
EEE146
PROGRAMMING –I

Course Information

- **Name of the Course:** Programming-I
- **Lecturers:** Dr. Seydi Kaçmaz & Dr. Sema Kayhan
- **E-mail:** seydikacmaz@gantep.edu.tr , skoc@gantep.edu.tr
- **Web Announcements:** Follow web announcements

Text Book and References:

1- Delores M.Etter & Jeanine A.Ingber, Engineering Problem Solving with C++, Pearson
2- Harvey M.Deitel & Paul J.Deitel, C++ How to Program, Pearson
3- John R.Hubbard, Schaum's Outline of Theory and Problems of Programming with C++
4- Steve Oualline, Practical C++ Programming, O'Reilly & Associates, Inc

Web pages: **http://cpp.gantep.edu.tr**

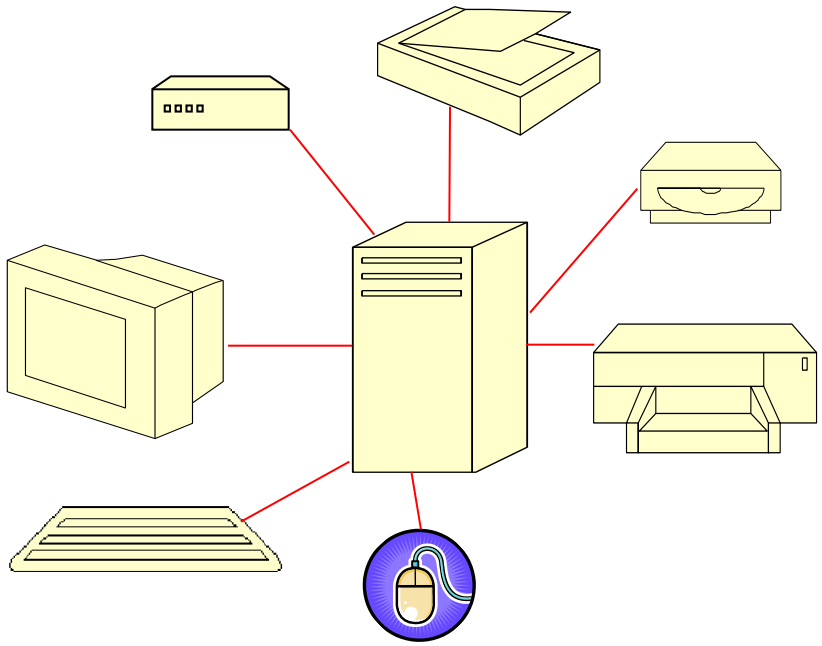
Compiler: **Dev C++**

Grading: Midterm 1 (%22,5) & Midterm 2 (%22,5) + Laboratory (%15) + Final (%40)

Syllabus

1. Introduction: Computer organization, algorithms, variables, data types, operators, intrinsic functions
2. Selection control structure: if, switch statements
3. Repetition control structure: for, while, do-while loops
4. Functions
5. Arrays
6. Vectors
7. Pointers
8. File input, output
9. String operations

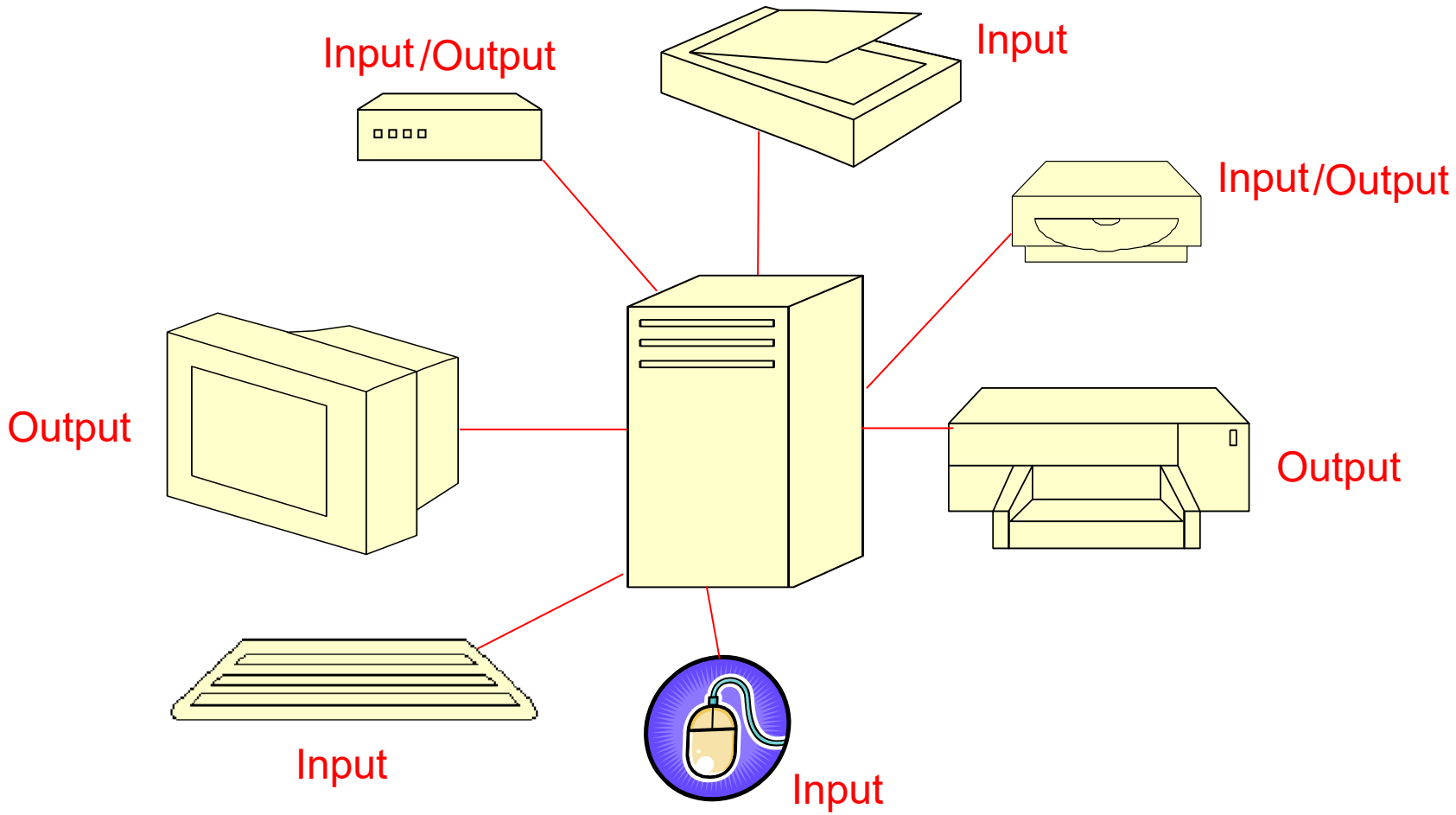
A Computer System



A computer system is composed of:

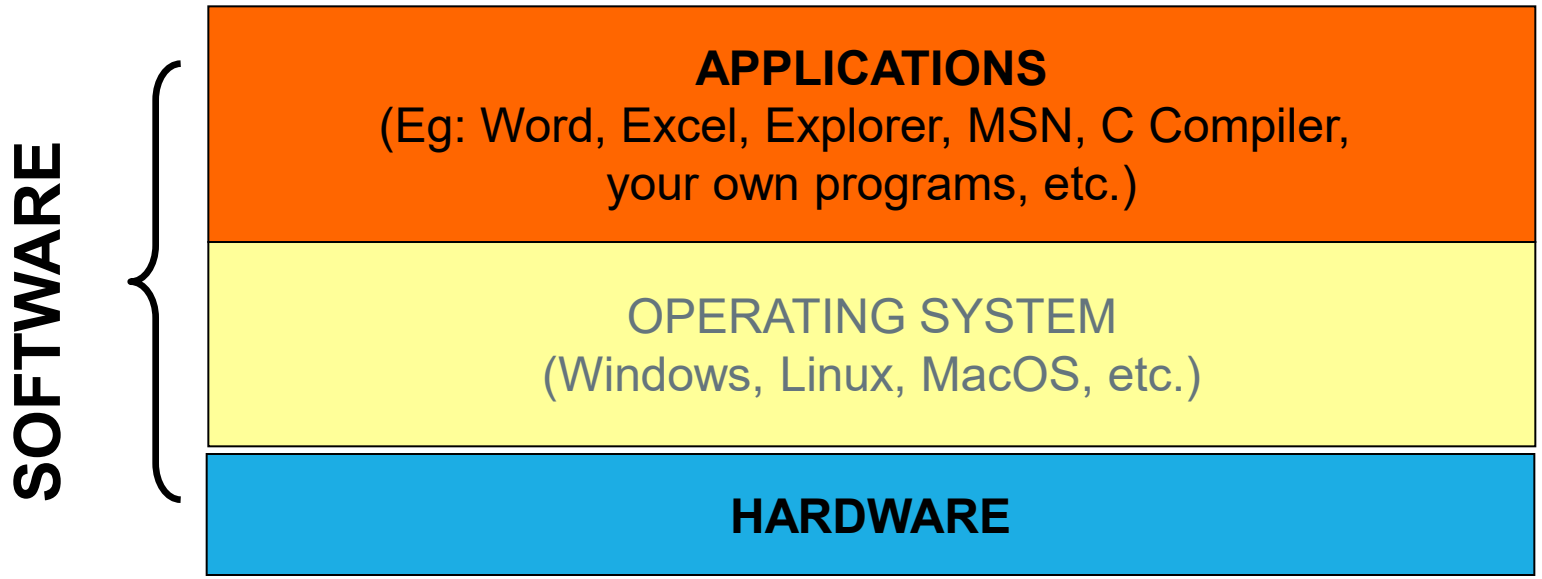
- a monitor,
- a keyboard,
- a mouse,
- and a case (that contains several controlling components such as processor and alike),
- and also other peripherals like
  - CD player (might have been included in the case),
  - printer,
  - scanner,
  - modem,
  - etc.

all connected together


Input /Output
Input
Input/Output
Output
Output
Input
Input

A Computer System

- Everything we had in the previous slide is **hardware**.
  - i.e., physical components that implement what is requested by the **software**.



A Computer System

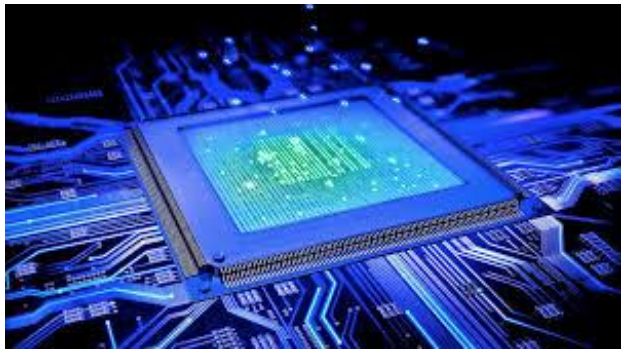
In this course, we will learn how to develop our own software (using C++ language), but we need to understand how our programs will be executed by the hardware.

CPU: Central Processing Unit

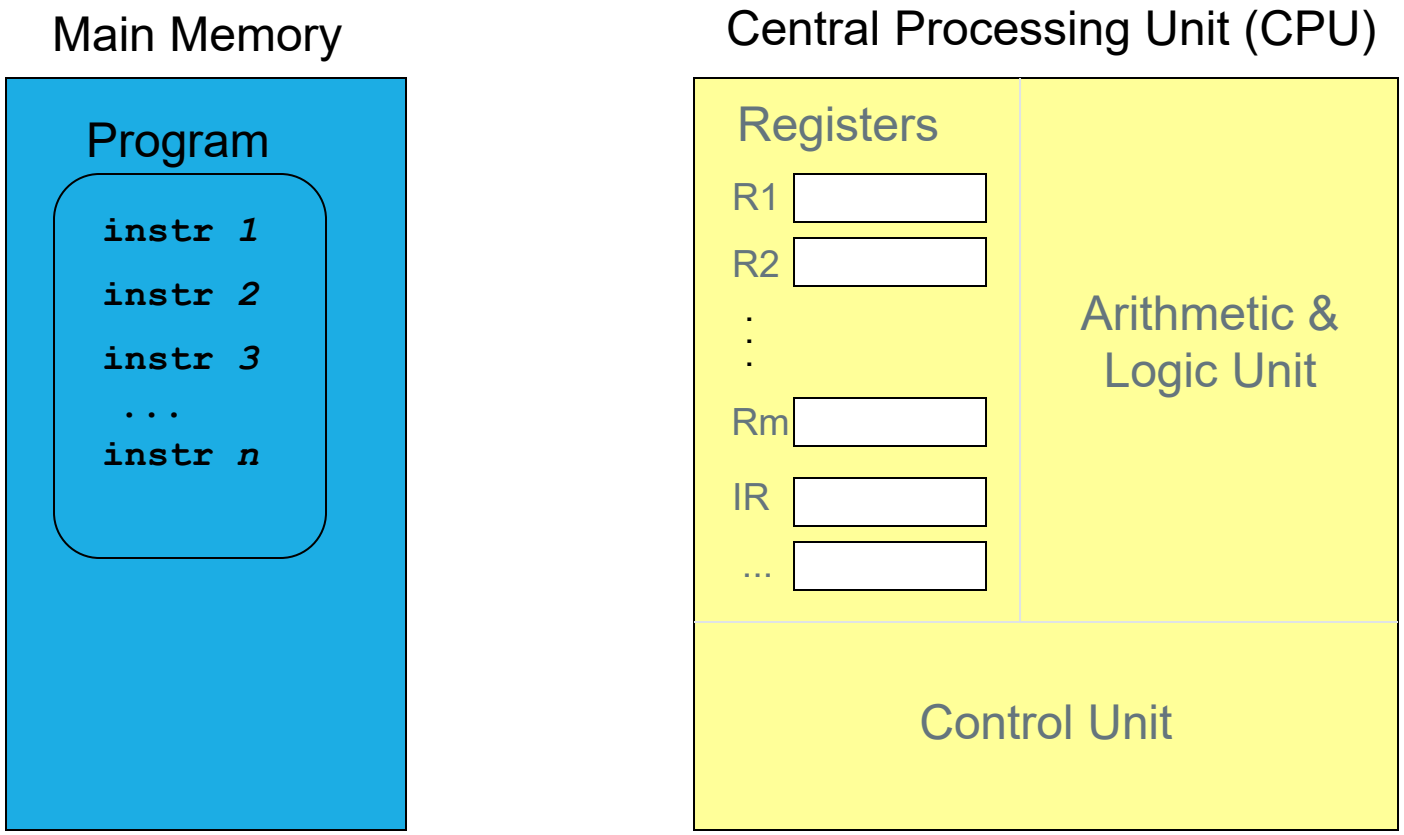
In terms of hardware, CPU is the most important part for us.

It does all processing and control.

Everything on the computer is controlled and executed by the CPU.



How are the instructions executed ?



How do we write programs ?

We write our programs in *"C++ language"* (which is an English-like language)

```
#include <iostream>
using namespace std;
int main()
{
   cout<<"Hello world!";
   return 0;
}
```

(source code)

We use a compiler (such as Visual C++, Dev C++, etc.) to translate our program from "C++ language" to *"machine language"*

Compile & Link

(object code)

This is the executable code in "machine language."

This is the only thing the computer can understand and run (execute).

```
11101010110010010100010100100010000100101001010101000101001000100101001
```

(machine code or executable code)

Statement vs. Instruction

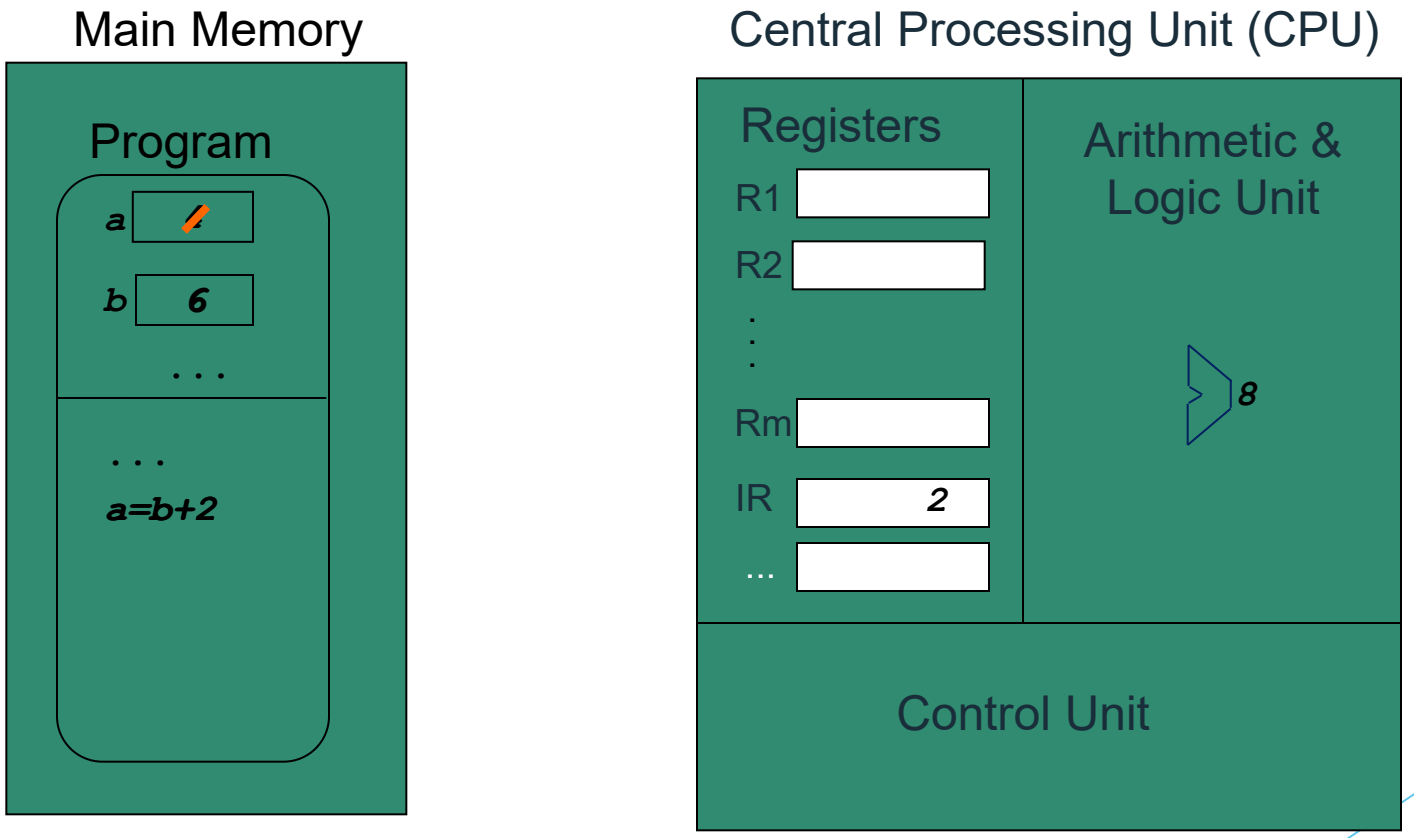
- Our source code (in C++) is composed of statements.
  - Eg: `a=b+c/2;`
- The corresponding machine code is composed of instructions.
  - Eg:
```
1101001010110010 (divide c by 2)
0110100100100101 (add it to b)
1010110110111011 (put the result in a)
```
  - CPU is capable of executing instructions, not statements. Statements may be too complex.
  - Compiler *implements* each statement using several instructions.
    - Eg: The statement "`a=b+c/2;`" can be implemented as
```
temp1 = c/2
a = b + temp1
```

Why have input/output ?

- A program should not always produce the same output.
  - O/w, you may keep the result and delete the program after you run it for the first time.
- A program should be consistent; i.e., it should not produce random results.
- Therefore, a program should take some input, process it, and produce some output as the result of that input.

Execution of an instruction

- Let's see how an instruction like "**a=b+2**" is executed.
  - Assume initially **a** is 4 and **b** is 6.



Our first C++ program: Hello World

- Every C program has a `main()` function.
  - It wraps all the statements to be executed.
- We make use of previously written functions. They are provided by header files.
  - Typically, we include the *standard input/output header* file, named iostream.
- We write all statements inside the `main()` function.

```
#include <iostream>
using namespace std;
int main()
{
   cout<<"Hello world!";
   return 0;
}
```

Need for input

- Note that the Hello World program has no input.
  - Therefore, it always produces the same output:

```
Hello World
```

  - So, after we run this program once, we know what it will always produce. Therefore, we don't need the program anymore; we can safely delete it.
- Definitely this is not what we want. (O/w, nobody will pay us ☺)
  - We want to write programs that can take input and produce different results according to the input.

A program that also performs input

C++ Program

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, c;

    cout<<"Enter two numbers: ";
    cin>>a>>b;
    c=a+b;
    cout<<"Result is"<<c;
    return 0;
}
```

\n

Read two integers (decimals) into variables a and b

Display the value of variable c after the text "Result is"

User screen

```
Enter two numbers: 5 8

Result is 13
```

Problem Solving with Computers

- Problem solving with computers involves several steps:
- 1.Clearly define the problem
- 2.Analyse the problem and formulate a method to solve it
- 3.Describe the solution in the form of an algorithm.
- 4.Draw a flowchart of the algorithm
- 5.Write the computer program
- 6.Compile and run the program (debugging)
- 7.Test the program (debugging)
- 8.Interpretation of results

The Objective first:

- To practice thinking algorithmically
- To understand and be able to implement proper program development
- To start learning about control structures
- To be able to express an algorithm using a flow chart

Algorithm and flowchart

Algorithm consists of a series of step-by step instructions for the solution of a problem.

Flowchart is a pictorial form of an algorithm.

What is an Algorithm?

- Steps used to solve a problem
- Problem must be
  - Well defined
  - Fully understood by the programmer

- Steps must be
  - Ordered
  - Unambiguous
  - Complete

Developing an Algorithm

Program Development

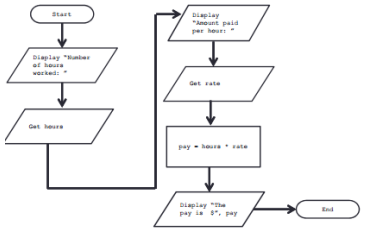
1. Understand the problem
2. Represent your solution (your algorithm)
   - Pseudocode
   - Flowchart
3. Implement the algorithm in a program
4. Test and debug your program

Step 1: Understanding the Problem

- Input
  - What information or data are you given?
- Process
  - What must you do with the information/data?
  - **This is your algorithm!**
- Output
  - What are your deliverables?

Step 2: Represent the Algorithm

- Can be done with flowchart or ***pseudocode***

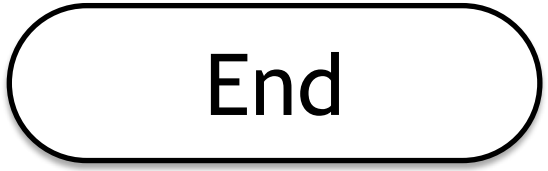


- Flowchart
  - Symbols convey different types of actions
- Pseudocode
  - A cross between code and plain English
- One may be easier for you – use that one

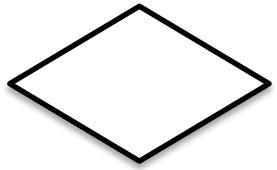
Flowchart Symbols

Start Symbol

Input/Output



End Symbol



Decision Symbol

Data Processing Symbol

Flow Control Arrows

Exercise

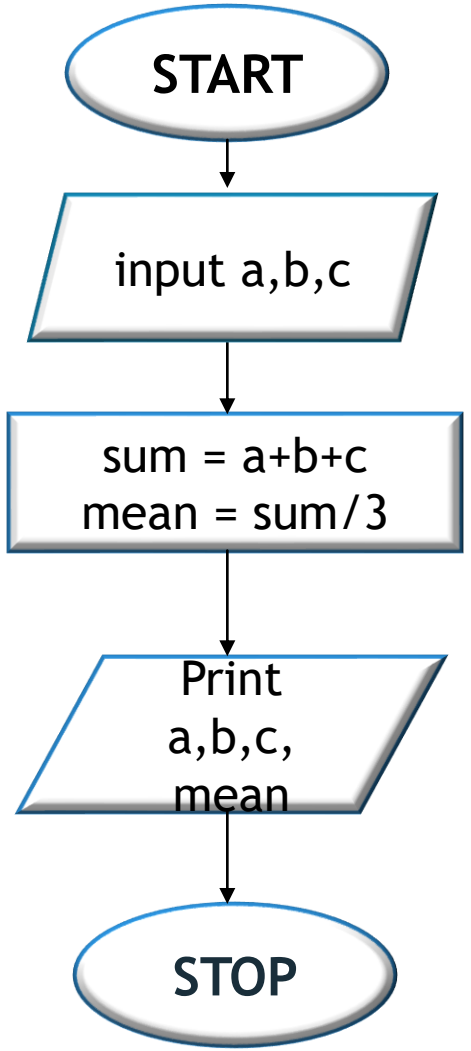
- Write an algorithm that asks a user for their name, then responds with “Hello NAME”

Pseudocode

1. **Display "what is your name: "**
2. **input the NAME**
3. **Display "Hello"**
4. **Output NAME**

Mean of three integers Example

Inputs: value of 3 integers
Process: sum and calculate mean
Output: mean value



Pseudocode

S1: Start

S2: Input a,b,c

S3: Set sum=a+b+c

S4: Set mean=sum/3

S5: Output a,b,c,mean

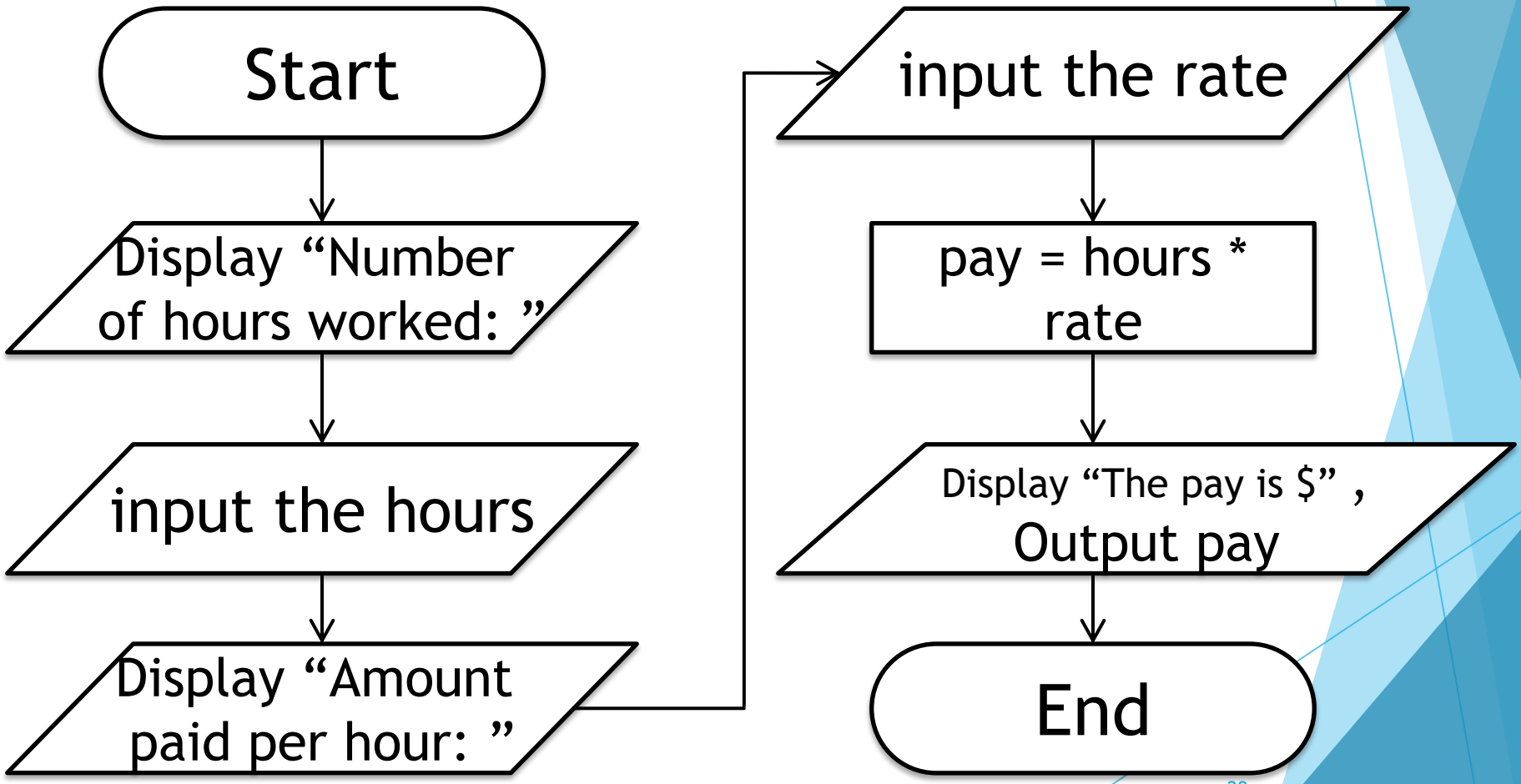
S6: End

“Weekly Pay” Example

- Create a program to calculate the weekly pay of an hourly employee
  - What is the input, process, and output?

- Input: pay rate and number of hours
- Process: multiply pay rate by number of hours
- Output: weekly pay

Flowchart


Start
Display “Number of hours worked: ”
input the hours
Display “Amount paid per hour: ”
input the rate
pay = hours * rate
Display “The pay is $” , Output pay
End

Pseudocode

- Start with a plain English description, then…

1. `Display "Number of hours worked: "`
2. `input the hours`
3. `Display "Amount paid per hour: "`
4. `input the rate`
5. `Compute pay = hours * rate`
6. `Display "The pay is $"`
7. `Output  pay`

Steps 3 and 4: Implementation and Testing/Debugging

- We'll cover implementation in detail next class
- Testing and debugging your program involves identifying errors and fixing them
  - We'll talk about this later today

Algorithms and Language

- Notice that developing the algorithm didn’t involve any C++ at all
  - Only pseudocode or a flowchart was needed
  - An algorithm can be coded in any language
- All languages have 3 important tools called ***control structures*** that we can use in our algorithms

Control Structures

Control Structures

- Structures that control how the program “flows” or operates, and in what order

- 1-Sequence
- 2-Decision Making
- 3-Looping

Sequence

- One step after another, with no branches
- Already wrote one for “Weekly Pay” problem
- What are some real life examples?
  - Dialing a phone number
  - Purchasing and paying for groceries

Decision Making

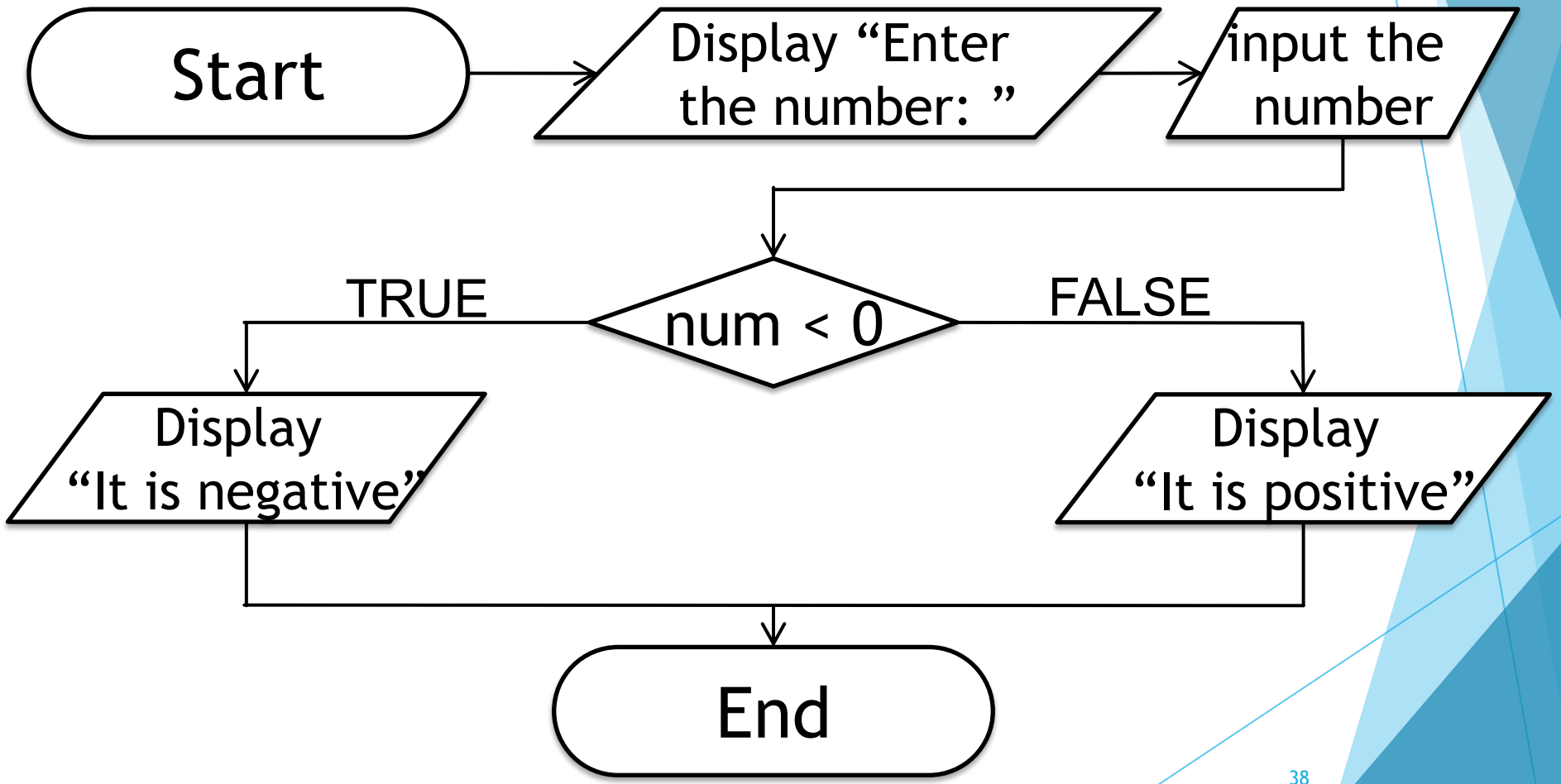
- Selecting one choice from many based on a specific reason or condition
  - If something is true, do **A** … if it's not, do **B**
- What are some real life examples?
  - Choosing where to eat lunch

Decision Making: Pseudocode

- Answer the question “Is a number positive?”
  - Start with a plain English description

```
1. Display "Enter the number: "
2. input the number (call it num)
3. If num < 0
4.     Display "It is negative"
5. Else
6.     Display "It is positive"
```

Decision Making: Flowchart



Looping

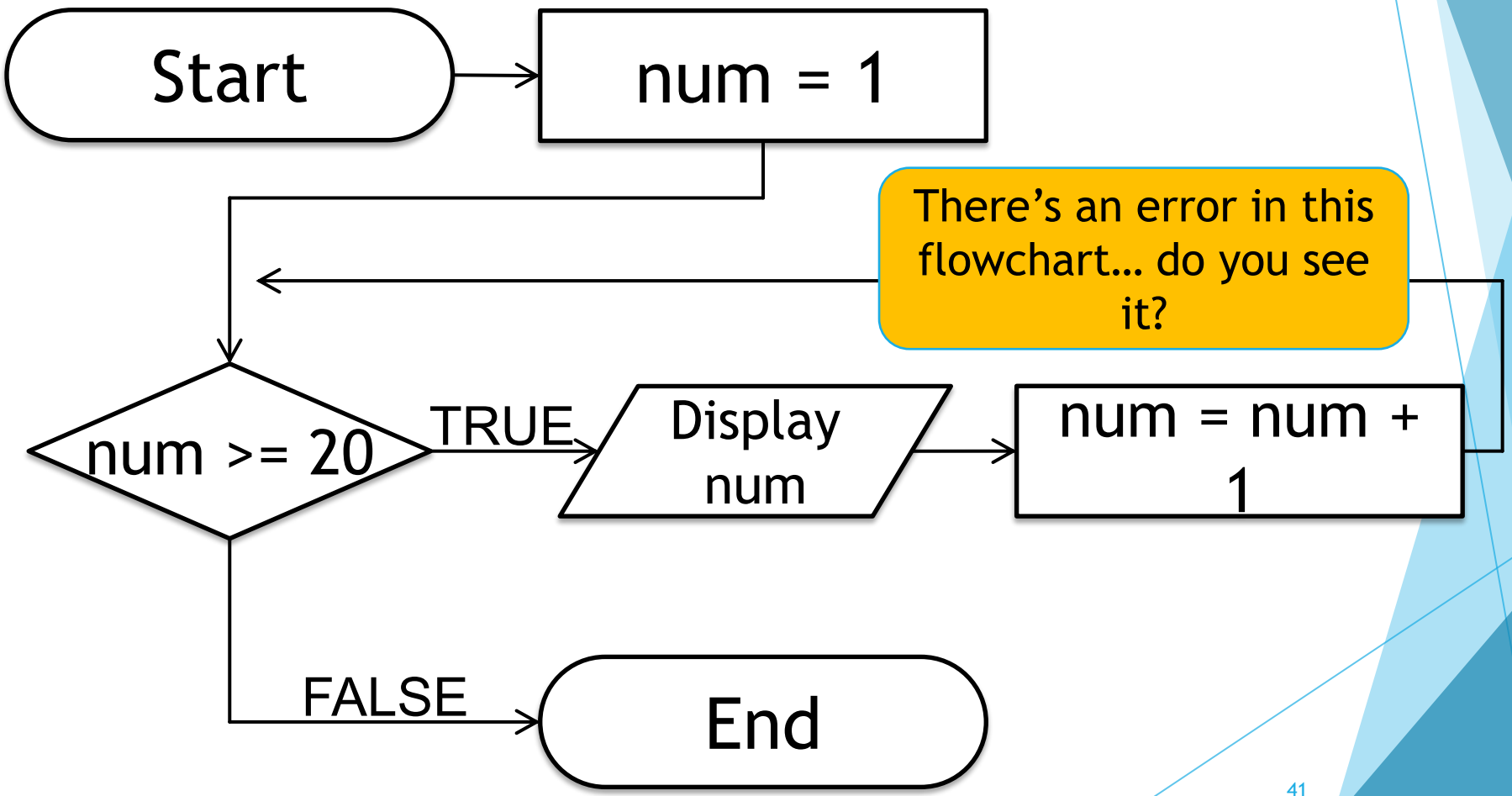
- Doing something over and over (and over) again
- Used in combination with decision making
  - Otherwise we loop forever
    - This is called an “infinite loop”
- What are some real life examples?
  - Doing homework problem sets
  - Walking up steps

Looping: Pseudocode

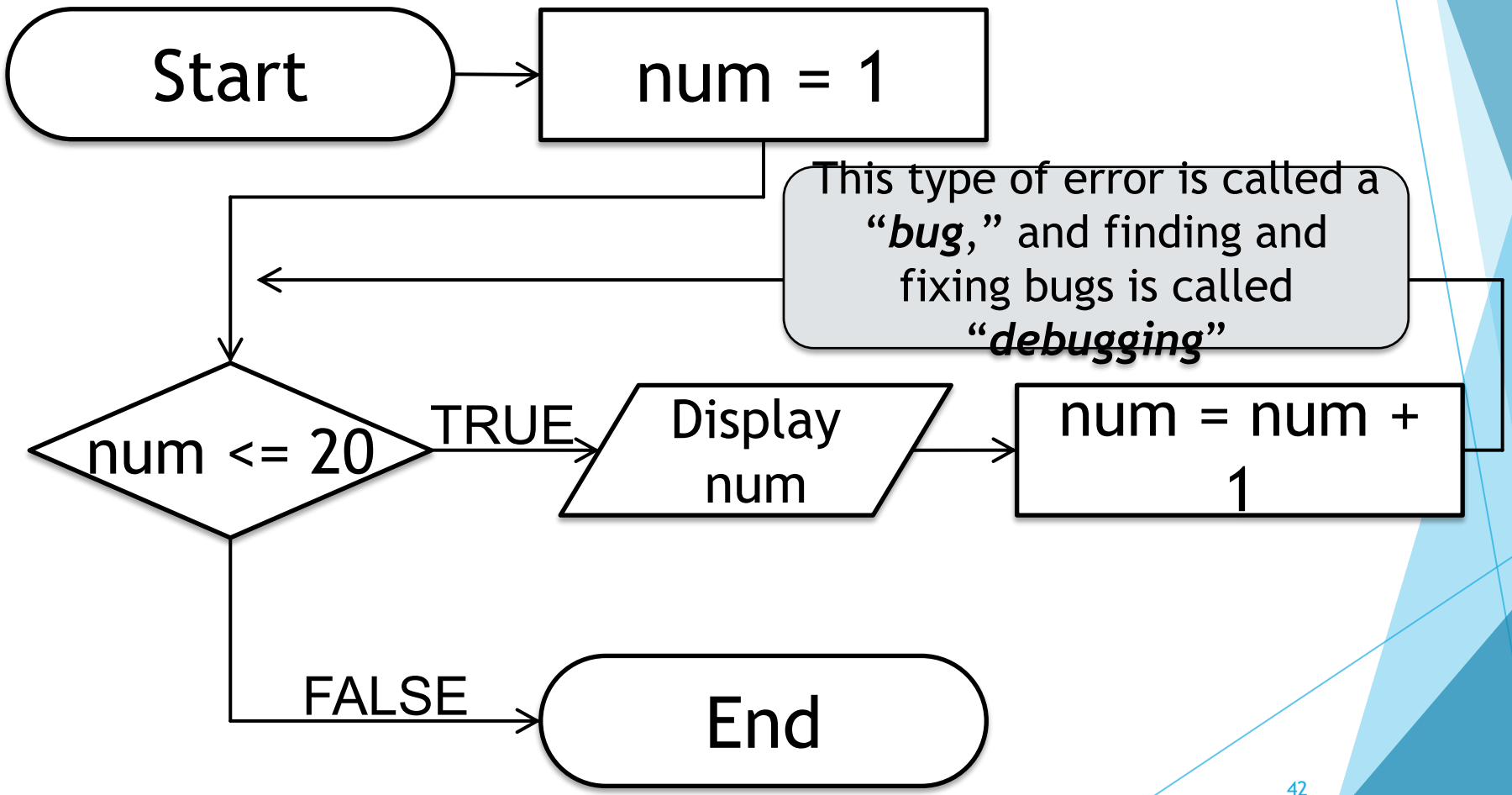
- Write an algorithm that counts from 1 to 20
  - Start with a plain English description

```
1. Set num = 1
2. While num <= 20
3.     Display num
4.     num = num + 1
5. (End loop)
```

Looping: Flowchart



Looping: Flowchart



Debugging

Errors (“Bugs”)

- Two main classifications of errors
- Syntax errors
  - Prevent C++ from understanding what to do
- Logical errors
  - Cause the program to run incorrectly, or to not do what you want

Syntax Errors

- "Syntax" is the set of rules followed by a computer programming language
  - Similar to grammar and spelling in English
- Examples of C++ syntax rules:
  - Keywords must be spelled correctly

    `cout`, float not `coutt` or `flot`
  - Quotes and parentheses must be closed:

```
("Open and close")
```

Syntax Error Examples

- Find the syntax errors in each line of code below:

```
1       coüt<<"Hello";
2       cout<<"Aloha!;
4       cout<<"Good Monring«;
```

Logical Errors

- Logical errors don't bother C++ at all… they only bother you!
- Examples of logical errors:
  - Using the wrong value for something

```
currentYear = 2013
```

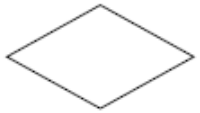
  - Doing steps in the wrong order
    - "Close jelly jar. Put jelly on bread. Open jelly jar."

Flowchart components

Beginning or end of an algorithm

Input or output of information

A computation



Decision making

The beginning of the repetition structure.

The direction of flow of the algorithm.

Circles with arrows connect the flowchart between pages.

Algorithm and flowchart example 1

- Mean of three integers

S1: Start

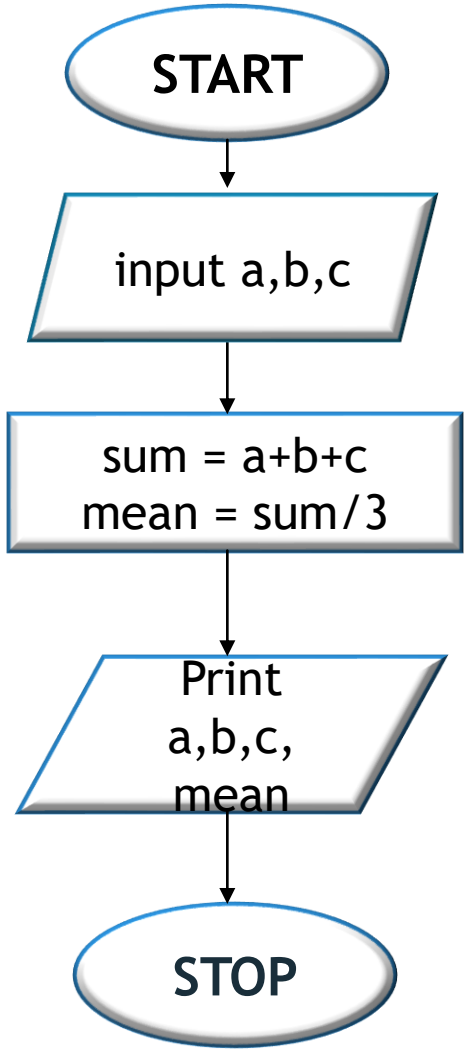
S2: Input a,b,c

S3: Set sum=a+b+c

S4: Set mean=sum/3

S5: Output a,b,c,mean

S6: End



Algorithm and flowchart example 2

- Sum of numbers 1 through 10.

S1: Start

S2: Set i=1, sum=0

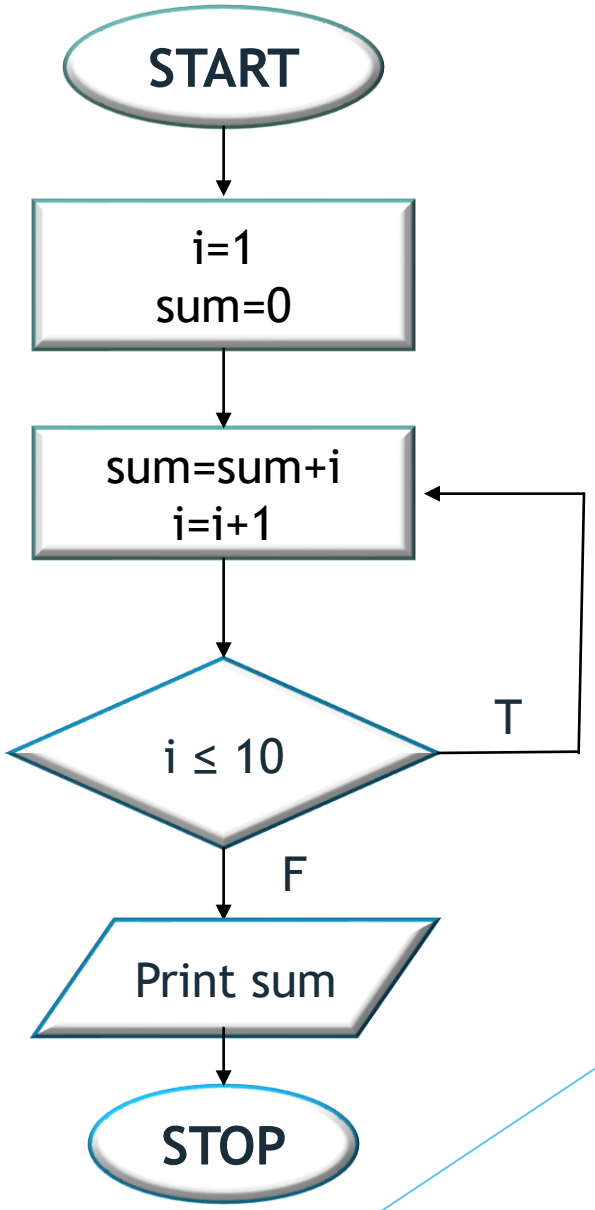
S3: sum=sum+i

S4: i=i+1

S5: if i <= 10 go to S3

S6: Output sum

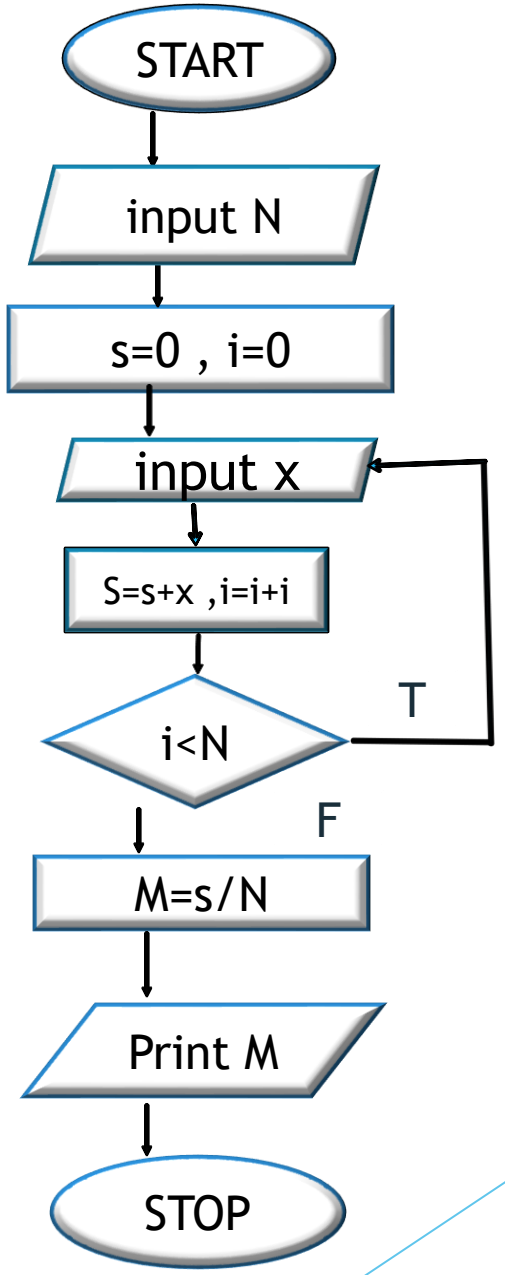
S7: End



Algorithm and flowchart example 3

- Mean of N numbers

S1: Start
S2: Input N
S3: Set s=0, i=0
S4: Input x
S5: s=s+x, i=i+1
S6: If i<N then go to S4
S7: M=s/N
S8: Output M
S9: End



Example 3: Determine Whether A Student Passed the Exam or Not:

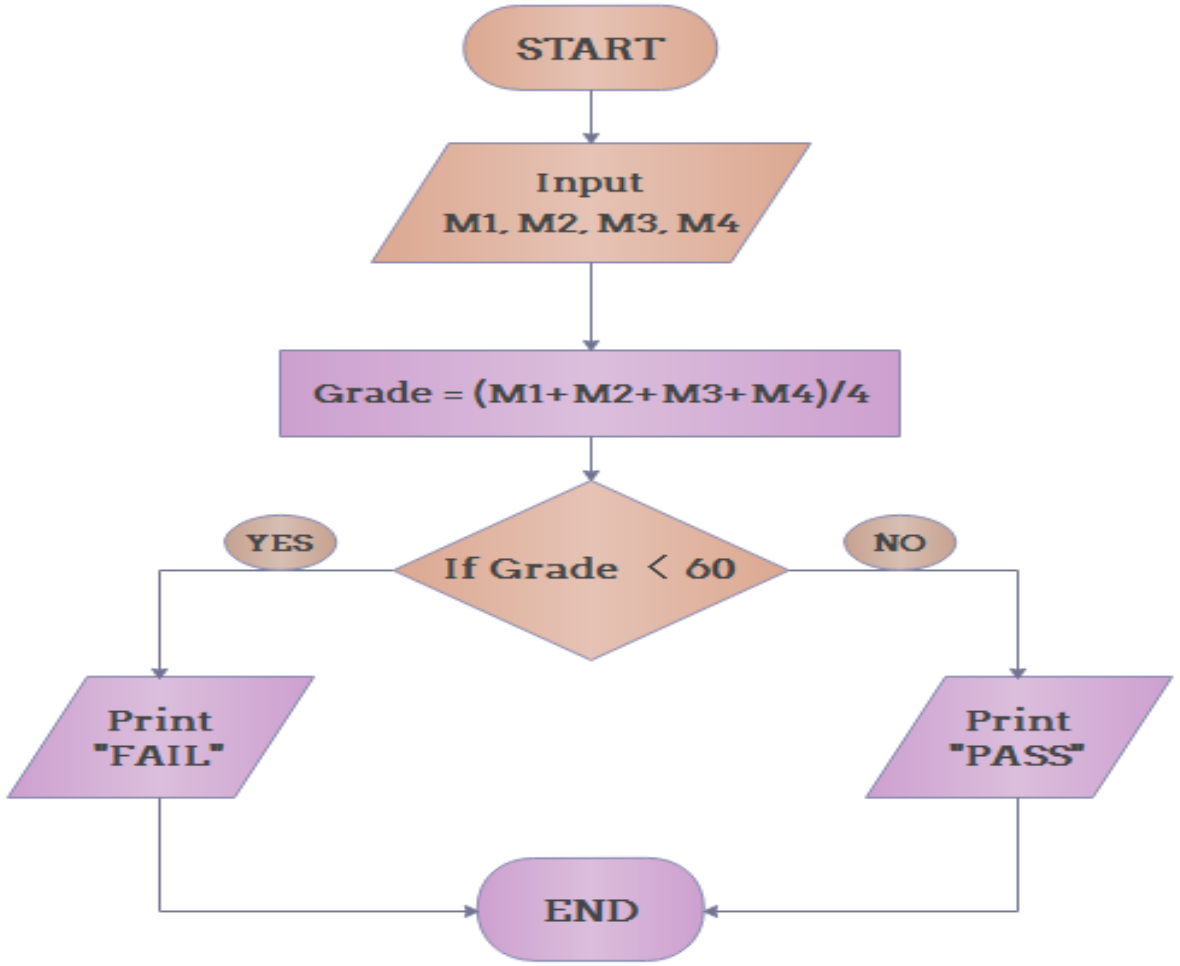
Algorithm:

Step 1: Input grades of 4 courses M1, M2, M3 and M4,

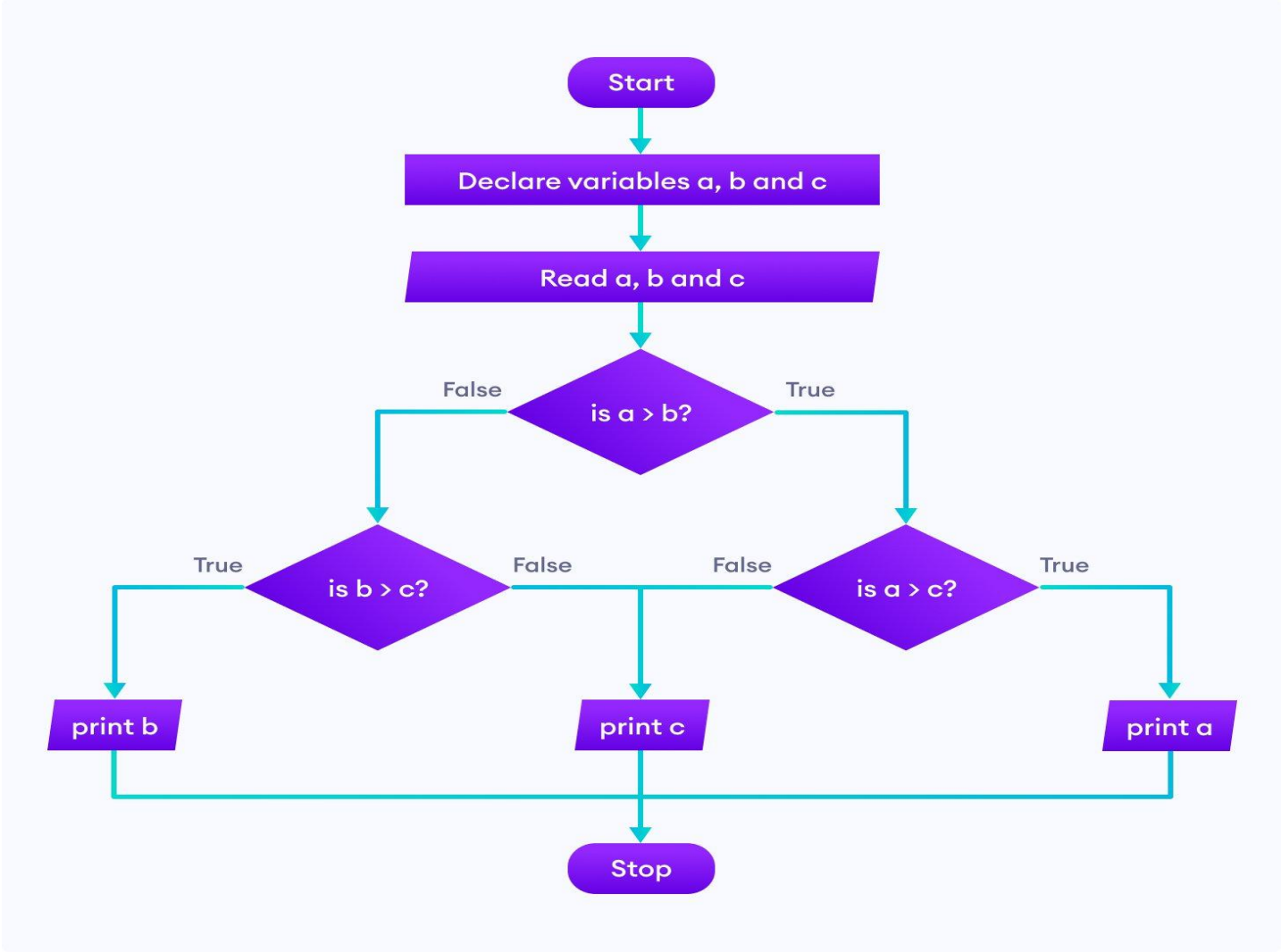
Step 2: Calculate the average grade with formula "Grade=(M1+M2+M3+M4)/4"

Step 3: If the average grade is less than 60, print "FAIL", else print "PASS".

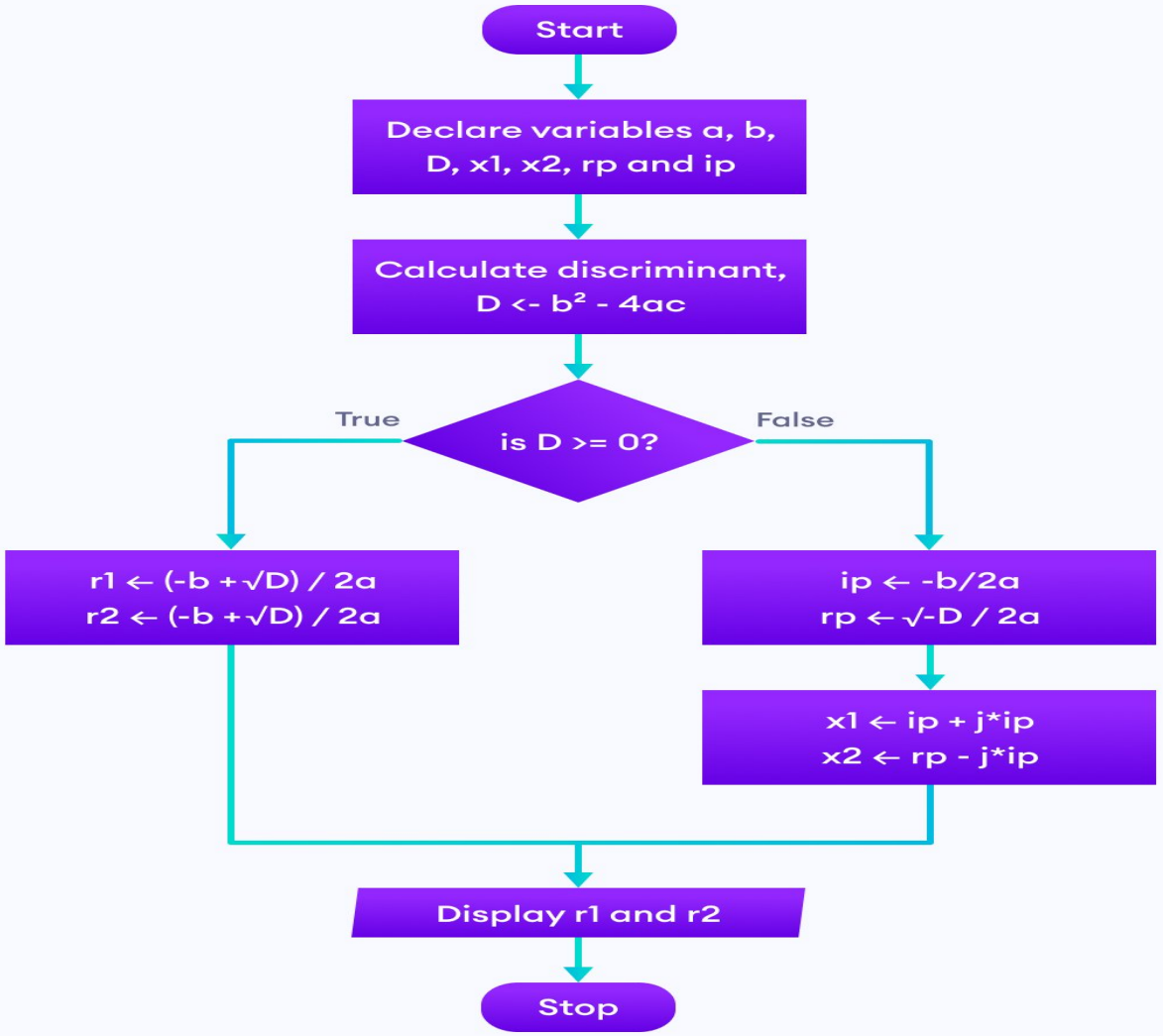
Flowchart:



Find the largest among three different numbers entered by the user.



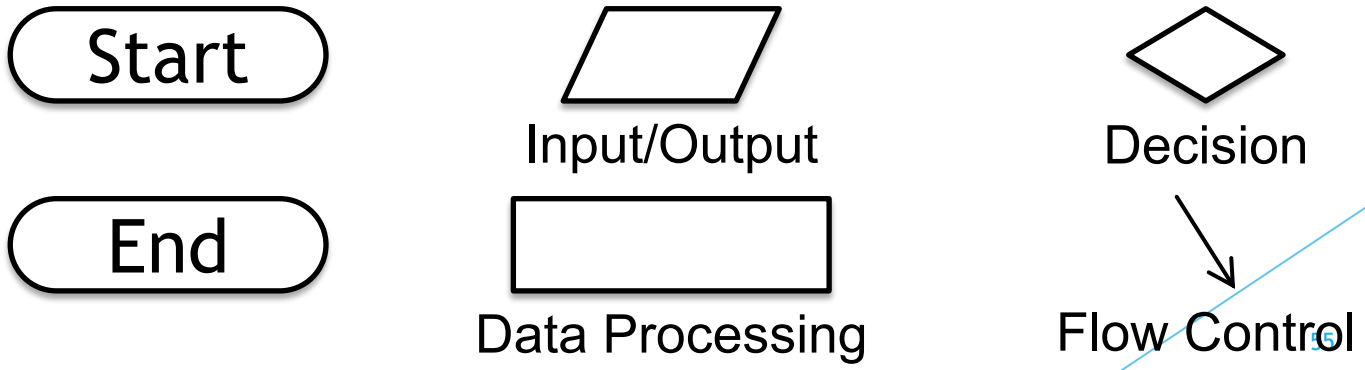
Find all the roots of a quadratic equation $ax^2+bx+c=0$



Exercise #2

- Write an algorithm that asks a user for their grade, and tells them their letter grade.

A: 100 - 90
B: <90 - 80
C: <80 - 70
D: <70 - 60
F: <60 - 0



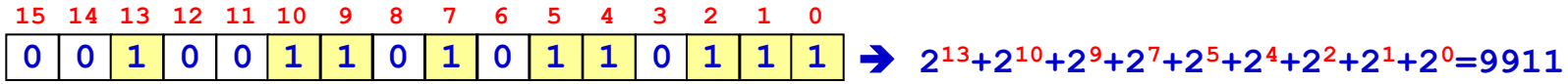
Variables

- Operations (such as addition, subtraction, etc.) operate on operands.
- You need some space to store the value of each operand.
- A variable provides storage space for a value.

Variables

- IMPORTANT: The value of a variable can never be empty. The value is represented via multiple bits, each of which is either 0 or 1. So, the variable always has a value.
- When a local variable is defined, its initial value is undefined. In other words, it has an arbitrary value. (For the moment, we will not use global variables.)
- So, make sure that the variable has a valid value before you perform any operation based on that value.

Variables



- Each variable consists of multiple bits. E.g.:
- Thus, every value is actually stored as a sequence of bits

(1s and 0s) in the computer.

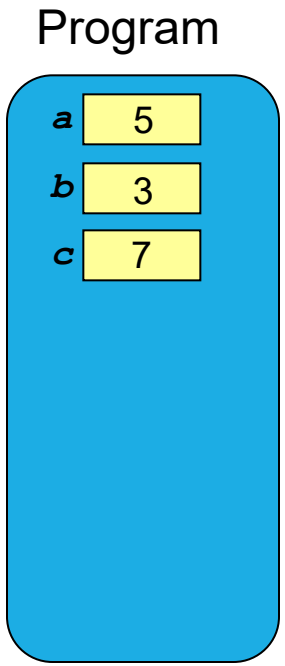
- The number of bits is called the size of the variable.
- The size of a variable depends on the type of the variable, the hardware, the operating system, and the compiler used.
  - So, in your programs NEVER make assumptions about the size of a
  - variable.

  - The size may change due to the factors listed above, and your program will not work.

Variables

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, c;

    a=10;
    b=3;
    c=a-b;
    a=b+2;
}
```



Rules for identifier names

- While defining names for variables (and also functions, user-defined types, and constants in the future) you should obey the following rules:
  - The first character of a name must be a letter or underscore ('_').
  - The remaining characters must be letters, digits, or underscore.
  - Only the first 31 characters are significant.
  - Avoid reserved words such as int, float, char, etc. as identifier names.
- However, it is better to avoid starting identifier names with underscore.
- Also remember that C++ language is case-sensitive.
- It is a very good practice to use meaningful names.

Rules for identifier names

- Valid:

```
a, a1, count, no_of_students, B56, b_56
```

- Invalid:

```
1a, sayı, int, $100
```

- Valid but not recommended:

```
_b56, Arzucan, FB, GS, BJK,
I_dont_remember_what_this_variable_means,
a_very_very_long_identifier_name_1,
a_very_very_long_identifier_name_2
```

Standard data types

- You have to specify the type of a variable when you define it.
- There are three standard data types:
  - Integer (i.e., whole numbers)
  - Float (i.e., real or floating-point numbers)
  - Characters
- We will discuss user-defined types later in the course.

Integers

- Syntax:

```
int variable_list;
```

where `variable_list` is a comma-separated list of variable names. Each variable name may be followed by an optional assignment operator and a value for initialization.

- Eg: `int a, b=10, c;`

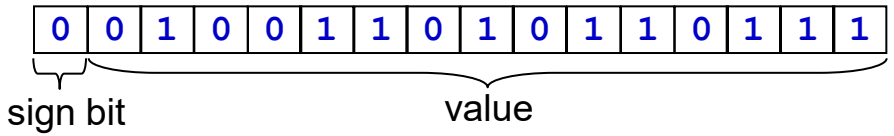
- Integer is a class of variable types. The most basic one is `int`.

- The size may change, but the leftmost bit is used for the sign. The remaining bits represent the value in binary.

- Though the size of an int variable may vary, it is always limited, i.e., it contains a limited number of bits. Therefore, the maximum and minimum values that can be represented by an int variable is limited.

Integers

- For example, assume in your system an integer has 16 bits.



- Leftmost bit is used for the sign, so 15 bits are left for the value. So, you have 2^{15}=32,768 positive values, ranging from 0 to 32,767. Similarly, you have 32,768 negative values, this time ranging from -1 to -32,768.

- If you have 32 bits (4 bytes) for an integer, than the maximum value is 2^{31}=2,147,483,647.

Integers

- There are variations of `int` such as `long int`, `short int`, `unsigned int`.
  - For each one of these types, you may ignore the word "int" and use `long`, `short`, and `unsigned`, respectively.
- The sizes of these types are ordered as follows:

  short int ≤ int ≤ long int

Floating-point numbers

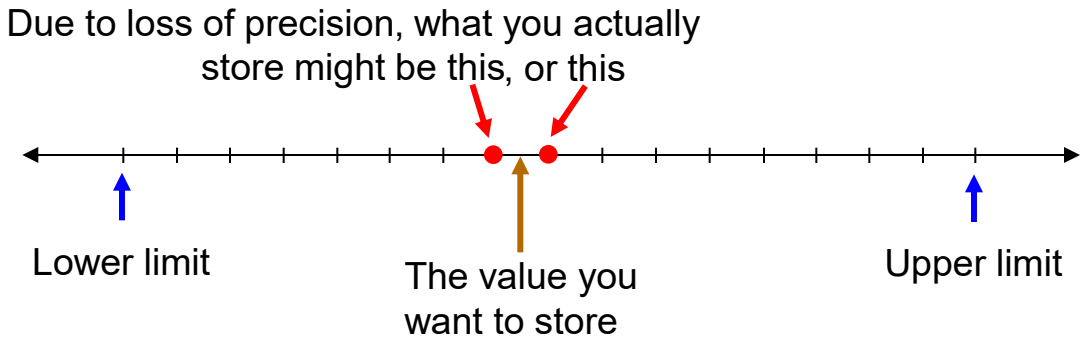
- Syntax:

```
float variable_list;
```

- Float type is used for real numbers.
- Note that all integers may be represented as floating-point numbers, but not vice versa.

Floating-point numbers

- **Similar to integers, floats also have their limits: maximum and minimum values are limited as well as the precision.**



Floating-point numbers

- There are two variations of `float`: `double` and `long double`.
  - They have wider range and higher precision.
- The sizes of these types are ordered as follows:

  float ≤ double ≤ long double

Characters

- Syntax:

```
char variable_list;
```

- Character is the only type that has a fixed size in all implementations: 1 byte.
- All letters (uppercase and lowercase, separately), digits, and symbols (such as +,-,!,?,$,£,^,#, comma itself, and many others) are of type character.

Characters

- Since every value is represented with bits (0s and 1s), we need a mapping for all these letters, digits, and symbols.
- This mapping is provided by a table of characters and their corresponding integer values.
  - The most widely used table for this purpose is the ASCII table.

Characters

- The ASCII table contains the values for 256 values (of which only the first 128 are relevant for you). Each row of the table contains one character. The row number is called the ASCII code of the corresponding character.

  (The topic of character encoding is beyond the scope of this course. So, we will work with the simplified definition here.)

ASCII table (partial)

ASCII code	Symbol	ASCII code	Symbol	ASCII code	Symbol	ASCII code	Symbol
...	...	66	B	84	T	107	k
32	blank	67	C	85	U	108	l
37	%	68	D	86	V	109	m
42	*	69	E	87	W	110	n
43	+	70	F	88	X	111	o
...	...	71	G	89	Y	112	p
48	0	72	H	90	Z	113	q
49	1	73	I	...	...	114	r
50	2	74	J	97	a	115	s
51	3	75	K	98	b	116	t
52	4	76	L	99	c	117	u
53	5	77	M	100	d	118	v
54	6	78	N	101	e	119	w
55	7	79	O	102	f	120	x
56	8	80	P	103	g	121	y
57	9	81	Q	104	h	122	z
...	...	82	R	105	i	...	...
65	A	83	S	106	j		

Characters

- Never memorize the ASCII codes. They are available in all programming books and the Internet. (Eg: http://www.ascii-code.com)
- What is important for us is the following three rules:
  - All lowercase letters (a,b,c,...) are consecutive.
  - All uppercase letters (A,B,C,...) are consecutive.
  - All digits are consecutive.

Characters

- Note that `a` and `A` have different ASCII codes (`97` and `65`).
- You could also have a variable with name `a`. To differentiate between the variable and the character, we specify all characters in single quotes, such as `'a'`. Variable names are never given in quotes.
  - Example:
```
char ch;
ch='a';
```
- Note that using double quotes makes it a string (to be discussed later in the course) rather than a character. Thus, `'a'` and `"a"` are different.
- Similarly, `1` and `'1'` are different. Former has the value `1`, whereas the latter has the ASCII value of `49`.

Characters

- A character variable actually stores the ASCII value of the corresponding letter, digit, or symbol.
- I/O functions (cin, cout, etc.) do the translation between the image of a character displayed on the screen and the ASCII code that is actually stored in the memory of the computer.

The table shows the fundamental data types in C++, as well as the range of values.

Tablo 2.1: *Fundamental data types and their size and ranges in the memory. The numbers are evaluated for a 32-bit system.*

Data Type	Description	Size (byte)	Lower Limit	Upper Limit
`char`	Character or small integer	1	-128	127
`unsigned char`			0	255
`short int`	Short integer	2	-32,768	32,767
`unsigned short int`			0	65,535
`int`	integer	4	-2,147,483,648	2,147,483,647
`unsigned int`			0	4,294,967,295
`long int`	Long integer	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
`unsigned long int`			0	18,446,744,073,709,551,615
`float`	Single precision floating point number (7 digits)	4	-3.4e +/- 38	+3.4e +/- 38
`double`	Double precision floating point number (15 digits)	8	-1.7e +/- 308	+1.7e +/- 308
`long double`	Quad precision floating point number (34 digits) [*]	16	-1.0e +/- 4931	+1.0e +/- 4931

[*] only on 64 bit platforms.

Note that the unqualified **`char`**, **`short`**, **`int`**, (**`long int`**) are **`signed`** by default. And **`unsigned`** integers are always positive and so have a larger positive range.

sizeof() operator

The unary operator **`sizeof()`** is used to calculate the size in bytes of data types, variables, arrays or any object.

The following code

```
int i;
double d;
cout << "sizeof(int)    = " << sizeof(int)    << " bytes" << endl;
cout << "sizeof(float)  = " << sizeof(float)  << " bytes" << endl;
cout << "sizeof(double)= " << sizeof(double)<< " bytes" << endl;
cout << "sizeof(i)      = " << sizeof(i)      << " bytes" << endl;
cout << "sizeof(d)      = " << sizeof(d)      << " bytes" << endl;
```

will output

```
sizeof(int)    = 4 bytes
sizeof(float)  = 4 bytes
sizeof(double)= 8 bytes
sizeof(i)      = 4 bytes
sizeof(d)      = 8 bytes
```

Constants

- Syntax:

```
#define constant_name constant_value
```

- As the name implies, variables store values that vary while constants represent fixed values.
- Note that there is no storage when you use constants. Actually, when you compile your program, the compiler replaces the constant name with the value you defined.
- The pre-processor replaces every occurrence of `constant_name` with everything that is to the right of `constant_name` in the definition.
  - Note that there is no semicolon at the end of the definition.
- Conventionally, we use names in uppercase for constants.

Example

```
#include <iostream>
using namespace std;

#define CURRENTYEAR 2014

int main(){
    int  year, age;
    char myName;

   cout<<"Enter the year you were born and your initial \n";
   cin>> year >> myName;
   cout<<"Your initial is:  " <<myName ;
   age = CURRENTYEAR - year;
   cout<<"Your age is: "<<age;

   return 0;
}
```

Enumerated type

- Used to define your own types.
- Syntax:

Text in green is optional

```
enum type_name {
    item_name=constant_int_value, ...
} variable_list;
```

- By default, the value of the first item is 0, and it increases by one for consecutive items. However, you may change the default value by specifying the constant value explicitly.
- Eg:

```
enum boolean {FALSE,TRUE} v1, v2;
enum days {SUN,MON,TUE,WED,THU,FRI,SAT};
enum {one=1,five=5,six,seven,ten=10,eleven} num;
enum months
{JAN=1,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC};
```

Operators

- We will cover the most basic operators in class. More operators will be covered in the labs.
- Assignment operator (=)
  - Note that this is not the "equals" operator. It should be pronounced as "becomes." (Equals is another operator.)
  - The value of the expression on the RHS is assigned (copied) to the LHS.
  - It has right-to-left associativity.

```
a=b=c=10;
```

makes all three variables `10`.

Assignment and type conversion

- When a variable of a narrower type is assigned to a variable of wider type, no problem.
  - Eg:
```
int a=10;    float f;
             f=a;
```
- However, there is loss of information in reverse direction.
  - Eg:
```
float f=10.9; int a;
              a=f;
```

Operators

- **Arithmetic operators (+,-,*,/,%)**
  - **General meanings are obvious.**
  - **What is important is the following: If one of the operands is of a wider type, the result is also of that type. (Its importance will be more obvious soon.)**
    - Eg: Result of int+float is float. Result of float+double is double.
  - **In C++ language, there are two types of division: integer division and float division.**
    - **If both operands are of integer class, we perform integer division and the result is obtained by truncating the decimal part.**
      - Eg: `8/3` is `2`, not `2.666667`.
    - **If one of the operands is of float class, the result is float.**
      - Eg: `8.0/3` or `8/3.0` or `8.0/3.0` is `2.666667`, not `2`.

Operators

- Remainder operator is `%`. Both operands must be of integer class.
  - Eg: `10%6` is `4` (equivalent to `10 mod 6`)
- `+,-,*,/,%` have left-to-right associativity. That means `a/b/c` is equivalent to `(a/b)/c`, but not `a/(b/c)`.

Operators

- **Logic operators (&&, ||, !)**
  - Logic operators take integer class operands.
    - Zero means false.
    - Anything non-zero means true.
  - "&&" does a logical-AND operation. (True only if both operands are true.)
  - "||" does a logical-OR operation. (False only if both operands are false.)
  - "!" does a negation operation. (Converts true to false, and false to true.)

Operators

- Logic operators follow the logic rules

a	b	a && b	a || b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

- The order of evaluation is from left to right
- As usual parenthesis overrides default order

Operators

- If the first operand of the "`&&`" operator is false, the second operand is not evaluated at all (since it is obvious that the whole expression is false).
  - Eg: In the expression below, if the values of `b` and `c` are initially `0` and `1`, respectively,

    ```
    a = b && (c=2)
    ```

    then the second operand is not evaluated at all, so `c` keeps its value as `1`.
- Similarly, if the first operand of the "`||`" operator is true, the second operand is not evaluated at all.

Operators

- Other assignment operators (+=, -=, *=, /=, %=)
  - Instead of writing `a=a+b`, you can write `a+=b` in short. Similar with –=, *=, /=, and others.

Operators

- Pre/Post increment/decrement operators (++, --)
  - The operator ++ increments the value of the operand by 1.
    - If the operator comes BEFORE the variable name, the value of the variable is incremented before being used, i.e., the value of the expression is the incremented value. This is pre-increment.
    - In post-increment, the operator is used after the variable name, and incrementation is performed after the value is used, i.e., the value of the expression is the value of the variable before incrementation.

Operators

Ex: `a=10;` `c=10,`

`b=++a;` `d=c++;`

Both `a` and `c` will become `11`, but `b` will be `11` while `d` is `10`.

Ex:

	x	y
`int x=10, y=20;`	10	20
`++x;`	11	20
`y= --x;`	10	10
`x= x-- +y;`	19	10
`y= x - ++x;`	20	0

Operators

- Comparison operators (==,!=,<,<=,...)
  - "==" is the "is equal to" operator. Like all other comparison operators, it evaluates to a Boolean value of true or false, no matter what the operand types are.
  - IMPORTANT: When you compare two float values that are supposed to be equal mathematically, the comparison may fail due to the loss of precision discussed before.

Operators

Symbol	Usage	Meaning
==	x == y	is x equal to y?
!=	x != y	is x not equal to y?

>	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?

Operators

- We can create complex expressions by joining several expressions with logic operators.

Symbol	Usage	Meaning
&&	exp1 && exp2	AND
||	exp1 || exp2	OR
!	! exp	NOT

Operators

- While using multiple operators in the same expression, you should be careful with the precedence and associativity of the operands.
  - Eg: The following does NOT check if `a` is between `5` and `10`.

```
bool = 5<a<10;
```

    - `bool` will be true if `a` is `20`. (Why?)
  - Don't hesitate to use parentheses when you are not sure about the precedence (or to make things explicit).

Operator precedence table

Operator	Associativity
() [] . ->	left-to-right
++ -- + - ! ~ (type) * & sizeof	right-to-left
* / %	left-to-right
+ -	left-to-right
<< >>	left-to-right
< <= > >=	left-to-right
== !=	left-to-right
&	left-to-right
^	left-to-right
|	left-to-right
&&	left-to-right
||	left-to-right
?:	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	right-to-left
,	left-to-right

Operators

- Precedence, associativity, and order of evaluation:
  - In the table is given in the previous slide, precedence decreases as you go down.
  - If two operands in an expression have the same precedence, you decide according to the associativity column.
  - There is a common misunderstanding about associativity.
    - Note that associativity has nothing to do with the order of evaluation of the operands.
    - Order of evaluation of operands is not specified in C++ language.

Type casting

- Also called *coersion* or *type conversion*.
- It does NOT change the type of a variable. It is not possible to change the type of a variable.
- What casting does is to convert the type of a value.

Type casting

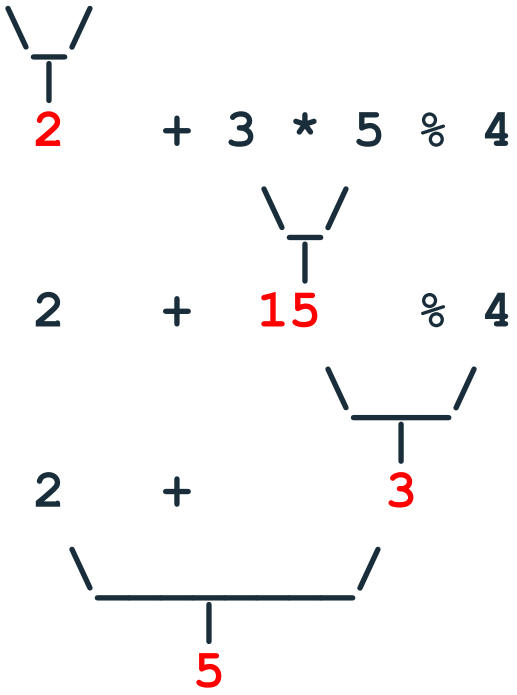
- Eg:
```
int a=10, b=3;
float f, g;
f=a/b;
g=(float)a/b;
```

- The type of `a` does not change; it is still an integer. However, in the expression `(float)a/b`, the value of `a`, which is `10`, is converted to float value of `10.0`, and then it is divided by `b`, which is `3`. Thus, we perform float division and `g` becomes `3.3333...`

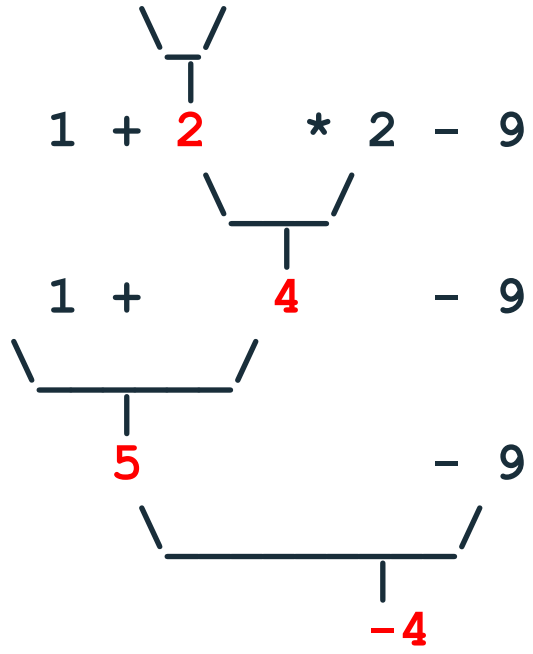
- On the other hand, we perform an integer division for `f`, so it becomes `3`.

Precedence examples

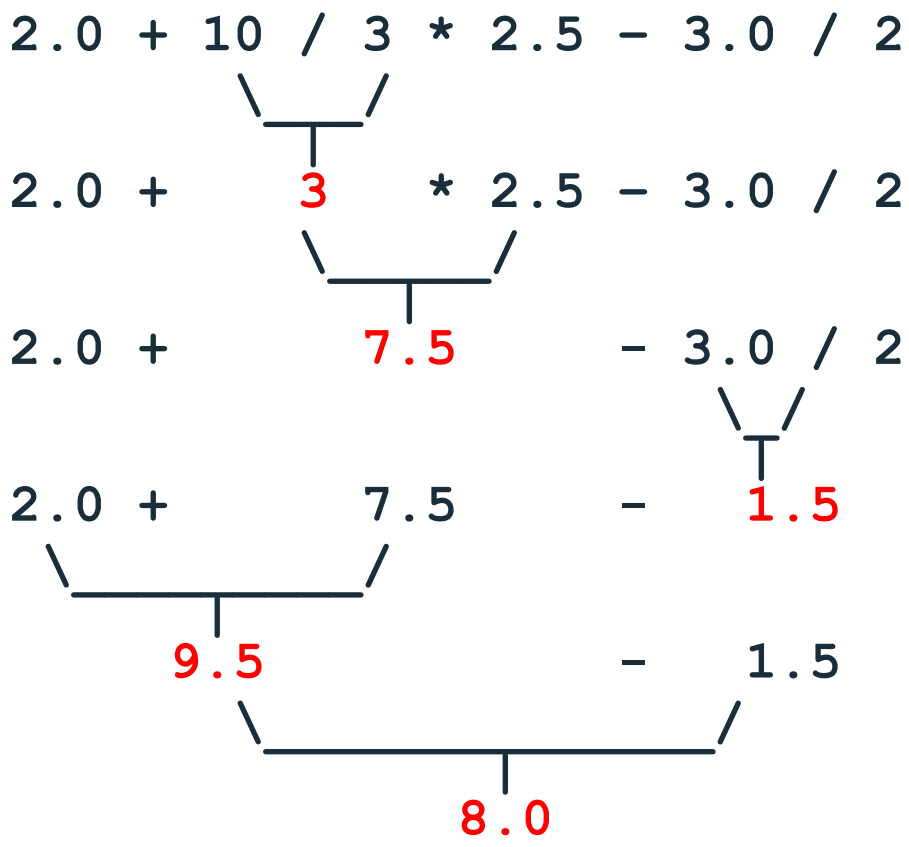
- 1 * 2 + 3 * 5 % 4



1 + 8 % 3 * 2 - 9



Mixing types



Basic intrinsic functions

An intrinsic or a library function is a function provided by C++ language. For example the cmath library contains the mathematical functions/constants.

Some C++ library mathematical functions and constants defined in <cmath>

Function Decleration	Description	Example	Result
`double fabs(double x);`	absolute value of real number, |x|	`fabs(-4.0)`	4.0
`int floor(double x);`	round down to an integer	`floor(-2.7)`	-3
`int ceil(double x);`	round up to an integer	`ceil(-2.7)`	-2
`double sqrt(double x);`	square root of x	`sqrt(4.0)`	2.0
`double pow(double x, double y);`	the value of x^y	`pow(2., 3.)`	8.0
`double exp(double x);`	the value of e^x	`exp(2.0)`	7.38906
`double log(double x);`	natural logarithm, $\log_e x = \ln x$	`log(4.0)`	1.386294
`double log10(double x);`	base 10 logarithm, $\log_{10} x = \log x$	`log10(4.0)`	0.602060
`double sin(double x);`	sinus of x (x is in radian)	`sin(3.14)`	0.001593
`double cos(double x);`	cosine of x (x is in radian)	`cos(3.14)`	-0.999999
`double tan(double x);`	tangent of x (x is in radian)	`tan(3.14)`	-0.001593
`double asin(double x);`	arc-sine of x in the range [-pi/2, pi/2]	`asin(0.5)`	0.523599
`double acos(double x);`	arc-cosine of x in the range [-pi/2, pi/2]	`acos(0.5)`	1.047198
`double atan(double x);`	arc-tangent of x in the range [-pi/2, pi/2]	`atan(0.5)`	0.463648
`M_PI`	constant pi	`myPI = M_PI`	3.141592...
`M_E`	constant e	`x = M_E`	2.718281...

Some standard C++ library functions and constant defined in <cstdlib>

Function Decleration	Description	Example	Result
`int abs(int x);`	absolute value of integer number, |x|	`abs(-4)`	4
`int atoi(const char *s);`	converts string to integer	`atoi("-1234")`	-1234
`double atof(const char *s);`	converts a string to double	`atof("123.54")`	123.54
`void exit(int status);`	terminates the calling process "immediately"	`exit(1)`	-
`int rand(void);`	Returns a random integer between 0 and RAND_MAX	`rand()`	1048513214
`RAND_MAX`	The largest number rand() will return	`x = RAND_MAX`	2147483647