2.1 Introduction

- Before writing a program:
  - Have a thorough understanding of problem
  - Carefully plan your approach for solving it
- While writing a program:
  - Know what “building blocks” are available
  - Use good programming principles

2.2 Algorithms

- All computing problems
  - can be solved by executing a series of actions in a specific order
- Algorithm
  - A procedure determining the
    - Actions to be executed
    - Order in which these actions are to be executed
- Program control
  - Specifies the order in which statements are to executed

2.3 Pseudocode

- Pseudocode
  - Artificial, informal language used to develop algorithms
  - Similar to everyday English
  - Not actually executed on computers
  - Allows us to “think out” a program before writing the code for it
  - Easy to convert into a corresponding C++ program
  - Consists only of executable statements

2.4 Control Structures

- Sequential execution
  - Statements executed one after the other in the order written
- Transfer of control
  - When the next statement executed is not the next one in sequence
- Bohm and Jacopini: all programs written in terms of 3 control structures
  - Sequence structure
    - Built into C++. Programs executed sequentially by default.
  - Selection structures
    - C++ has three types - **`if`**, **`if/else`**, and **`switch`**
  - Repetition structures
    - C++ has three types - **`while`**, **`do/while`**, and **`for`**

2.4 Control Structures

- C++ keywords
  - Cannot be used as identifiers or variable names.

C++ Keywords				
Keywords common to the C and C++ programming languages				
auto	**break**	**case**	**char**	**const**
continue	**default**	**do**	**double**	**else**
enum	**extern**	**float**	**for**	**goto**
if	**int**	**long**	**register**	**return**
short	**signed**	**sizeof**	**static**	**struct**
switch	**typedef**	**union**	**unsigned**	**void**
volatile	**while**			
C++ only keywords				
asm	**bool**	**catch**	**class**	**const_cast**
delete	**dynamic_cast**	**explicit**	**false**	**friend**
inline	**mutable**	**namespace**	**new**	**operator**
private	**protected**	**public**	**reinterpret_cast**	
static_cast	**template**	**this**	**throw**	**true**
try	**typeid**	**typename**	**using**	**virtual**
wchar_t				

2.4 Control Structures

- Flowchart
  - Graphical representation of an algorithm
  - Drawn using certain special-purpose symbols connected by arrows called flowlines.
  - Rectangle symbol (action symbol)
    - Indicates any type of action.
  - Oval symbol
    - indicates beginning or end of a program, or a section of code (circles).
- single-entry/single-exit control structures
  - Connect exit point of one control structure to entry point of the next (control-structure stacking).
  - Makes programs easy to build.

2.5 The if Selection Structure

- Selection structure
  - used to choose among alternative courses of action
  - Pseudocode example:

    If student's grade is greater than or equal to 60

    Print "Passed"
  - If the condition is **`true`**
    - print statement executed and program goes on to next statement
  - If the condition is **`false`**
    - print statement is ignored and the program goes onto the next statement
  - Indenting makes programs easier to read
    - C++ ignores whitespace characters

2.5 The if Selection Structure

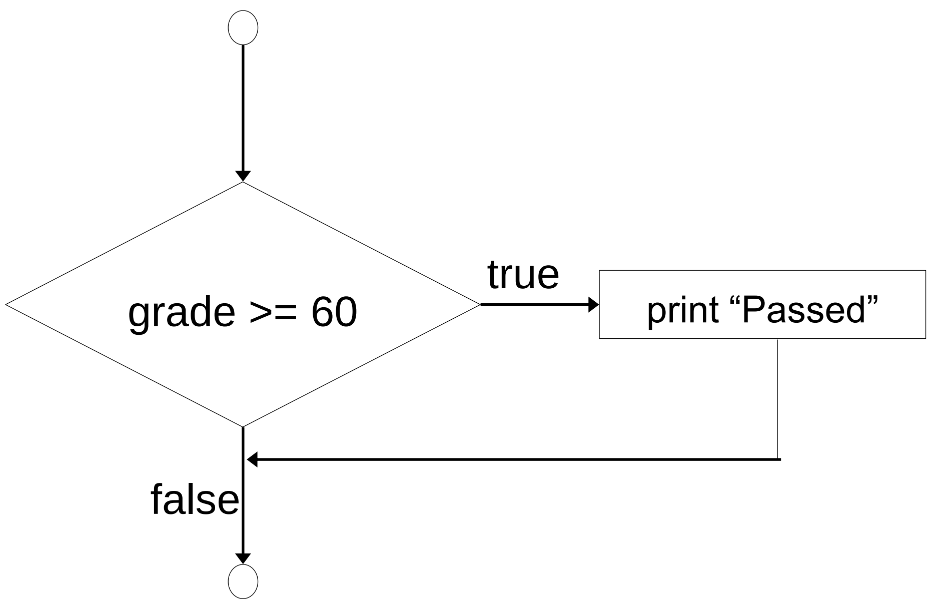
- Translation of pseudocode statement into C++:

```
if ( grade >= 60 )
   cout << "Passed";
```

- Diamond symbol (decision symbol)
  - indicates decision is to be made
  - Contains an expression that can be true or false.
    - Test the condition, follow appropriate path
- **`if`** structure is a single-entry/single-exit structure

2.5 The if Selection Structure

- Flowchart of pseudocode statement



A decision can be made on any expression.

zero - **`false`**

nonzero - **`true`**

Example:

`3 - 4` is **`true`**

2.6 The `if/else` Selection Structure

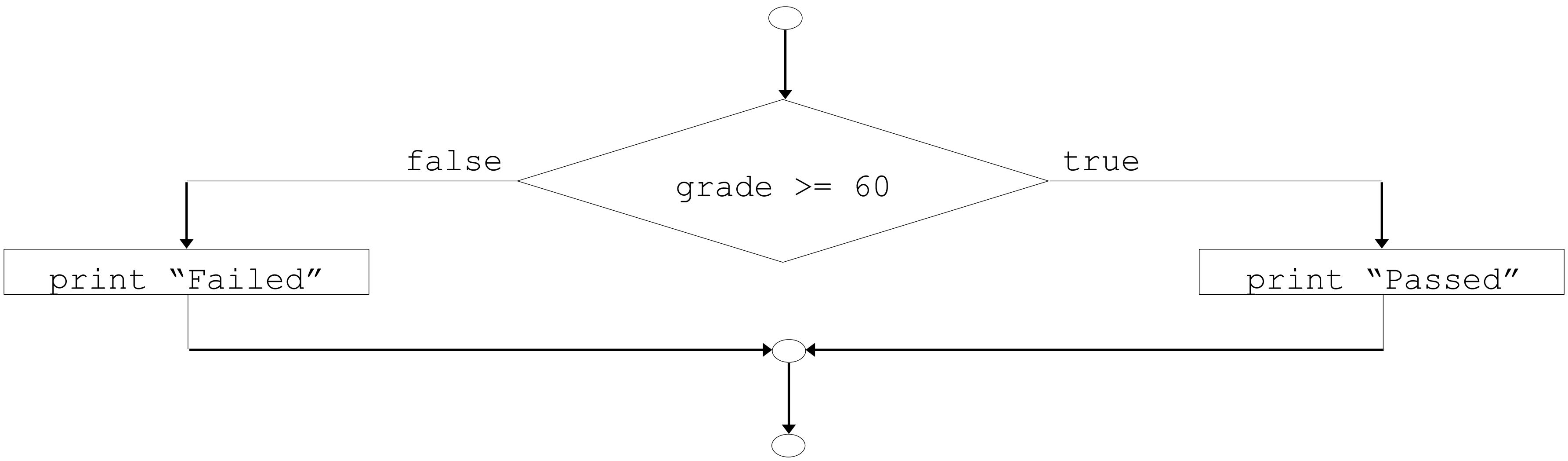
- **`if`**
  - Only performs an action if the condition is true
- **`if/else`**
  - A different action is performed when condition is true and when condition is false
- Psuedocode

if student's grade is greater than or equal to 60
print "Passed"
else
print "Failed"

- C++ code

```
if ( grade >= 60 )
   cout << "Passed";
else
   cout << "Failed";
```

2.6 The `if/else` Selection Structure



- Ternary conditional operator (**`?:`**)
  - Takes three arguments (condition, value if **`true`**, value if **`false`**)
- Our pseudocode could be written:

```
cout << ( grade >= 60 ? "Passed" : "Failed" );
```

٢,٦ The `if/else` Selection Structure

- Nested **`if/else`** structures
  - Test for multiple cases by placing **`if/else`** selection structures inside **`if/else`** selection structures.

if student's grade is greater than or equal to 90
Print "A"
else
if student's grade is greater than or equal to 80
Print "B"
else
if student's grade is greater than or equal to 70
Print "C"
else
if student's grade is greater than or equal to 60
Print "D"
else
Print "F"

  - Once a condition is met, the rest of the statements are skipped

٢,٦ The if/else Selection Structure

- Compound statement:
  - Set of statements within a pair of braces
  - Example:

```
if ( grade >= 60 )
   cout << "Passed.\n";
else {
   cout << "Failed.\n";
   cout << "You must take this course
again.\n";
}
```

  - Without the braces,

```
cout << "You must take this course again.\n";
```

  would be automatically executed

- Block
  - Compound statements with declarations

٢,٦ The if/else Selection Structure

- Syntax errors
  - Errors caught by compiler
- Logic errors
  - Errors which have their effect at execution time
    - Non-fatal logic errors
      - program runs, but has incorrect output
    - Fatal logic errors
      - program exits prematurely

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`

- Rules of operator precedence:

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.

Operator precedence

- How does we evaluate `1 + 3 * 4`?
  Is it `(1 + 3) * 4`, or is it `1 + (3 * 4)`?
  - In a complex expression with several operators, Java uses internal rules of *precedence* to decide the order in which to apply the operators.

- **precedence**: Order in which operations are computed in an expression.
  - Multiplicative operators have a higher level of precedence than additive operators, so they are evaluated first.
    - `* / %` before `+ -`
  - In our example, * has higher precedence than +, just like on a scientific calculator, so `1 + 3 * 4` is `13`.
  - Parentheses can be used to force a certain precedence. `(1 + 3) * 4` is `16`.

Precedence examples

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)*

Precedence examples

```
1 * 2 + 3 * 5 / 4
 \_/
  |
  2   + 3 * 5 / 4
         \_/
          |
  2   +  15   / 4
          \___/
            |
  2   +     3
   \________/
       |
       5
```

```
1 + 2 / 3 * 5 - 4
     \_/
      |
1 +   0   * 5 - 4
       \___/
         |
1 +      0    - 4
 \_______/
    |
    1         - 4
     \________/
         |
        -3
```

Precedence examples

- What values result from the following expressions?
  - `9 / 5`
  - `695 % 20`
  - `7 + 6 * 5`
  - `7 * 6 + 5`
  - `248 % 100 / 5`
  - `6 * 3 - 9 / 4`
  - `(5 - 7) * 4`
  - `6 + (18 % (17 - 12))`
- Which parentheses above are unnecessary (which do not change the order of evaluation?)

Real numbers

- The expressions we have seen so far used integers, but C also can manipulate real numbers (numbers with a decimal point).
  - Examples: `6.022` `-15.9997` `42.0` `2.143e17`
- The operators we saw, `+ - * / %` , as well as parentheses `( )` , all work for real numbers as well.
  - The `/` operator produces an exact answer when used on real numbers, rather than an integer quotient.
    - Example: `15.0 / 2.0` is `7.5`
  - The `%` operator is not often used on real numbers.
- The same rules of precedence that apply to integers also apply to real numbers.
  - `( )` before `* / %` before `+ -`

Real number example

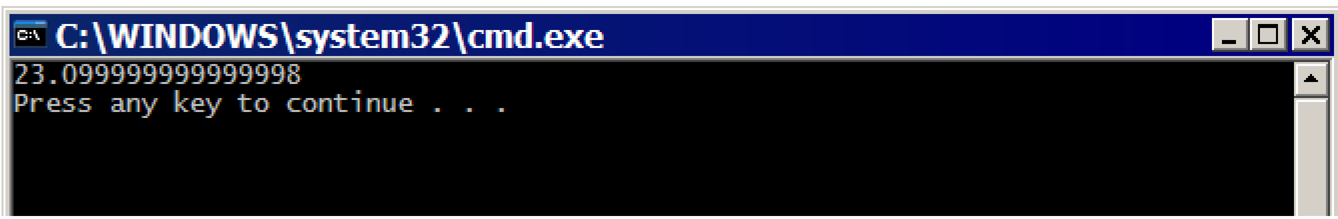
```
1.5 * 2.4 + 3.3 * 4.25 / 5.5
   \_/
    |
   3.6    + 3.3 * 4.25 / 5.5
                 \_/
                  |
   3.6    +   14.025   / 5.5
                    \___/
                      |
   3.6    +          2.55
       \____________/
             |
            6.15
```

Real number precision

- Strange things are afoot with real numbers:

```
Cout << ((35.0 + 22.4 + 11.9) / 3.0);
```

  – The mathematically correct answer should be 23.1
  – Instead, we get this:



- Unfortunately, the computer represents real numbers in an imprecise way internally, so some calculations with them are off by a very slight amount.
  – We cannot do anything to change this.
  – We will generally ignore this problem for this course and tolerate the precision errors, but later on we will learn some ways to produce a better output for examples like above.

Mixing integers and reals

- When a Java operator is used on an integer and a real number, the result is a real number.
  - Example: `3 * 4.2` is `12.6`
  - Example: `1 + 1.0` is `2.0`

- The kind of number that results from a given operator depends only on its operands, not any other operands.

```
7 / 3 * 1.2 + 3 / 2
 \___/
   |
   2    * 1.2 + 3 / 2
    \_______/
        |
       2.4      + 3 / 2
                   \___/
                     |
       2.4      +    1
        \____________/
               |
              3.4
```