$b \geq a$

$a \geq c$

$a \geq 5$

$a \leq c$

$b \leq 10$

$b \leq a$

$a \neq b$

$b \neq c$

$a \neq c$

NOTE: In C++, relational expressions represent true states with the number 1 and false states with the number 0.

Precedence of Relational Operators,

Table Precedence of Relational Operators
(Highest to Lowest)

>	>=	<	<=
==	!=		

- **Example:** If $a = 9$, $b = 24$, and $c = 0$, the following statement would cause a 1 to be printed:

```
cout << (c == a > b);
```

- Because of the relative precedence of the operators in this expression, $a > b$ would be evaluated first. Since 9 is not greater than 24, it would evaluate to false, or 0. Then $c == 0$ would be evaluated. Since c does equal 0, this would evaluate to true, or 1. So a 1 would be inserted into the output stream and printed.

WARNING NOTE!

- ✓ Notice the equality operator is two = symbols together. Don't confuse this operator with the assignment operator, which is one = symbol.
- ✓ The == operator determines whether a variable is equal to another value, but the = operator assigns the value on the operator's right to the variable on its left.

Example,

Referring to the expressions shown in the table below, Assume x is 10 and y is 7. For each expression state if its True or False.

Expression	Value
$x < y$	False, because x is not less than y .
$x > y$	True, because x is greater than y .
$x \geq y$	True, because x is greater than or equal to y .
$x \leq y$	False, because x is not less than or equal to y .
$y \neq x$	True, because y is not equal to x .

Logical Operators

CONCEPT: Logical operators connect two or more relational expressions into one or reverse the logic of an expression. Logical operators are used to check whether an expression is **true** or **false**. If the expression is **true**, it returns **1** whereas if the expression is **false**, it returns **0**.

Operator	Meaning	Effect
&&	AND	Connects two expressions into one. Both expressions must be true for the overall expression to be true.
	OR	Connects two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which.
!	NOT	The ! operator reverses the "truth" of an expression. It makes a true expression false, and a false expression true.

○ The && Operator,

Expression	Value of Expression
<code>true && false</code>	false (0)
<code>false && true</code>	false (0)
<code>false && false</code>	false (0)
<code>true && true</code>	true (1)

NOTE: If the sub-expression on the left side of an && operator is false, the expression on the right side will not be checked. Since the entire expression is false if only one of the sub-expressions is false, it would waste CPU time to check the remaining expression. This is called *Short Circuit Evaluation*.

○ The || Operator,

Expression	Value of the Expression
<code>true false</code>	true (1)
<code>false true</code>	true (1)
<code>false false</code>	false (0)
<code>true true</code>	true (1)



NOTE: The || operator also performs short circuit evaluation. If the sub-expression on the left side of an || operator is true, the expression on the right side will not be checked. Since it's only necessary for one of the sub-expressions to be true, it would waste CPU time to check the remaining expression.

○ The ! Operator,

Expression	Value of the Expression
<code>!true</code>	false (0)
<code>!false</code>	true (1)

Precedence and Associativity of Logical Operators

○ Logical Operators in Order of Precedence:

Table

Logical Operators in Order of Precedence
!
&&

- The **!** operator has a higher precedence than many of the C++ operators. To avoid an error, you should always enclose its operand in parentheses unless you intend to apply it to a variable or a simple expression with no other operators.
 - ✓ For example, consider the following expressions: $!(x > 2)$ $!x > 2$
- The **&&** and **||** operators rank lower in precedence than the relational operators, so precedence problems are less likely to occur. If you feel unsure, however, it doesn't hurt to use parentheses anyway. For example,
 - ✓ $a > b \ \&\& \ x < y$ is the same as $(a > b) \ \&\& \ (x < y)$
 - ✓ $a > b \ || \ x < y$ is the same as $(a > b) \ || \ (x < y)$

Class Activity,

If $a = 2$, $b = 4$, and $c = 6$, indicate whether each of the following conditions is true or false:

1. $(a == 4) \parallel (b > 2)$
2. $(6 \leq c) \&\& (a > 3)$
3. $(1 \neq b) \&\& (c \neq 3)$
4. $(a \geq -1) \parallel (a \leq b)$
5. $!(a > 2)$
6. $(b > a) \parallel (b > c) \&\& (c == 5)$

Arithmetic operator example,

Write a program to enter two numbers and find:

- A. Their summation , subtraction, multiplication, division and modulus result.**
- B. Their increment and decrement**
- C. Repeat (A) using the Assignment operators**

Example, write a program to find the result for the following expressions by supposing
 $a = 2$, $b = 4$, $c = 6$. Then,

$(a == 4) \parallel (b > 2)$

$(6 \leq c) \&\& (a > 3)$

$(1 \neq b) \&\& (c \neq 3)$

$(a \geq -1) \parallel (a \leq b)$

$!(a > 2)$

$(b > a) \parallel (b > c) \&\& (c == 5)$

Example, write a program to find the circumference of a circle. Use the formula: $C=2\pi r$, where π is approximately equivalent 3.1416. by first writing the algorithm of the program and drawing the flowchart.

Homework, Write a program that converts the dollar to Iraqi dinar. Assume that the present exchange rate is 130,000 against the 100 dollar. Then display the dinar equivalent exchange rate. By first writing the algorithm of the program and drawing the flowchart.



Next Lecture

- making decisions (control structures)
- Flow of Control (IF, ELSE IF, ELSE)

References

- Tony Gaddis, Starting Out with C++ from Control Structures to Objects, 8th Edition (Main).