Lecture 4: Control Statements – Part I

Class page: https://github.com/tsung-wei-huang/cs1410-40

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Announcement

- **TODO: ask students to email PA with a "fixed-format" subject**

Learning Objective

- Basic problem-solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement.
- To use the `if` and `if...else` selection statements to choose among alternative actions.
- To use the `while` repetition statement to execute statements in a program repeatedly.
- Counter-controlled repetition and sentinel-controlled repetition.
- To use the increment, decrement and assignment operators.

Introduction

- **Before writing a program to solve a problem, we must have a thorough understanding of the problem and a carefully planned approach to solving it.**
- **When writing a program, we must also understand the types of building blocks that are available and employ proven program construction techniques.**
- **We use several lectures to discuss these issues as we present the theory and principles of structured programming.**

Algorithm

- **Any solvable computing problem can be solved by the execution of a series of actions in a specific order.**
- **An algorithm is procedure for solving a problem in terms of**
  - the actions to execute and
  - the order in which the actions execute
- **Specifying the order in which statements (actions) execute in a computer program is called program control.**
- **We investigate program control flow using C++'s control statements.**

Pseudocode

- **Pseudocode (or "fake" code) is an artificial and informal language that helps you develop algorithms.**
- **Similar to everyday English**
- **Convenient and user friendly.**
- **Helps you "think out" a program before attempting to write it.**
- **Carefully prepared pseudocode can easily be converted to a corresponding C++ program.**
- **Normally describes only executable statements.**
  - Declarations are not executable statements.

Pseudocode Example

```
1  Prompt the user to enter the first integer
2  Input the first integer
3
4  Prompt the user to enter the second integer
5  Input the second integer
6
7  Add first integer and second integer, store result
8  Display result
```

Control Structure

- **Normally, statements in a program execute one after the other in the order in which they're written.**
  - Called sequential execution.
- **Various C++ statements enable you to specify the next executing statement that is not the next one in sequence.**
  - Called transfer of control.
- **All programs could be written in terms of only three control structures (referred as "control statements")**
  - the sequence structure
  - the selection structure and
  - the repetition structure

Control Structure (cont'd)

- **C++ provides three types of selection statements**
- **The `if` selection statement: (single selection)**
  - The condition is true: perform the following action
  - The condition is false: skip the action
- **The `if...else` selection statement: (double selection)**
  - The condition is true: perform the following action
  - The condition is false: perform a different action
- **The `switch` selection statement: (multiple selection)**
  - Perform one of many different actions, depending on the value of selection expression.

Control Structure (cont'd)

- **C++ provides three repetition statements (also called looping statements) for performing statements repeatedly.**
  - These are the while, do…while and for statements.
- **The `while` and `for` statements perform the action (or group of actions) in their bodies zero or more times.**
- **The `do…while` statement performs the action (or group of actions) in its body at least once.**

Control Structure (cont'd)

- Each of the words `if`, `else`, `switch`, `while`, `do` and `for` is a C++ keyword.
- Keywords must not be used as identifiers, such as variable names.

Control Structure Keywords

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

and	and_eq	asm	bitand	bitor
bool	catch	class	compl	const_cast
delete	dynamic_cast	explicit	export	false
friend	inline	mutable	namespace	new
not	not_eq	operator	or	or_eq
private	protected	public	reinterpret_cast	static_cast
template	this	throw	true	try
typeid	typename	using	virtual	wchar_t
xor	xor_eq			

Control Structure (cont'd)

- **Each program combines control statements as appropriate for the algorithm the program implements.**
- **C++ control statements are single-entry/single-exit**
- **Control statements are attached to one another by connecting the exit point of one to the entry point of the next.**
  - Called control-statement stacking
- **Another way to connect control statements is containing one control statement inside another one**
  - Called control-statement nesting

`if` Selection Statement

- **The following pseudocode determines whether "student's grade is greater than or equal to 60" is `true` or `false`.**

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

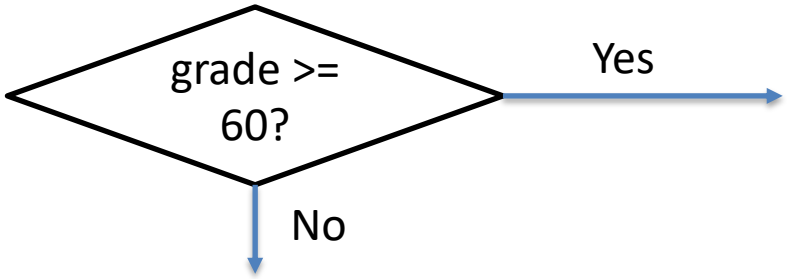
  - If `true`, "Passed" is printed and the next pseudocode statement in order is "performed".
  - If `false`, the print statement is ignored and the next pseudocode statement in order is performed.
  - The indentation of the second line is optional, but it's recommended because it emphasizes the inherent structure of structured programs.

`if` Selection Statement

- **The preceding pseudocode can be written in C++ as**

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

- **The diamond (decision symbol) indicates that a decision is to be made.**
  - The workflow will continue along a path determined by the symbol's associated guard conditions, which can be true or false.
  - If a guard condition is true, the workflow enters the action state to which that transition arrow points.



if Selection Statement

- If the expression evaluates to zero, it's treated as false; if the expression evaluates to nonzero, it's treated as true.
- C++ provides the data type bool for variables that can hold only the values true and false—each of these is a C++ keyword.
- For compatibility with earlier versions of C, which used integers for Boolean values, the bool value true also can be represented by any nonzero value

if…else Double-Selection Statement

- **`if…else` double-selection statement**
  - specifies an action to perform when the condition is true and a different action to perform when the condition is `false`.
- **The following pseudocode prints "Passed" if the student's grade is greater than or equal to 60, or "Failed" if the student's grade is less than 60.**

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

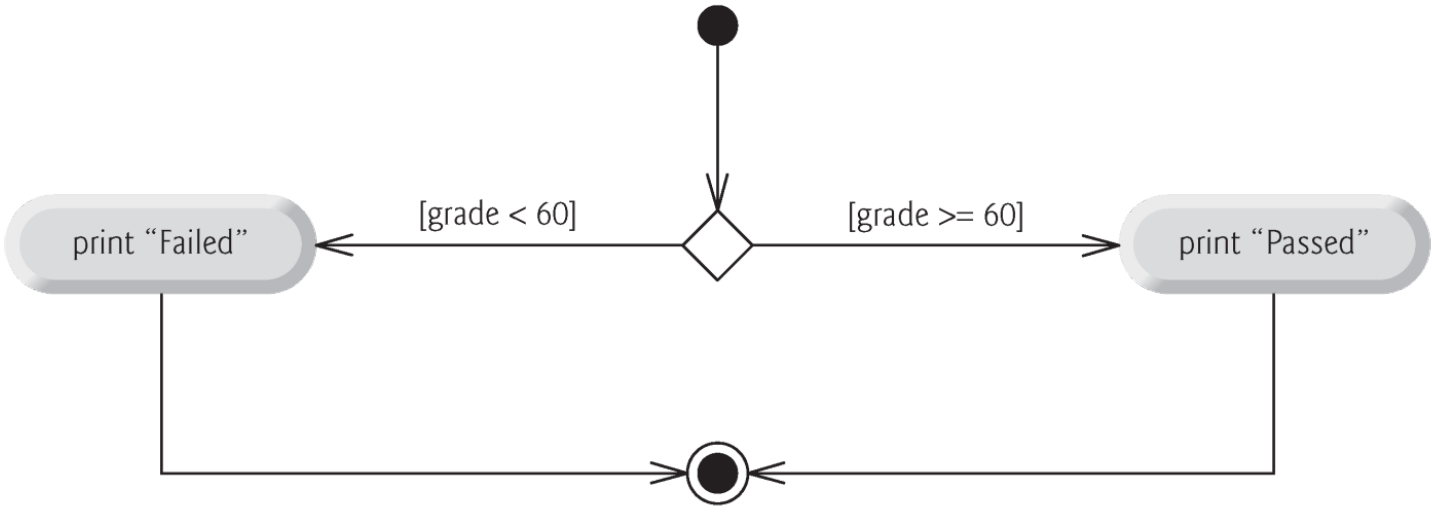
- **In either case, after printing occurs, the next pseudocode statement in sequence is "performed."**

if…else Double-Selection Statement

- The preceding pseudocode *If…Else* statement can be written in C++ as

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

- The control flow of `if…else` is shown as follows.



if…else Double-Selection Statement

```
1. (grade>=60) ? cout << “Passed” : cout << “Failed” ;
2. final = (grade>=50) ? 60 : grade ;
```

- **Conditional operator (?:)**
  - Closely related to the `if…else` statement.
- **C++’s only ternary operator—it takes three operands.**
  - The first operand is a condition
  - The second operand is the value if the condition is `true`
  - The third operand is the value if the condition is `false`.
- **The values in a conditional expression can be actions.**

if…else Double-Selection Statement

- **Nested if…else statements test for multiple cases by placing `if…else` selection statements inside other ones.**

```
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"
```

if...else Double-Selection Statement

The pseudo code can rewritten in C++ as follows.

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

- **If studentGrade is greater than or equal to 90, only the output statement after the first test executes.**
  - Skip the other branches

if...else Double-Selection Statement

- Most write the preceding `if...else` statement as

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

- The two forms are identical except for the spacing and indentation, which the compiler ignores.
- The latter form is popular because it avoids deep indentation.

if…else Double-Selection Statement

- The C++ compiler always associates an `if` or `else` with the immediately preceding actions.
- This behavior can lead to the **dangling-else problem**.

```
if ( x > 5 )
   if ( y > 5 )
      cout << "x and y are > 5";
else
      cout << "x is <= 5";
```

- What's the difference with a pair of braces ({}) ?

```
if ( x > 5 )
{
   if ( y > 5 )
      cout << "x and y are > 5";
}
else
   cout << "x is <= 5";
```

if...else Double-Selection Statement

- The `if` selection statement expects only one statement in its body.
- Similarly, the `if` and `else` parts of an `if...else` statement each expect only one body statement.
- To include several statements in the body of an `if` or in either part of an `if...else`, enclose the statements in braces (`{` and `}`).
- A set of statements contained within a pair of braces is called a compound statement or a block.

if…else Double-Selection Statement

- A block {} can be placed anywhere in a program that a single statement can be placed

- Forgetting one or both of the braces that delimit a block can lead to syntax errors or logic errors in a program

```
{  int a;      // missing '}'
   ing b; }  // missing '{'
```

- Always putting the braces in an if…else statement (or any control statement) helps prevent their accidental omission, especially when adding statements to an if or else clause at a later time.

if…else Double-Selection Statement

- Just as a block can be placed anywhere a single statement can be placed, it's also possible to have no statement at all

  —called a null statement (or an empty statement).

```
{
   ;    // empty statement
}
```

- The null statement is represented by placing a semicolon (;) where a statement would normally be.

while Repetition Statement

- **A repetition statement (also called a looping statement) allows you to repeat an action while some condition remains true.**

While there are more items on my shopping list
Purchase next item and cross it off my list

- **Consider a program segment designed to find the first power of 3 larger than 100.**
  - Suppose the integer variable `product` has been initialized to 3.
- **When the following `while` repetition statement finishes executing, `product` contains the result:**

```
int product = 3;
while ( product <= 100 ) {
   product = 3 * product;
   cout << product << '\n';
}
```

Avoid Writing Infinite Loop!

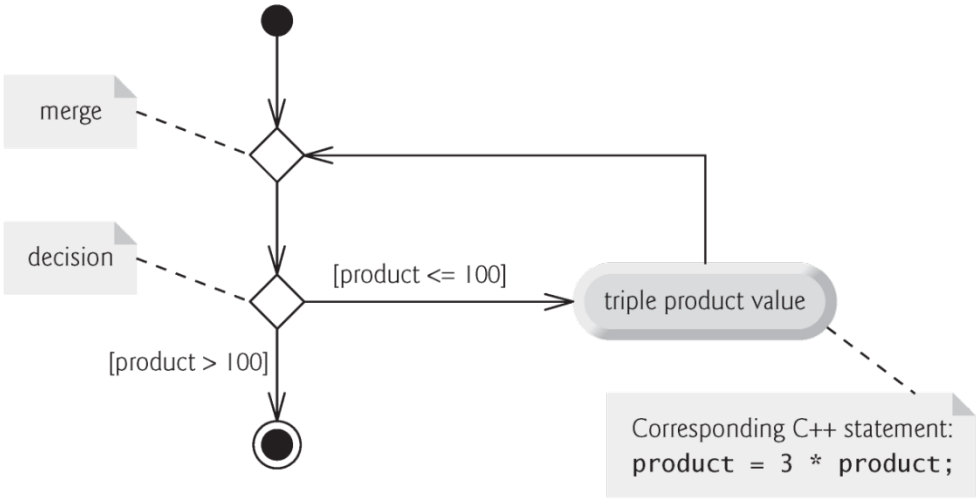
- **Not providing, in the body of a while statement, an action that eventually causes the condition in the while to become false normally results in a logic error called an "in-finite" loop, in which repetition never stops!**

```
int product = 3;
while ( product >= 3 ) {
   product = 3;
}
```

- **Infinite loop makes a program appear to hang or freeze if the loop body does not contain statements that interact with the user**
  - This is a very frequently used method to attack your computer (computer virus)

Diagram of the While statement

- `while` statement can be represented by a merge symbol
- The merge symbol joins the transitions from the initial state and from the action state
  - Determine whether the loop should begin (or continue) executing
- A merge symbol has two or more input transition arrows and only one output transition arrow
  - Unlike the decision symbol, the merge symbol does not have a counterpart in C++ code



Formulating Algorithms: Counter-Controlled Repetition

- **Consider the following problem statement:**
  - A class of 10 students taking a quiz.
  - The grades (integers in the range 0 to 100) are available to you.
  - Calculate and display the total of all student grades and the class average on the quiz.
- **We use counter-controlled repetition to input the 10 grades one by one.**
  - This technique uses a variable called a counter to control the number of times a group of statements will execute (also known as the number of iterations of the loop).
  - Often called definite repetition because the number of repetitions is known before the loop begins executing.

Formulating Algorithms: Counter-Controlled Repetition

- A **total** is a variable used to accumulate the sum of several values.
- A **counter** is a variable used to count—in this case, the grade counter indicates which of the 10 grades is about to be entered by the user.
- The class average is equal to the sum of the grades (**total**) divided by the number of students (**10**).
- Dividing two integers results in integer division—any fractional part of the calculation is lost (i.e., **truncated**).

Formulating Algorithms: Counter-Controlled Repetition

```
2  // Class average program with counter-controlled repetition.
3  #include <iostream>
4  using namespace std;
5
6  int main ()
7  {
8     int total; // sum of grades entered by user
9     int gradeCounter; // number of the grade to be entered next
10    int grade; // grade value entered by user
11    int average; // average of grades
12
13    // initialization phase
14    total = 0; // initialize total
15    gradeCounter = 1; // initialize loop counter
16
17    // processing phase
18    while ( gradeCounter <= 10 ) // loop 10 times
19    {
20       cout << "Enter grade: "; // prompt for input
21       cin >> grade; // input next grade
22       total = total + grade; // add grade to total
23       gradeCounter = gradeCounter + 1; // increment counter by 1
24    } // end while
```

Formulating Algorithms: Counter-Controlled Repetition

```
25
26      // termination phase
27      average = total / 10; // integer division yields integer result
28
29      // display total and average of grades
30      cout << "\nTotal of all 10 grades is " << total << endl;
31      cout << "Class average is " << average << endl;
32   } // end main
```

```
Enter grade: 67
Enter grade: 78
Enter grade: 89
Enter grade: 67
Enter grade: 87
Enter grade: 98
Enter grade: 93
Enter grade: 85
Enter grade: 82
Enter grade: 100

Total of all 10 grades is 846
Class average is 84
```

Example from Lecture 1

- **Find a power of a number**
  - Input: a, b ($1 < a, b < 2147483647$)
  - Output: $x = a^b$
    - $a=3, b=4, x=3^4=81$
    - $a=2, b=5, x=2^5=32$
  - Assume you can only do multiplication one at a time
- **Naïve method**
  - $2^{16} = 2*2*2*2*2*2*\ldots*2$ total 15 calculations
- **Can we do better?**

Divide and Conquer

- **Naïve method**
  - $2^{16} = 2*2*2*2*2*2*\ldots*2$ total 15 calculations
- **A better way as follows:**

$2^{16} = 2^8 * 2^8$

$2^8 = 2^4 * 2^4$

$2^4 = 2^2 * 2^2$

$2^2 = 2 * 2$ We need only 4 calculations!!!

How Efficient is it to Compute a^b?

- **Naïve method**
  - # calculations: linear to b
- **Divide and Conquer**
  - # calculations: $\log_2(b)$
- **Let's say n = 2147483648**
  - Naïve method takes **2147483647** calculations (~10-30s)
  - Divide and Conquer takes only **31** calculations (~1us)
    - 10000000x faster!
  - Indeed, this is a Goo___ interview question

Practice

- **Write a program that asks the user to type in two integer number a and b; your program then computes the value $x = a^b$ and print the result**
  - For simplicity, don't worry about overflow now
  - use while statement to manipulate the counter
- **Compare the two versions**
  - Linear time
  - Log time
- **Both versions should output the same value**
- **Do you see speed difference when b is large?**

Summary

- **Control flow**
- **Algorithm and pseudocode**
- **If..else statement**
- **while repetition statement**
- **Programming Assignment 2 is out**
  - Due 9/9 by class time
  - Email your solution to your LAB section TA