CS4002: Applied Programming

Fall 2022

Dr. Usman Habib
Associate Professor
AI & DS Department
FAST School of Computing
National University of Computers and Emerging Sciences.

Computer Programming

Computer Programming

- Computer programming is the process of writing ***instructions*** that direct a computer to carry out specific tasks

- A ***computer program*** is a set of ***step-by-step instructions*** that tell a computer how to solve a problem or carry out a task
  - The instructions that make up a computer program are often referred to as ***code***
  - A program is written in a computer ***programming language***

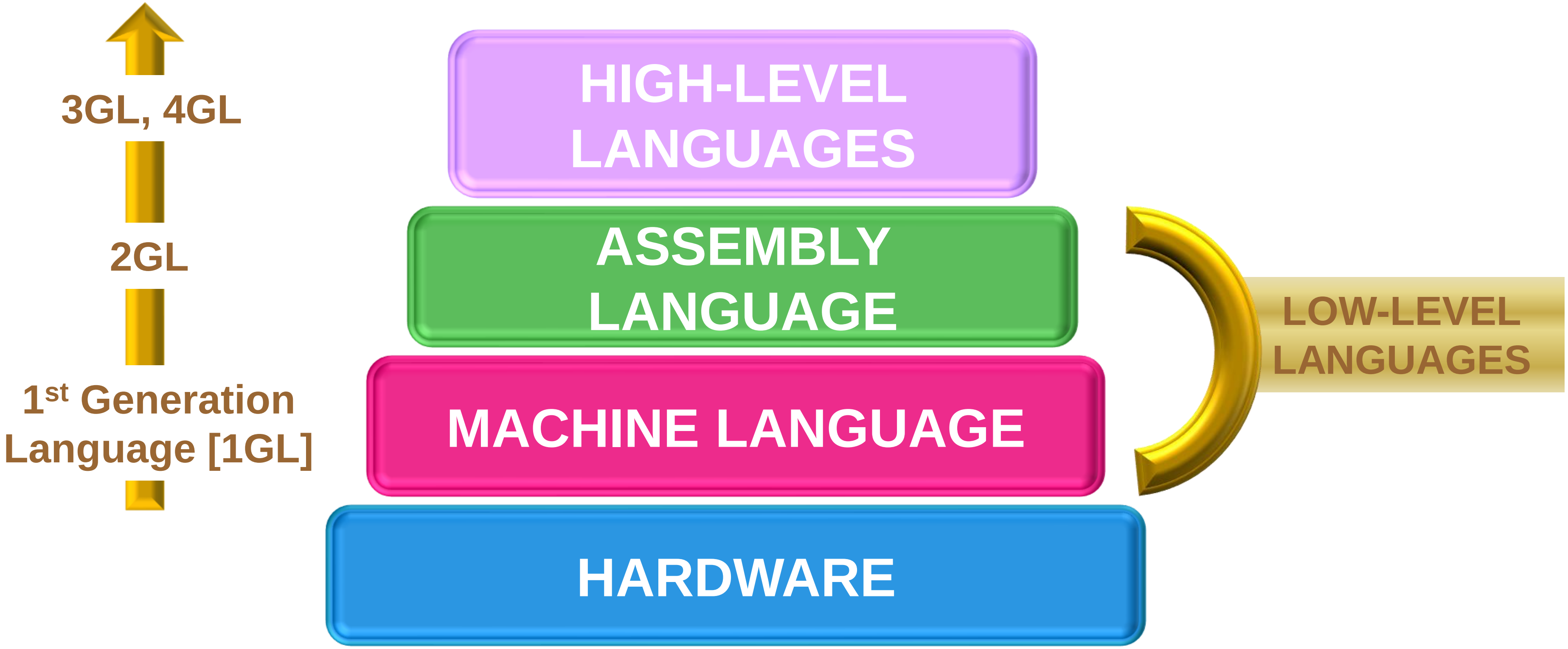
Programming Languages (-1-)

- A ***programming language*** is an *artificial language* designed for creating instructions that a computer can carry out
  - In contrast, the languages we speak [e.g., English / Urdu] are ***natural languages***
- Programming languages differ from ***natural languages*** in that
  - natural languages are used for interaction between people
  - programming languages allow humans to communicate instructions to machines as well as machine-to-machine interaction

Programming Languages (-2-)

- Programming languages can be divided into two major categories
  - Low-level languages
    - Require the programmer to write instructions for the lowest level of computer’s hardware
      - Easy for computer to understand; Difficult for humans
    - Examples: *Machine Language, Assembly Language*
  - High-level languages
    - Makes programming process easier by providing commands such as PRINT or WRITE instead of unintelligible strings of 1s and 0s
    - Examples: *FORTRAN, C, C++, Java, Python …*
- Languages can also be categorized by generations

Programming Language Categorization



Machine Language / Machine Code

- The first languages for programming computers – *sometimes referred to as **first-generation languages***
  - A machine language consists of a set of commands, represented as a ***series of 1s and 0s***, corresponding to the ***instruction set*** understood by a microprocessor
  - A machine language is specific to a particular CPU or microprocessor family
- High-level languages are (mostly) ***translated*** [*compiled*] to machine language in order to be understood and executed by the microprocessor

Machine Language / Machine Code

Example

Add the registers 1 and 2. Place the result in register 6.

[op | rs | rt | rd | shamt | funct]

0 1 2 6 0 32 **decimal**

000000 00001 00010 00110 00000 100000 **binary**

Assembly Language

- Allows programmers to use abbreviated command words rather than 1s and 0s used in machine languages
  - A significant improvement over machine languages
    - Mnemonics such as ADD, SUB, MUL, DIV, JMP etc are more understandable than 0001, 0100 etc
  - Also referred to as ***second-generation languages***
  - Assembly languages are also machine specific
    - Each assembly language command corresponds on a one-to-one basis to a machine language instruction

Assembly Language

Example 1

Add 10 to the variable MARKS

```
ADD MARKS, 10
```

Example 2

Transfer the value 10 to the AL register

```
MOV AL, 10
```

High-level languages

C/C++, JAVA, BASIC and etc.

- Similar to everyday English,
- Use mathematical notations

Example 1

Add **10** to the variable **MARKS**

```
MARKS = MARKS + 10;
```

Example 2

Assign the value **10** to the variable **A**, value **20** to variable **B**, **add** them and store the results in variable **C**

```
A = 10;        B = 20;
C = A + B;
```

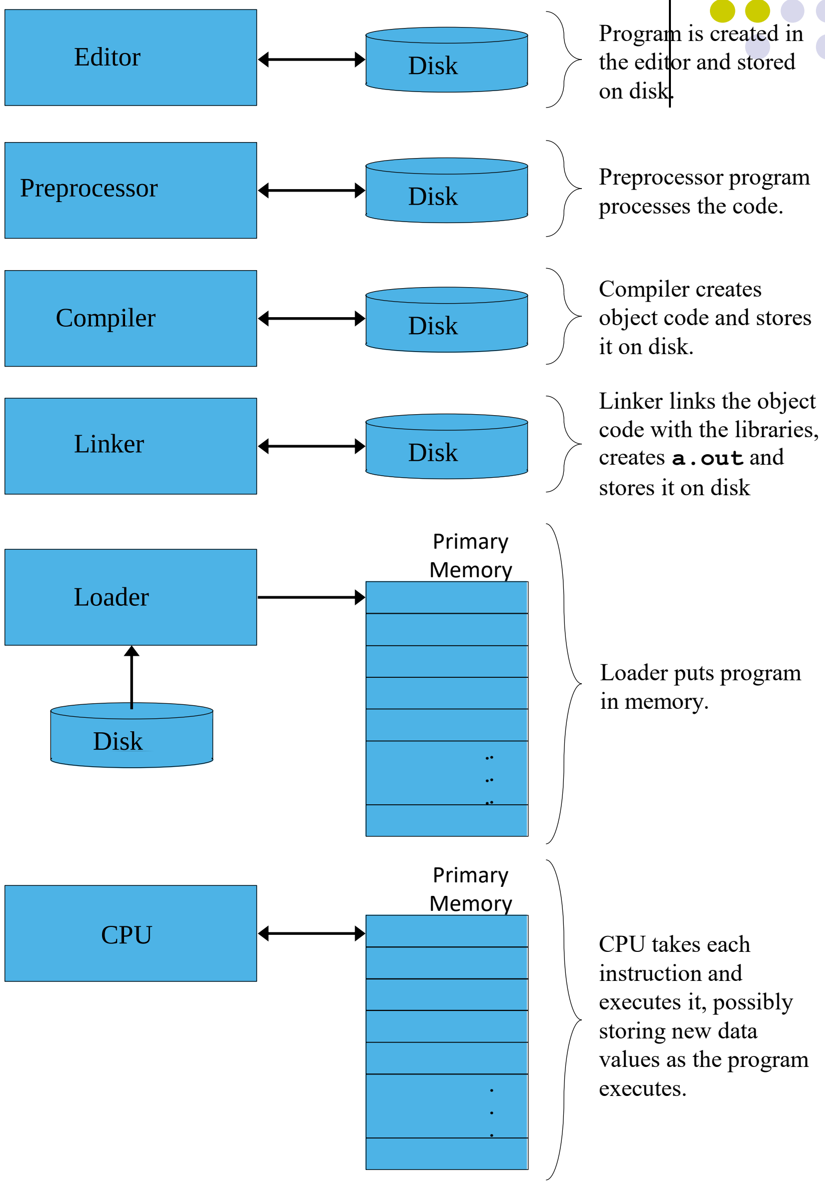
Next

- Now we move on to: C How to Program

1.13 Basics of a Typical C++ Environment

Phases of C++ Programs:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



Program is created in the editor and stored on disk.

Preprocessor program processes the code.

Compiler creates object code and stores it on disk.

Linker links the object code with the libraries, creates `a.out` and stores it on disk

Loader puts program in memory.

CPU takes each instruction and executes it, possibly storing new data values as the program executes.

1.18 Introduction to C++ Programming

- C++ language
  - Facilitates a structured and disciplined approach to computer program design

- Following are several examples
  - The examples illustrate many important features of C++
  - Each example is analyzed one statement at a time.

Outline

```
// Fig. 1.2: fig01_02.cpp
// A first program in C++
#include <iostream>

int main()
{
   std::cout << "Welcome to C++!\n";

   return 0;      // indicate that progr
}
```

```
Welcome to C++!
```

m>

to

Comments

Written between **/*** and ***/** or following a **//**.

Improve program readability and do not cause the computer to perform any action.

preprocessor directive

Message to the C++ preprocessor.

Lines beginning with **#** are preprocessor directives.

#include <iostream> tells the preprocessor to include the contents of the file **<iostream>**, which

C++ programs contain one or more functions, one of which must be **main**

Parenthesis are used to indicate a function

int means that **main** "returns" an integer value.

very function

Prints the *string* of characters contained between the

ncluding **std::cout**, the **<<**

ing **"Welcome to C++!\n"** and

), is called a *statement*.

All statements must end with a semicolon.

return is a way to exit a function from a function.

return 0, in this case, means that the program terminated normally.

1.19 A Simple Program: Printing a Line of Text

- **`std::cout`**
  - Standard output stream object
  - “Connected” to the screen
  - **`std::`** specifies the "namespace" which **`cout`** belongs to
    - **`std::`** can be removed through the use of **`using`** statements
- **`<<`**
  - Stream insertion operator
  - Value to the right of the operator (right operand) inserted into output stream (which is connected to the screen)
  - **`std::cout << “Welcome to C++!\n”;`**
- **`\`**
  - Escape character
  - Indicates that a “special” character is to be output

1.19 A Simple Program: Printing a Line of Text

Escape Sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double quote character.

- There are multiple ways to print text
  - Following are more examples

```cpp
// Fig. 1.4: fig01_04.cpp
// Printing a line with multiple statements
#include <iostream>

int main()
{
   std::cout << "Welcome ";
   std::cout << "to C++!\n";

   return 0;   // indicate that program ended successfully
}
```

```
Welcome to C++!
```

Unless new line `'\n'` is specified, the text continues on the same line.

Outline

1. Load `<iostream>`

2. `main`

2.1 Print `"Welcome"`

2.2 Print `"to C++!"`

2.3 newline

2.4 exit (`return 0`)

Program Output

```cpp
// Fig. 1.5: fig01_05.cpp
// Printing multiple lines with a single statement
#include <iostream>

int main()
{
   std::cout << "Welcome\nto\n\nC++!\n";

   return 0;   // indicate that program ended successfully
}
```

```
Welcome
to

C++!
```

Multiple lines can be printed with one statement.

Outline

1. Load `<iostream>`

2. `main`

2.1 Print `"Welcome"`

2.2 newline

2.3 Print `"to"`

2.4 newline

2.5 newline

2.6 Print `"C++!"`

2.7 newline

2.8 exit (`return 0`)

Program Output

1.20 Another Simple Program: Adding Two Integers

- Variables
  - Location in memory where a value can be stored for use by a program
  - Must be declared with a name and a data type before they can be used
  - Some common data types are:
    - `int` - integer numbers
    - `char` - characters
    - `double` - floating point numbers
  - Example: `int myvariable;`
    - Declares a variable named `myvariable` of type `int`
  - Example: `int variable1, variable2;`
    - Declares two variables, each of type `int`

Primitive Data types

Name	Description	Size	Range
char	Character or small integer	1 byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer	2 bytes	signed: -32768 to 32767 unsigned: 0 to 65535
Int	Integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
iong int (long)	Long integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false	1 byte	true or false
float	Floating point number	4 bytes	+/- 3.4e +/- 38 (~7 digits) (24 –bit coefficient, 8 bit exponent)
double	Double precision floating point number	8 bytes	+/- 1.7e +/- 308 (~15 digits) (53 –bit coefficient, 11 bit exponent)
long double	Long double precision floating point number	16 bytes	+/- 1.2e +/- 4932 (~19 digits)
wchar_t	Wide character	2 or 4 bytes	1 wide character

1.20 Another Simple Program: Adding Two Integers

- **>>** (stream extraction operator)
  - When used with **`std::cin`**, waits for the user to input a value and stores the value in the variable to the right of the operator
  - The user types a value, then presses the *Enter* (Return) key to send the data to the computer
  - Example:

    ```
    int myVariable;
    std::cin >> myVariable;
    ```

    - Waits for user input, then stores input in **`myVariable`**
- **=** (assignment operator)
  - Assigns value to a variable
  - Binary operator (has two operands)
  - Example:

    ```
    sum = variable1 + variable2;
    ```

```
// Fig. 1.6: fig01_06.cpp
// Addition program
#include <iostream>

int main()
{
   int integer1, integer2, sum;        // declaration

   std::cout << "Enter first integer\n"; // p
   std::cin >> integer1;               // r
   std::cout << "Enter second integer\n"; // prompt
   std::cin >> integer2;               // read an integer
   sum = integer1 + integer2;          // assignment of sum
   std::cout << "Sum is " << sum << std::endl; // print sum

   return 0;   // indicate that program ended success
}
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Notice how **std::cin** is used to get user input.

std::endl flushes the buffer and prints a newline.

Variables can be output using **std::cout << variableName**.

Outline

1. **Load <iostream>**

2. main

2.1 Initialize variables integer1,

m

rst integer"

2.2.1 Get input

2.3 Print "Enter second integer"

2.4 Add variables and put result into sum

2.5 Print "Sum is"

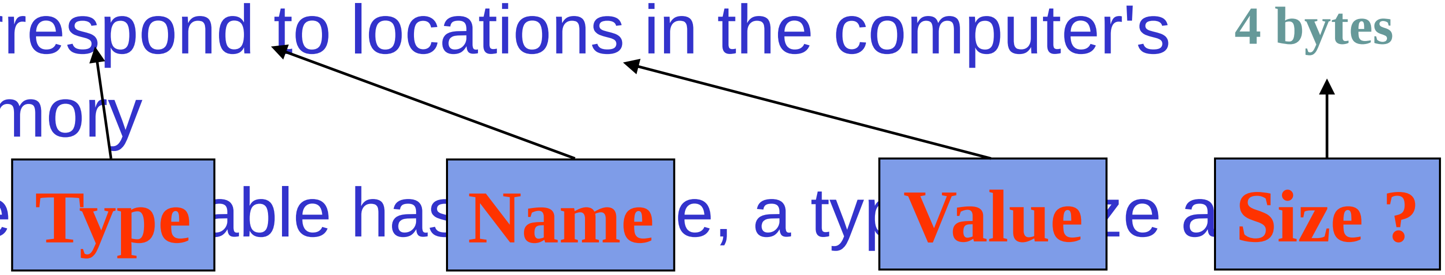
2.5.1 Output sum

2.6 exit (return 0)

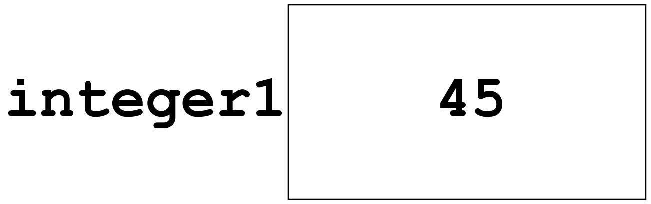
Program Output

1.21 Memory Concepts

- Variable names
  - Correspond to locations in the computer's memory



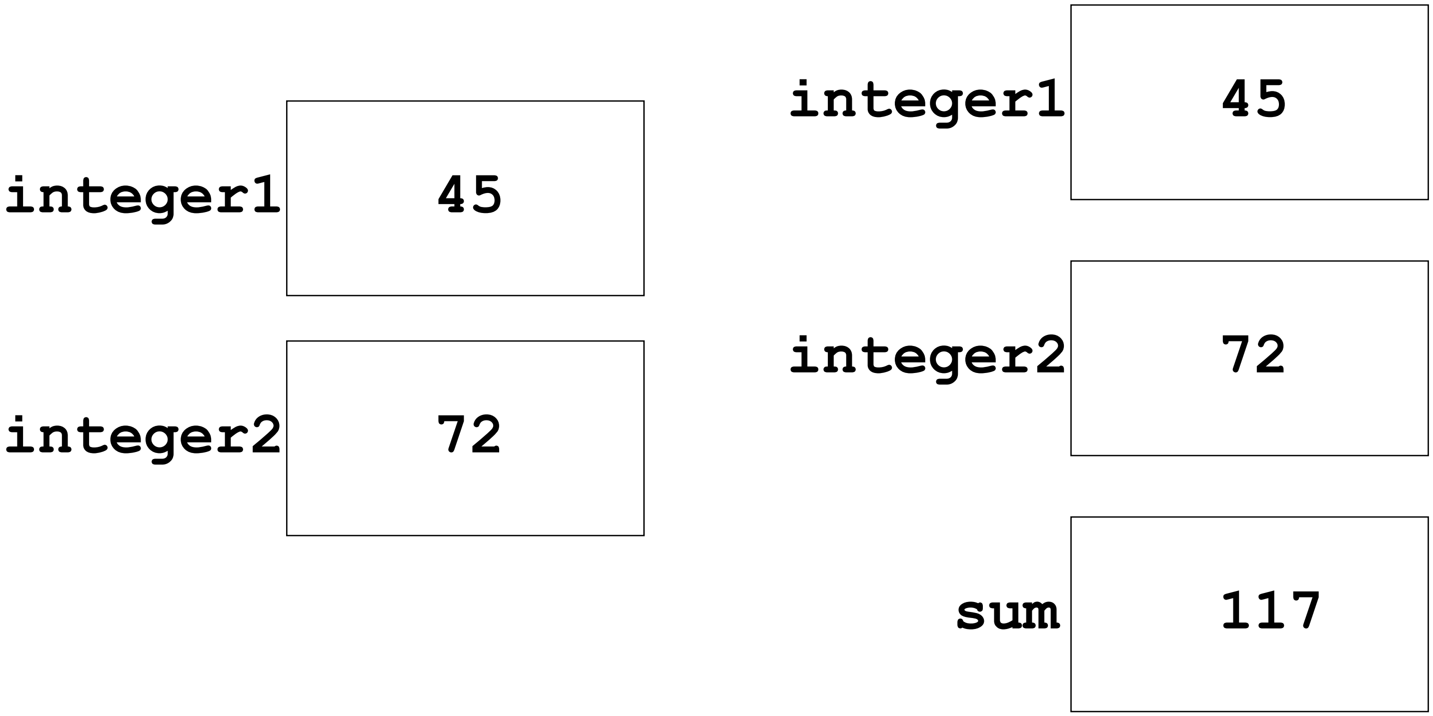
  - Eve able has e, a typ ze a value
    - `int myVariable = 10;`



  - Reading variables from memory does not change them
- A visual representation

1.21 Memory Concepts

- A visual representation (continued)



integer1 45

integer2 72

integer1 45

integer2 72

sum 117

1.22 Arithmetic

- Arithmetic calculations
  - Use `*` for multiplication and `/` for division
  - Integer division truncates remainder
    - `7 / 5` evaluates to 1
  - Modulus operator returns the remainder
    - `7 % 5` evaluates to 2
- Operator precedence
  - Some arithmetic operators act before others (i.e., multiplication before addition)
    - Be sure to use parenthesis when needed
  - Example: Find the average of three variables a, b and c
    - Do not use: `a + b + c / 3`
    - Use: `(a + b + c ) / 3`

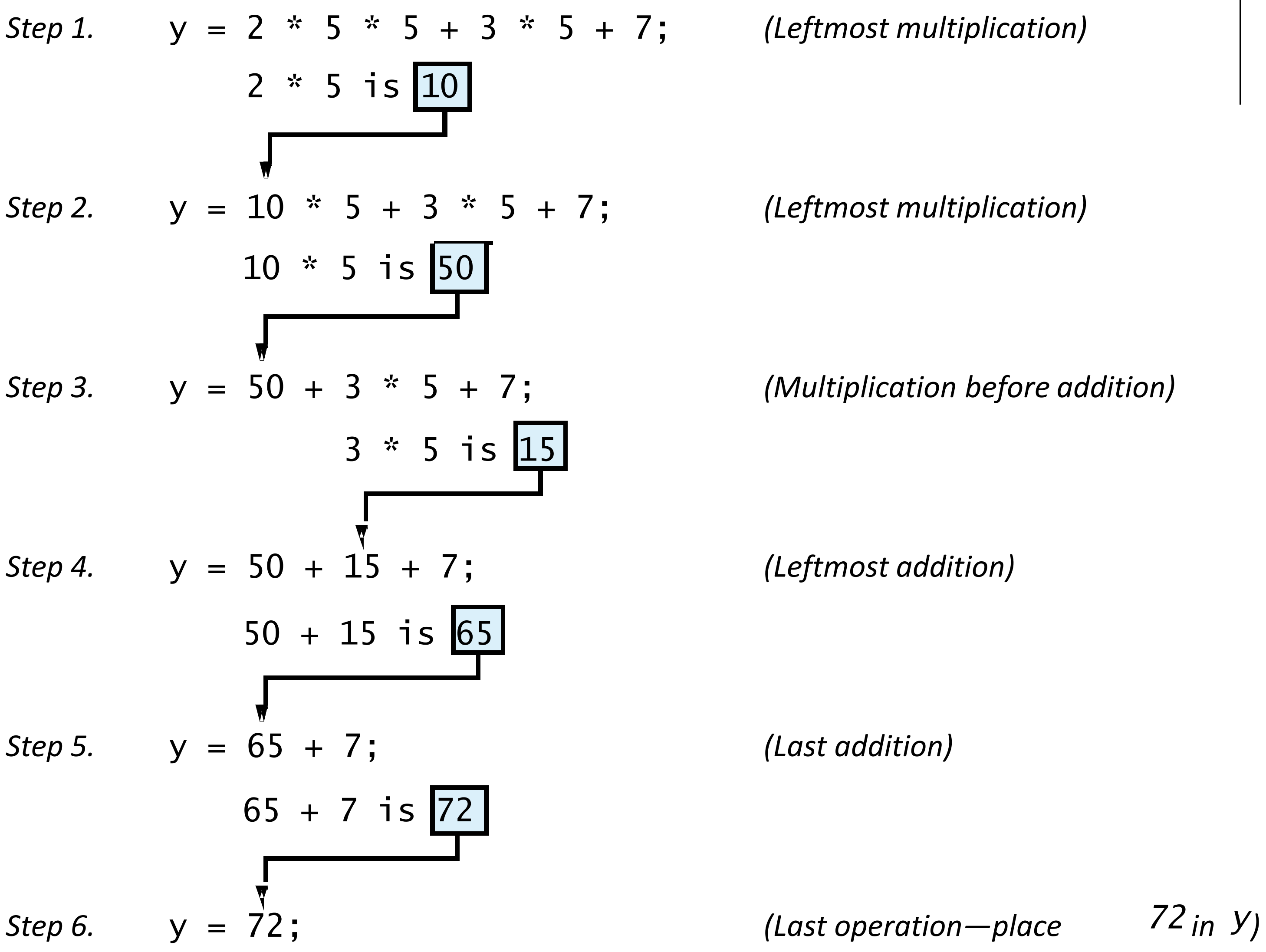
1.22 Arithmetic

- Arithmetic operators:

C++ operation	Arithmetic operator	Algebraic expression	C++ expression
Addition	`+`	*f* + 7	`f + 7`
Subtraction	`-`	*p* – *c*	`p - c`
Multiplication	`*`	***bm***	`b * m`
Division	`/`	***x*** / ***y***	`x / y`
Modulus	`%`	***r mod s***	`r % s`

Operator(s)	Operation(s)	Order of evaluation (precedence)
`()`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.
`*`, `/`, or `%`	Multiplication Division Modulus	Evaluated second. If there are several, they re evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Arithmetic Example



Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is 10`

Step 2. `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

`50 + 15 is 65`

Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is 72`

Step 6. `y = 72;` *(Last operation—place 72 in y)*

Conversion from Fahrenheit to Celsius

- Output
  - Temperature in Celsius (C)
- Inputs
  - Temperature in Fahrenheit (F)
- Process

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

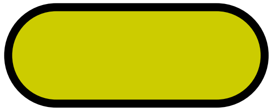
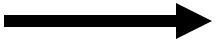
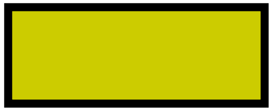
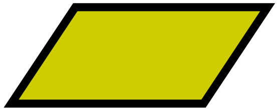
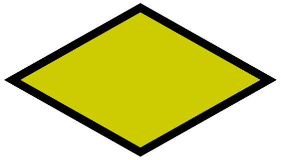
Calculate and print the average grade of 3 tests for the entire class

- Input
  - 3 test scores for each student
- output
  - Average of 3 tests for each student
- Process
  1. Get three scores
  2. Add them together
  3. Divide by three to get the average
  4. Print the average
  5. Repeat step 1 to 4 for next student
  6. Stop if there are no more students

Flow Charts

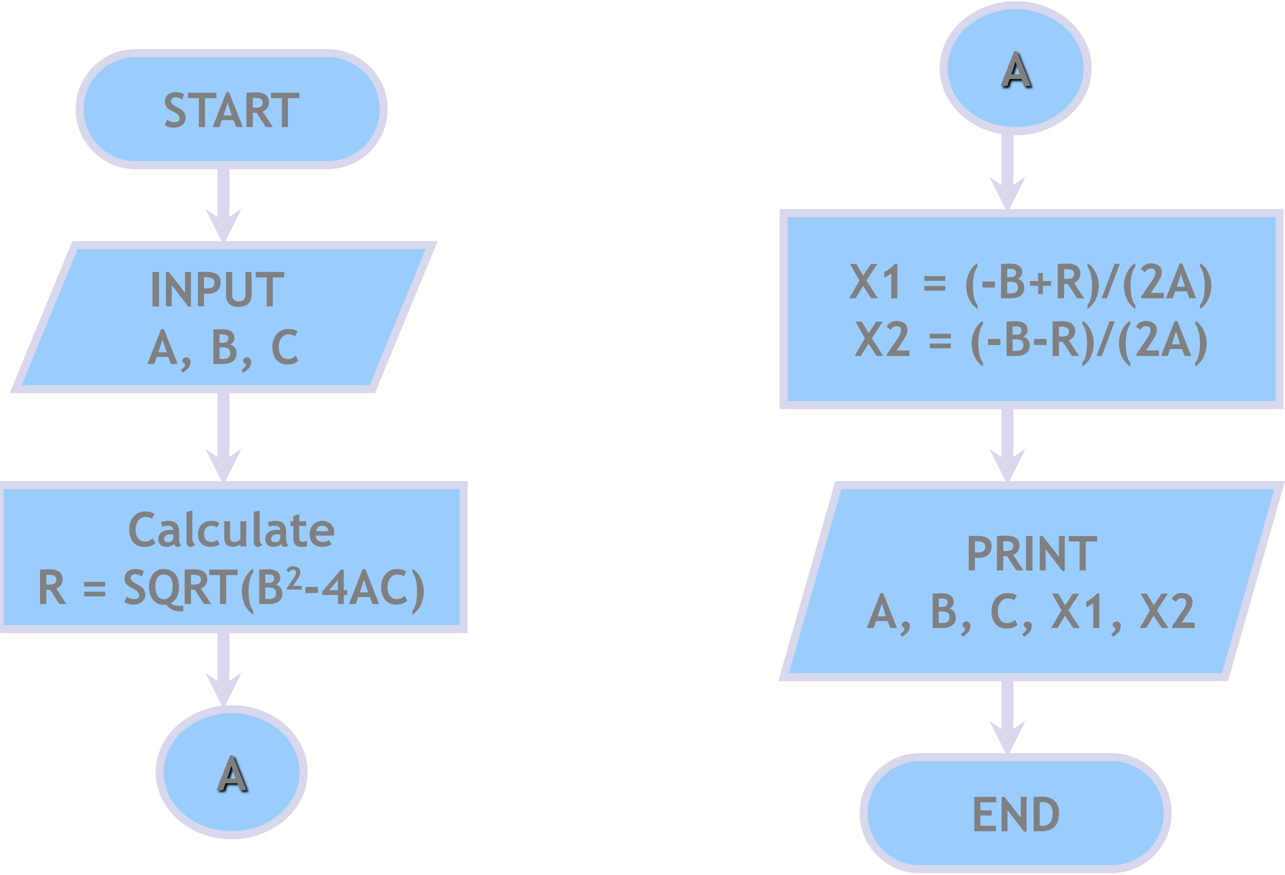
- A flowchart is a visual or graphical representation of an algorithm.
- The flowchart employs a series of blocks and arrows, each of which represents a particular operation or step in the algorithm.
- The arrows represent the sequence in which the operations are implemented.

Flowcharts – Most Common Symbols

Symbol	Name	Function
	Terminal	Represents the beginning or end of a program.
	Flow-line	Represents the flow of logic.
	Process	Represents calculations or data manipulation.
	Input/Output	Represents inputs or outputs of data and information.
	Decision	Represents a comparison, question, or decision that determines alternative paths to be followed.

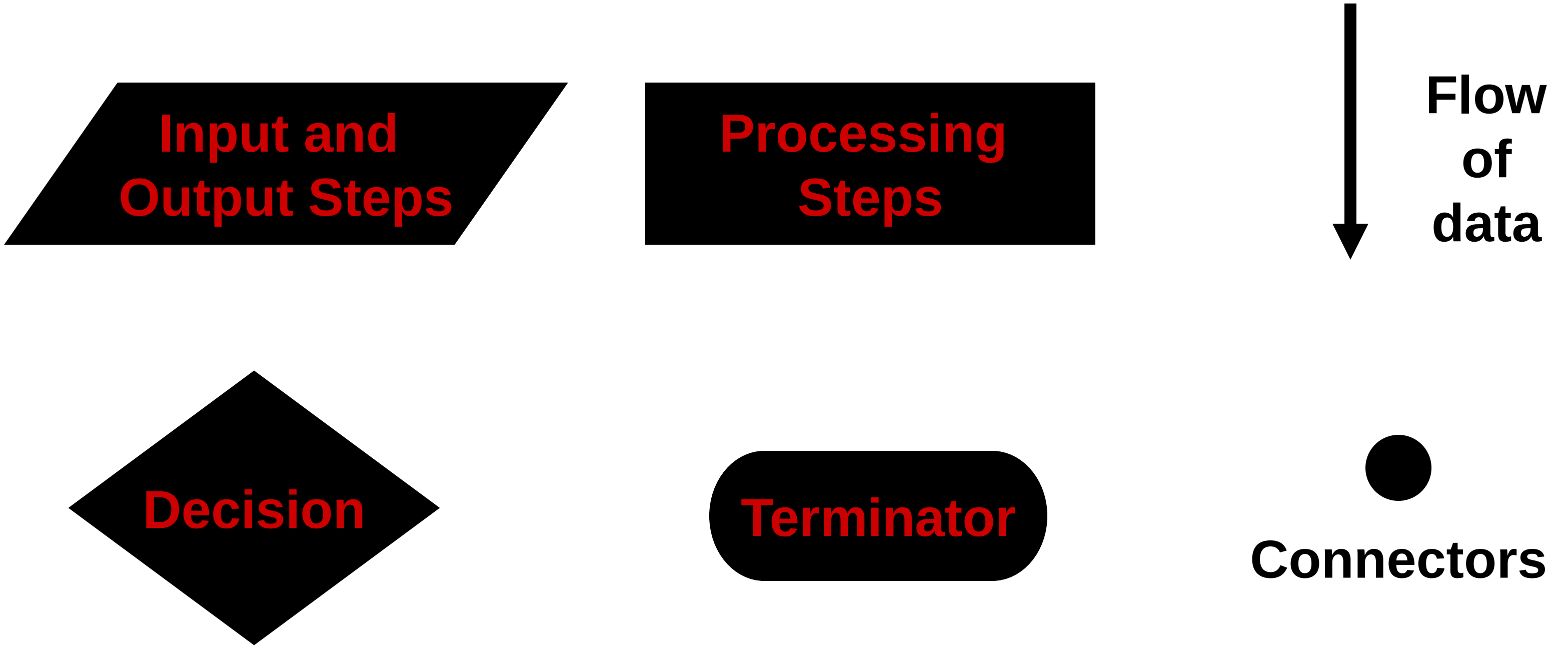
Flowcharts – An Example

Find the solution of a quadratic equation $Ax^2+Bx+C=0$, given A, B and C.



Flow Charting

Expresses the flow of processing in a structured pictorial format.



Input and Output Steps

Processing Steps

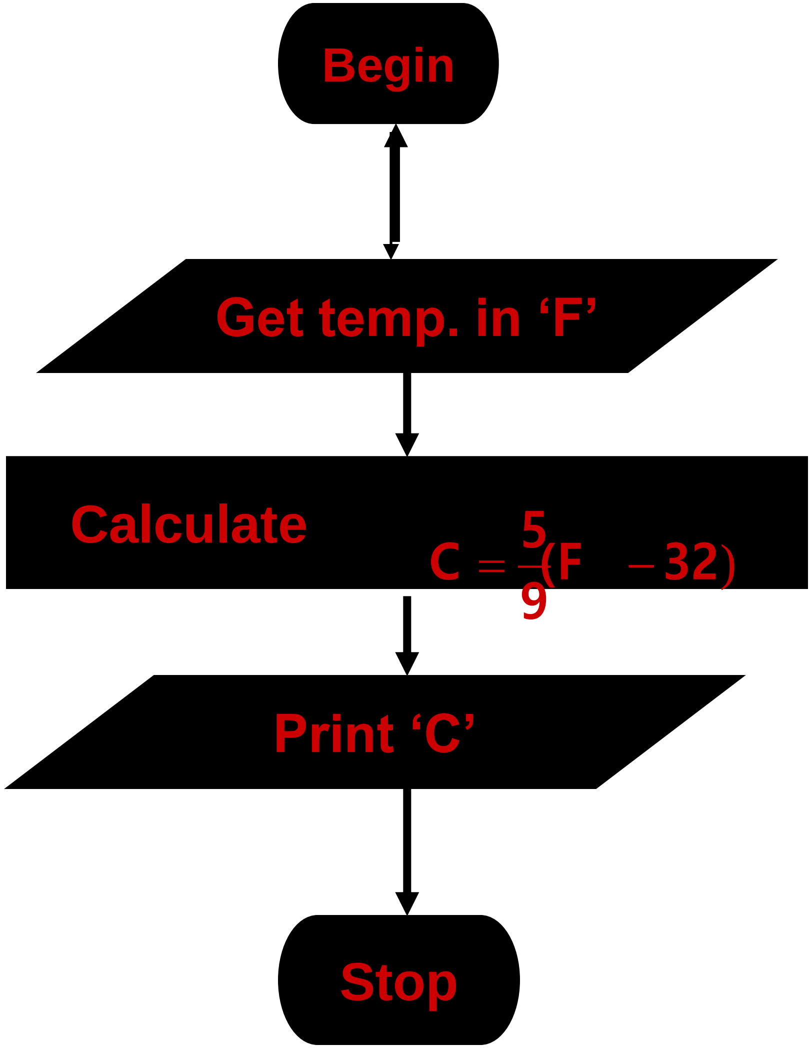
Flow of data

Decision

Terminator

Connectors

Flow chart for Converting Fahrenheit into Celsius



Begin

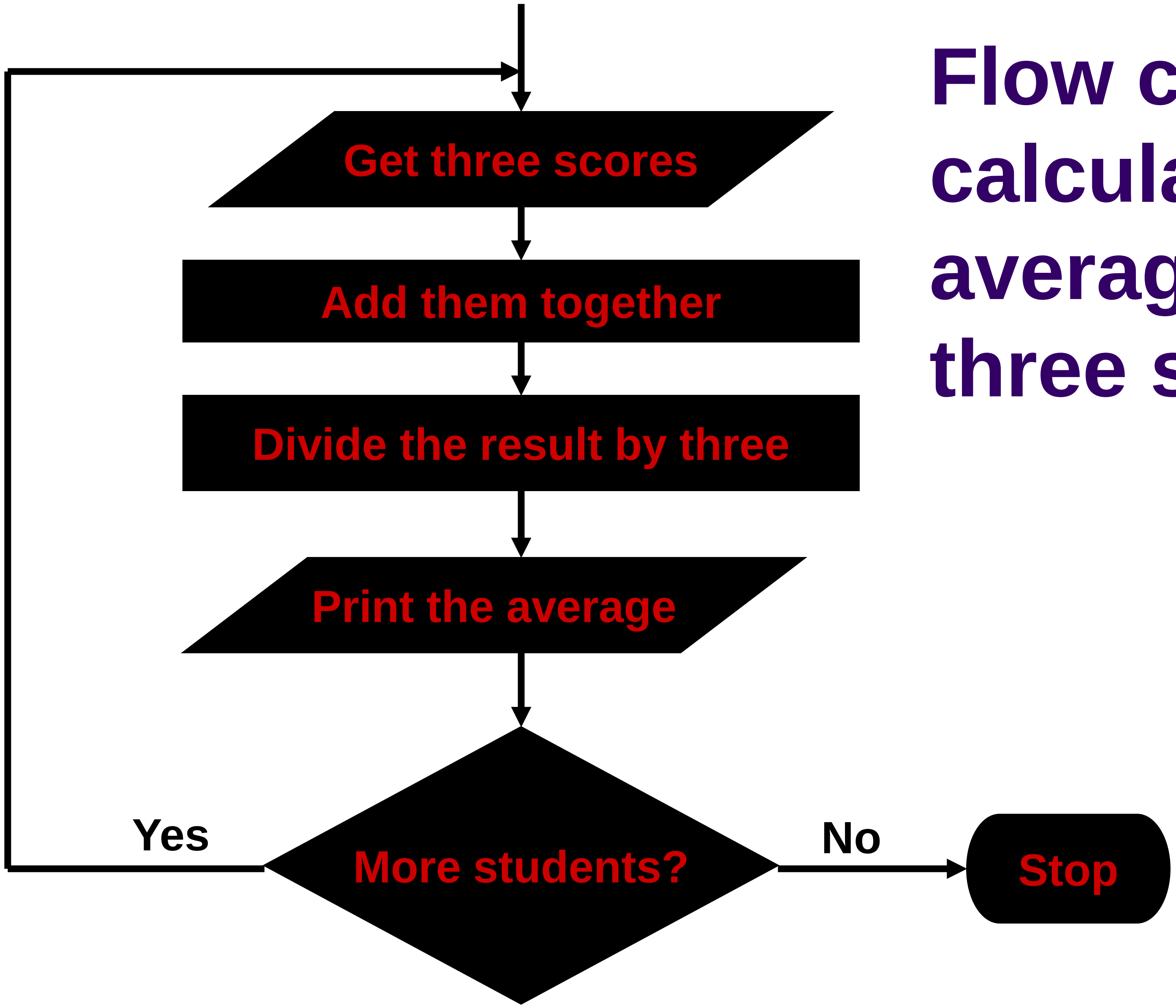
Get temp. in ‘F’

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

Print ‘C’

Stop

Flow chart for calculating average of three scores



Comparison of Algorithm representations in Natural language, flowchart and Pseudo-code

Step 1: Begin the calculations

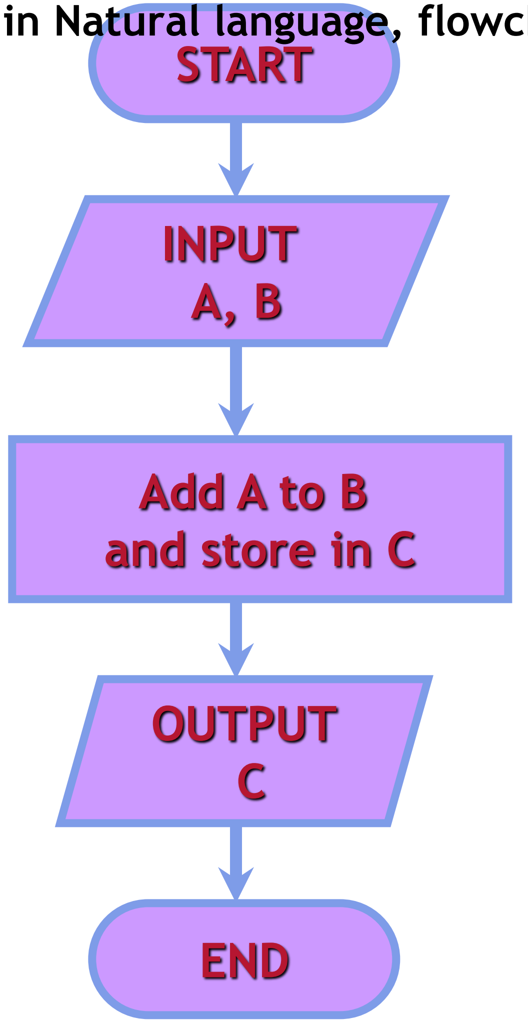
Step 2: Input two values A and B

Step 3: Add the values

Step 4: Display the result

Step 5: End the calculation

Natural language



Flowchart

```
BEGIN Adder
    Input A and B
    C = A + B
    PRINT C
END Adder
```

Pseudo-code

Using Namespecifier

- **`using`** statements
  - Eliminate the need to use the **`std::`** prefix
  - Allow us to write cout instead of **`std::cout`**
  - To use the following functions without the **`std::`** prefix, write the following at the top of the program

```
using std::cout;
using std::cin;
using std::endl;
```

Decision Making: Equality and Relational Operators

- **`if`** structure
  - Test conditions truth or falsity. If condition met execute, otherwise ignore
- Equality and relational operators
  - Lower precedence than arithmetic operators
- Table of relational operators on next slide

Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	**`>`**	**`x > y`**	**`x`** is greater than **`y`**
<	**`<`**	**`x < y`**	**`x`** is less than **`y`**
≥	**`>=`**	**`x >= y`**	**`x`** is greater than or equal to **`y`**
≤	**`<=`**	**`x <= y`**	**`x`** is less than or equal to **`y`**
Equality operators			
=	**`==`**	**`x == y`**	**`x`** is equal to **`y`**
≠	**`!=`**	**`x != y`**	**`x`** is not equal to **`y`**

```
// Fig. 1.14: fig01_14.cpp
// Using if statements, relational
// operators, and equality operators
#include <iostream>

using std::cout;  // program uses cout
using std::cin;   // program uses cin
using std::endl;  // program uses endl

int main()
{
   int num1, num2;

   cout << "Enter two integers, and I will tell you\n"
        << "the relationships they satisfy: ";
   cin >> num1 >> num2;   // read two integers

   if ( num1 == num2 )
      cout << num1 << " is equal to " << num2 << endl;

   if ( num1 != num2 )
      cout << num1 << " is not equal to " << num2 << endl;

   if ( num1 < num2 )
      cout << num1 << " is less than " << num2 << endl;

   if ( num1 > num2 )
      cout << num1 << " is greater than " << num2 << endl;

   if ( num1 <= num2 )
      cout << num1 << " is less than or equal to "
           << num2 << endl;

```

num1 → 3 num2 → 7

Notice the **`using`** statements.

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
```

The **`if`** statements test the truth of the condition. If it is **`true`**, body of **`if`**

```
3 is not equal to 7
```

If not, body is skipped.

```
3 is less than 7
```

iple statements in a body, delineate them with braces **`{}`**.

```
3 is less than or equal to 7
```

Outline

1. Load `<iostream>`

n

2.1 Initialize `num1` and `num2`

2.1.1 Input data

nts

```
    if ( num1 >= num2 )
       cout << num1 << " is greater than or equal to "
              << num2 << endl;

    return 0;   // indicate that program ended successfully
}
```

Outline

2.3 exit (`return 0`)

Program Output

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

3 != 7 ~~3 == 7~~

3 < 7 ~~3 > 7~~

3 <= 7 ~~3 >= 7~~

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

22 != 12 ~~22 == 12~~

22 > 12 ~~22 < 12~~

22 >= 12 ~~22 <= 12~~

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

7 == 7 ~~7 != 7~~

7 <= 7 ~~7 > 7~~

7 >= 7 ~~7 < 7~~

2.6 Key words

Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Fig. 2.15 C’s reserved keywords.