COMP1011
Programming Fundamentals

Lecture 2
Control Structures I

Lecture 2

- Algorithm
- Pseudocode
- Introduction to Control Structures
- Selection Structures
  - **`if`** and **`if/else`**
  - Nested **`if/else`**
- Ternary conditional operator
- More operators
  - Assignment Operators
  - Increment and Decrement Operators
  - Logical Operators
- Selection Structures
  - **`switch`**

Algorithm

- Remember the purpose of a program?
  - To solve a computational problem
- Well-defined steps
  - Input → Processing → Output
- Some problems can be generalized
  - E.g., sorting a list of numbers in ascending order, regardless of how many numbers
- General problem-solving method
  - E.g., the same algorithm can be used to sort a list of numbers or a list of names
- Termination and Correctness

Pseudocode

- Artificial, informal language used to develop algorithms
- Similar to everyday English
- Not executed on computers
  - Used to think out the logic of program before coding
    - Easy to convert into C++ syntax
  - Only executable statements
    - No need to declare variables

Pseudocode

› For example,

Write a program that accepts two integers, calculate and display the sum.

› Write the pseudocode:

Prompt the user to input integer1
Prompt the user to input integer2
Calculate sum = integer1 + integer2
Print sum to the screen

Pseudocode

› C++ Implementation

pseudo.cpp

```cpp
#include <iostream>

using namespace std;

int main() {

    int integer1;
    int integer2;
    int sum;

    cout << "Please enter the first integer: ";
    cin >> integer1;

    cout << "Please enter the second integer: ";
    cin >> integer2;

    sum = integer1 + integer2;
    cout << "The sum is " << sum << endl;

    return 0;
}
```

```
Please enter the first integer: 10
Please enter the second integer: 20
The sum is 30
```

Pseudocode

- Another Example

  Write a program to calculate the average score in an examination of three students.

- Pseudocode

Prompt the user to input score1
Prompt the user to input score2
Prompt the user to input score3
Calculate sum = score1 + score2 + score3
Calculate average = sum / 3
Print average to the screen

pseudo2.cpp

```cpp
#include <iostream>

using namespace std;

int main() {

    int score1, score2, score3, sum;
    double average;

    cout << "Please enter the score of student 1: ";
    cin >> score1;

    cout << "Please enter the score of student 2: ";
    cin >> score2;

    cout << "Please enter the score of student 3: ";
    cin >> score3;

    sum = score1 + score2 + score3;
    average = sum / 3.0;
    cout << "The average is " << average << endl;

    return 0;
}
```

```
Please enter the score of student 1: 90
Please enter the score of student 2: 58
Please enter the score of student 3: 34
The average is 60.6667
```

Pseudocode

- More or less like a real computer program: assignment statements, arithmetic expressions, control structures (if, if/else, while, for, etc.), and so on
- But no strict rules for the syntax
- OK as long as it is clear, readable and understandable
- Enables you to concentrate on the algorithm instead of details of syntax

Hints on Writing Pseudocode

- Always think about the sequence

  Input ⇨ Processing ⇨ Output

- Input
  - What data does the user need to provide to the program?
  - E.g., an integer value? student's scores?
- Processing
  - What does the program calculate?
  - E.g., calculating the sum? calculating the average?
- Output
  - What does the program display to the user?
  - E.g., calculation result? sum? average?

Exercise 1

- Soft drinks are sold in cans and bottles
  - 1 bottle of soft drink is 2 liters
  - 1 can of soft drink is 12 ounce (1 ounce = 0.0296L)
- Develop a program to calculate the total volume (in liters) of the soft drinks
  - The number of cans and bottles of the soft drinks are input by the user
- Sample input and output:

```
Please enter the number of bottles: 2
Please enter the number of cans: 6
Total volume: 6.1312
```

Exercise 1 – Pseudocode

› Write down the pseudocode:

Exercise 1 – C++ Code

› Write down the code:

Introduction to Control Structures

- By default, statements in a program are executed in sequential order
  - All programs studied so far behave in this way
- C++ provides control structures to achieve *transfer of control*
  - Next statement executed *not necessarily* next one in sequence
- 2 kinds of control structures
  - Selection structures
    - **`if`**, **`if/else`**, **`switch`**
  - Repetition structures
    - **`while`**, **`do-while`**, **`for`**

An Example of Selection Structure

selectionExample.cpp

```cpp
#include <iostream>

using namespace std;

int main() {

    int x, y;

    cout << "Please enter x and y: ";
    cin >> x >> y;

    // if selection structure
    if (x > y) {
        cout << "x is greater." << endl;
    }
    else if (x < y) {
        cout << "x is smaller." << endl;
    }
    else {
        cout << "x and y are equal." << endl;
    }
}
```

```
Please enter x and y: 10 20
x is smaller.
```

`if` Selection Structure

- 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, program continues to next statement
- If the condition is **`false`**
  - *Print statement* ignored, program continues
- *Indenting* makes programs/pseudocode easier to read
  - Note: C++ ignores whitespace characters (tabs, spaces, etc.)

`if` Selection Structure

› Translate to C++

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

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

`if` Selection Structure

› General Structure

```
if (<condition>)
   statement
```

› A `condition` is one that, after evaluation, it must be either **`true`** or **`false`**

– E.g.,

`x <= y` (Is *x* less than or equal to *y*?)

`m != n` (Is *m* not equal to *n*?)

Equality and Relational Operators

- Used to compare the relation of two data items
- Result in either **`true`** or **`false`**
- Equality operators

  `==`, `!=`
- Relational operators

  `>`, `<`, `>=`, `<=`

Equality and Relational Operators

Standard algebraic relational operator or equality 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.

`if` Selection Structure

› If grade is 54, what will the output be?

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

`grade >= 60` will be evaluated to **`false`** and **`Passed`** will not be printed.

`if/else` Selection Structure

- Different actions if conditions **`true`** or **`false`**
- Pseudocode

  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";
```

`if/else` Selection Structure

› General Structure

```
if (<condition>)
   statement
else
   statement
```

› No condition after the "`else`" keyword, "`else`" means "`otherwise`".

Nested `if/else` Structures

- In most scenarios, there are more than two possible actions to be performed.
- By extending the `if/else` structures, another `if/else` structure is included in the "else" segment.

```
if (<condition>)
   statement for the 1st case
else
   if (<condition>)
        statement for the 2nd case
   else
        statement otherwise
```

Nested `if/else` Structures

› The following example illustrates how to determine the grade of a given numeric score:

```
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"
```

Nested `if/else` Structures

› C++ code

```
if (grade >= 90)  // 90 and above
      cout << "A";
else
      if (grade >= 80)  // 80-89
            cout << "B";
      else
            if (grade >= 70)  // 70-79
                  cout << "C";
            else
                  if (grade >= 60)  // 60-69
                        cout << "D";
                  else  // less than 60
                        cout << "F";
```

Nested `if/else` Structures

› Another indenting style (better)

```
if (grade >= 90)        // 90 and above
      cout << "A";
else if (grade >= 80)   // 80-89
      cout << "B";
else if (grade >= 70)   // 70-79
      cout << "C";
else if (grade >= 60)   // 60-69
      cout << "D";
else                    // less than 60
      cout << "F";
```

› Note: **`else`** and **`if`** are in alignment

`if/else` Selection Structure

- Compound statement
  - Set of statements within a pair of braces { }

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

- Without braces, what will happen?

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

always executes

`if/else` Selection Structure

- Block
  - Set of statements within braces
- Recommendation: Use braces also for single statement Why?

```
if (grade >= 60) {
    cout << "Passed.\n";
}
```

Ternary conditional operator (`?:`)

- To choose between two values based on a condition
- It is used for value assignment/utilization
- Three arguments
  - (1) – condition
  - (2) – use this value if (1) is **`true`**
  - (3) – use this value if (1) is **`false`**
- General form:

```
(1) ? (2) : (3)
```

Ternary conditional operator (?:)

› E.g.,

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

which is equivalent to

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

Assignment Operators

› Assignment expression abbreviations

› Statements of the form

variable = variable operator expression;

can be rewritten as

variable operator= expression;

E.g., Addition assignment operator

```
c = c + 3;
c += 3;
```

Assignment Operators

› Other assignment operators

```
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```

Increment and Decrement Operators

- Increment operator **++** can be used instead of **c += 1**
- Decrement operator **--** can be used instead of **c -= 1**
  - Pre-increment
    - When the operator is placed before the variable (**++c** or **--c**), the variable value is changed, then the entire expression is evaluated.
  - Post-increment
    - When the operator is placed after the variable (**c++** or **c--**), the entire expression is evaluated, then the variable value is changed.

increment.cpp

```cpp
// Demonstrating pre-increment and post-increment operators
#include <iostream>

using namespace std;

int main() {

    int counter1 = 10;
    int counter2 = 10;

    cout << "Original counter1: " << counter1 << endl;

    // pre-increment
    cout << ++counter1 << endl;

    cout << "Original counter2: " << counter2 << endl;

    // post-increment
    cout << counter2++ << endl;

    return 0;
}
```

```
Original counter1: 10
11
Original counter2: 10
10
```

Increment and Decrement Operators

› When the variable is not in expression,
  – pre-incrementing and post-incrementing have the same effect

› That means,

```
++c;
cout << c;
```

and

```
c++;
cout << c;
```

have the same behavior and output

Equality and Relational Operators

Standard algebraic relational operator or equality 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.

Equality and Relational Operators

- Outcome will either be **`true`** or **`false`**
- E.g.,

```
if (true) {
    cout << "You can always see me." << endl;
}
```

- In C++, **`true`** is represented (stored) as a non-zero value and **`false`** as 0 in computer memory

Confusing == and =

- Comparison Operators == and Assignment Operators =
- Common error
  - Does not typically cause syntax errors (i.e., the compiler does not notice any errors)
- Aspects of problem
  - Expressions that have a value can be used for decision (condition checking)
    - Remember how `true` and `false` are represented in computers?
    - Assignment statements produce a value
    - The value is then evaluated to make logical decision (may not be intended)

Confusing == and =

› E.g.,

```
if (payCode == 4) {
    cout << "You get a bonus!" << endl;
}
```

– If paycode is 4, bonus given

› When == is replaced by =

```
if (payCode = 4) {
    cout << "You get a bonus!" << endl;
}
```

– **paycode** set to 4
– The statement is ALWAYS **true** (since 4 is non-zero)
– Bonus given in every case

True/False Tables

T = true
F = false

AND		
T	T	T
T	F	F
F	T	F
F	F	F

OR		
T	T	T
T	F	T
F	T	T
F	F	F

Logical Operators

- ! (logical **NOT**, logical negation)
  - Returns **true** when its condition is **false**, and vice versa

```
if (!(grade == 'A')) {
   cout << "Work Hard!" << endl;
}
```

Alternative:

```
if (grade != 'A') {
    cout << "Work Hard!" << endl;
}
```

Logical Operators

› Used in the decision-making part of control structures

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

&& (logical **AND**)

- **true** if BOTH conditions are **true**

```
if (gender == 1 && age >= 65) {
   ++seniorFemales;
}
```

|| (logical **OR**)

- **true** if EITHER of condition is **true**

```
if (semesterAverage < 40 || finalExam < 30) {
   cout << "Student grade is F." << endl;
}
```

Exercise 2

- Write a program that reads three nonzero integers. Determine and prints whether they could be the sides of a right triangle.
- Hint: Pythagorean Theorem

Exercise 2

› Write down your code here:

`switch` Selection Structure

- Besides **`if`** and (nested) **`if/else`** control structures, we have **`switch`** structure to help decision making based on *discrete data value(s)*.
- In the following example, the program displays a rating based on an input letter grade.

switch Selection Structure

switch.cpp (Page 1 of 2)

```
// Demonstrating switch control structure
#include <iostream>

using namespace std;

int main() {

    char grade;

    cout << "Please enter a grade: ";
    cin >> grade;

    switch (grade) {

        case 'A':
            cout << "Excellent!" << endl;
            break;

        case 'B':
            cout << "Good!" << endl;
            break;
```

```
        case 'C':
        cout << "OK!" << endl;
        break;

        case 'D':
        cout << "Marginal" << endl;
        break;

        case 'F':
        cout << "Failed!" << endl;
        break;

        default:
        cout << "Input is Wrong!" << endl;
    }

    return 0;
}
```

switch.cpp (Page 2 of 2)

```
Please enter a grade: B
Good!
```

`switch` Selection Structure

- `switch`
  - Test variable for multiple values
  - Series of **`case`** labels and optional **`default`** case
  - variable can ONLY be short, int, long or char
- Remember
  - The switch structure is not designed to test a range of values, but discrete values!

`switch` Selection Structure

```
switch (variable) {

case value1:          // taken if variable equals value1
    statements
    break;            // necessary to exit switch

case value2:          // taken if variable equals value2
    statements
    break;            // necessary to exit switch

default:              // taken if variable matches no other cases
    statements
    break;            // optional
}
```

`switch` Selection Structure

- In the previous example, what if we want to handle both uppercase (e.g., 'A') and lowercase input grades (e.g., 'a')?
- Computers treat letters (characters) 'A' and 'a' differently
  - English letters are encoded as ASCII (American Standard Code for Information Interchange) codes in C++.

ASCII code table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Properties of ASCII code

- Space (32) comes before all the printable characters
- Numbers (48 – 57) come before (less than) letters
- Uppercase letters (65 – 90) always come before than lowercase letter (97 – 122)

switch2.cpp (Page 1 of 2)

```
// Demonstrating switch control structure
// Uppercase and lowercase input are also considered
#include <iostream>

using namespace std;

int main() {

    char grade;

    cout << "Please enter a grade: ";
    cin >> grade;

    switch (grade) {

        case 'a':
        case 'A':
            cout << "Excellent!" << endl;
            break;

        case 'b':
        case 'B':
            cout << "Good!" << endl;
            break;
```

```
    case 'c':
    case 'C':
        cout << "OK!" << endl;
        break;

    case 'd':
    case 'D':
        cout << "Marginal" << endl;
        break;

    case 'f':
    case 'F':
        cout << "Failed!" << endl;
        break;

    default:
        cout << "Input is Wrong!" << endl;
    }

    return 0;
}
```

switch2.cpp (Page 2 of 2)

```
Please enter a grade: c
OK!
```

`switch` Selection Structure

› Remarks
  – The previous example illustrates selection of appropriate action based on a **`char`** variable
  – For numeric values (i.e., **`int`**, **`short`** and **`long`**), no single quotes are required. E.g.,

```
int num;
cin >> num;

switch (num) {

     case 10:
          cout << "Ten" << endl;
          break;

     case 20:
          cout << "Twenty" << endl;
          break;
}
```

Summary

- Algorithm
- Pseudocode
- Introduction to Control Structures
- Selection Structures
  - **`if`** and **`if/else`**
  - Nested **`if/else`**
- Ternary conditional operator
- More operators
  - Assignment Operators
  - Increment and Decrement Operators
  - Logical Operators
- Selection Structures
  - **`switch`**