



Chapter 3

More Flow of Control

Prof. Chien-Nan (Jimmy) Liu
Dept. of Electrical Engineering
National Chiao-Tung Univ.

Tel: (03)5712121 ext:31211
E-mail: jimmyliu@nctu.edu.tw
<http://mseda.ee.nctu.edu.tw/jimmyliu>



Chien-Nan Liu, NCTUEE



Overview

3.1 Using Boolean Expressions

3.2 Multiway Branches

3.3 More about C++ Loop Statements

3.4 Designing Loops



Chien-Nan Liu, NCTUEE



Truth Tables of Boolean Operations

AND

<i>Exp_1</i>	<i>Exp_2</i>	<i>Exp_1 && Exp_2</i>
true	true	true
true	false	false
false	true	false
false	false	false

NOT

<i>Exp</i>	<i>!(Exp)</i>
true	false
false	true

OR

<i>Exp_1</i>	<i>Exp_2</i>	<i>Exp_1 Exp_2</i>
true	true	true
true	false	true
false	true	true
false	false	false



Chien-Nan Liu, NCTUEE

3-3



Precedence Rules

- Items in expressions are grouped by **precedence rules** for arithmetic and boolean operators
 - Operators with **higher precedence** are performed first
 - For the operators with equal precedence:
 - Binary operators** are performed **left to right**
 - Unary operators** are performed **right to left**
 - If you are not sure about the precedence, **add parenthesis!!**

The unary operators : ++, --, !
The binary arithmetic operations : *, /, %
The binary arithmetic operations : +, -
The Boolean operations : <, >, <=, >=
The Boolean operations : ==, !=
The Boolean operations : &&
The Boolean operations : ||

Highest precedence
(done first)

Lowest precedence
(done last)



Chien-Nan Liu, NCTUEE

3-4

Precedence Rule Example

- The expression

$$(x+1) > 2 \mid \mid (x + 1) < -3$$

is equivalent to

$$((x + 1) > 2) \mid \mid ((x + 1) < -3)$$

is also equivalent to

$$x + 1 > 2 \mid \mid x + 1 < -3$$

- Because $>$ and $<$ have higher precedence than $\mid \mid$
- According to the precedence rules:
 - First apply the unary $-$
 - Next apply the $+$'s
 - Now apply the $>$ and $<$
 - Finally do the $\mid \mid$



Chien-Nan Liu, NCTUEE

3-5

Short-Circuit Evaluation

- Some boolean expressions do not need to be completely evaluated → **short-circuit evaluation**
 - if x is negative, the value of the expression $(x \geq 0) \ \&\& \ (y > 1)$ can be determined by evaluating only $(x \geq 0)$
- Sub-expressions are evaluated from **left to right**
 - Once the final value is determined, the rest of the expression is not evaluated
- Short-circuit evaluation can prevent run-time errors

```
if ((kids != 0) && (pieces / kids >= 2) )  
    cout << "Each child may have two pieces!";
```

- If $kids$ is zero, $(pieces / 0 \geq 2)$ is not executed!!
 - **Division by zero** causes a run-time error



Chien-Nan Liu, NCTUEE

3-6

Problems with !

Given an expression (! time > limit).

If limit = 60, time = 36, what is evaluation result?

- According to the precedence rules, it is evaluated as $(!time) > limit$
- *time* is an integer (36), what is *!time* ?
 - **FALSE!** or **zero** since it will be compared to an integer
 - The expression is further evaluated as $0 > limit \rightarrow false$
- The intent of the expression is most likely the expression $(! (time > limit))$
 - With parenthesis, it is evaluated as $(! (false)) \rightarrow true$



Chien-Nan Liu, NCTUEE

3-7

Avoiding !

- Just as not in English can make things not undifficult to read, the ! operator can make C++ expressions difficult to understand
 - 你是否不同意未來不應不使用不能讓民眾不理解的多重否定句?? → 好懂嗎??
- Before using the ! operator see if you can express the same idea more clearly without the ! operator



Chien-Nan Liu, NCTUEE

3-8

Enumeration Types

- An **enumeration type** is a type with values defined by **a list of integer constants**
- If numeric values are not specified, identifiers are assigned consecutive values **starting with 0**
 - `enum Direction { NORTH = 0, SOUTH = 1, EAST = 2, WEST = 3};`
is equivalent to
`enum Direction {NORTH, SOUTH, EAST, WEST};`
- Unless specified, the value assigned an enumeration constant is **1 more** than the previous constant
 - `enum Test {ONE = 17, TWO, THREE, FOUR = -3, FIVE};`
→ ONE = 17, TWO = **18**, THREE = **19**, FOUR = -3, FIVE = **-2**
(red values are automatically assigned)



Chien-Nan Liu, NCTUEE

3-9

Comparison of Using enum

```
// 0 = Menu
// 1 = Playing
// 2 = Pause
// 3 = GameOver

int status = 0;
.....
if ( status == 0 )
{
    // 開頭選單處理.....
}
if ( status == 1 )
{
    // 遊戲進行中處理.....
}
```

- You may forget the meaning of those hard-coded numbers in a large program!!

```
enum GameStatus
{
    Menu,      // 開頭選單
    Playing,   // 遊戲進行中
    Pause,     // 遊戲暫停
    GameOver  // 遊戲結束
}
GameStatus status = Menu;
.....
If ( status == Menu )
{
    // 開頭選單處理.....
}
If ( status == Playing )
{
    // 遊戲進行中處理.....
}
.....
```



Chien-Nan Liu, NCTUEE

3-10

Overview

3.1 Using Boolean Expressions

3.2 *Multiway Branches*

3.3 More about C++ Loop Statements

3.4 Designing Loops



Chien-Nan Liu, NCTUEE

3-11

Nested Statements

- A statement that is a subpart of another statement is a nested statement

- Example:

suggested to indent each level of nesting

```
if (count < 10)
    if ( x < y)
        cout << x << " is less than " << y;
    else
        cout << y << " is less than " << x;
```

```
if (count > 0)
    if (score > 5)
        cout << "count > 0 and score > 5\n";
    else
        cout << "count > 0 and score <=5\n";
```



Chien-Nan Liu, NCTUEE

3-12

Nested if-else Statements

To design an if-else statement to warn a driver when fuel is low, but tells the driver to bypass gas stations if the fuel is close to full. Otherwise, there should be no output.

■ Straight-forward approach

```
if (fuelGaugeReading < 0.75)
  if (fuelGaugeReading < 0.25)
    cout << "Fuel very low. Caution!\n";
  else
    cout << "Fuel over 3/4. Don't stop\n";
```

Read=0.5 → Fuel over 3/4. Don't stop

Read=0.8 → Nothing

Correct??

■ Correct approach

```
if (fuelGaugeReading < 0.75)
{
  if (fuelGaugeReading < 0.25)
    cout << "Fuel very low. Caution!\n";
}
else
  cout << "Fuel over 3/4. Don't stop\n";
```

Read=0.5 → Nothing

Read=0.8 → Fuel over 3/4. Don't stop



Chien-Nan Liu, NCTUEE

3-13

Braces and Nested Statements

- The compiler pairs the "else" with the **nearest previous "if"**
 - **Only one statement** is allowed under each "if" and "else" without braces
 - Not depend on the number of spaces before if/else
- **Braces** in nested statements are like parenthesis in arithmetic expressions
 - Braces group things as one statement in compiler
- Use braces around substatements
 - **Multiple operations** are allowed under each branch
 - Be careful to check the hierarchy of each statement



Chien-Nan Liu, NCTUEE

3-14

Example of Nested if (Display 3.4)

```
double fuelGaugeReading;
cout << "Enter fuel gauge reading: ";
cin >> fuelGaugeReading;

cout << "First with braces:\n";
if (fuelGaugeReading < 0.75)
{
    if (fuelGaugeReading < 0.25)
        cout << "Fuel very low. Caution!\n";
}
else
{
    cout << "Fuel over 3/4. Dont stop now!\n";
}

cout << "Now without braces:\n";
if (fuelGaugeReading < 0.75)
    if (fuelGaugeReading < 0.25)
        cout << "Fuel very low. Caution!\n";
else
    cout << "Fuel over 3/4. Don't stop now!\n";
```

Sample Dialogue 1

```
Enter fuel gauge reading: 0.1
First with braces:
Fuel very low. Caution!
Now without braces:
Fuel very low. Caution!
```

Braces make no difference in this case, but see Dialogue 2.

Sample Dialogue 2

```
Enter fuel gauge reading: 0.5
First with braces:
Now without braces:
Fuel over 3/4. Don't stop now!
```

There should be no output here, and thanks to braces, there is none.
Incorrect output from the version without braces.



Chien-Nan Liu, NCTUEE

3-15

Multi-way if-else-statements

- Three or four (or more) way branches can be designed using **nested if-else-statements**

- Example: guess the number stored in variable by proper hints
- if (guess > number)
 cout << "Too high.";
else
 if (guess < number)
 cout << "Too low.";
 else
 if (guess == number)
 cout << "Correct!";

guess > number → FALSE
guess < number → TRUE

guess > number → FALSE
guess < number → FALSE
guess == number → TRUE



Chien-Nan Liu, NCTUEE

3-16

Indenting Nested if-else

- The code on the previous page looks like a slide
 - Leaving less and less space for coding
- There is an alternative way for indenting several nested if-else-statements
- When the conditions in an if-else-statement are mutually exclusive, the final if-else can be omitted

```
if (guess > number)
    cout << "Too high.";
else if (guess < number)
    cout << "Too low.";
else if (guess == number)
    cout << "Correct!";
```



```
if (guess > number)
    cout << "Too high.";
else if (guess < number)
    cout << "Too low.";
else
    cout << "Correct!";
```

all other possibilities



Chien-Nan Liu, NCTUEE

3-17

Example: State Income Tax

- Write a program for a state that computes tax according to the rate schedule:
 - No tax on first \$15,000 of income
 - 5% tax on each dollar from \$15,001 to \$25,000
 - 10% tax on each dollar over \$25,000
- Code example using "if-else-if" is shown in next page
- Notice that the line

```
else if (( netIncome > 15000 && netIncome <= 25000))
```

can be replaced with

```
else if (netIncome <= 25000)
```

- The computer will not get to this line unless it is already determined that `netIncome > 15000`



Chien-Nan Liu, NCTUEE

3-18

Code Example for Incoming Tax

```
cout << "Enter net income (rounded to whole dollars) $";  
cin >> netIncome;
```

```
if (netIncome <= 15000)  
    taxBill = 0;  
else if ((netIncome > 15000) && (netIncome <= 25000))  
    taxBill = (0.05*(netIncome - 15000)); //5% of amount over $15,000  
else //netIncome > $25,000  
{  
    //fivePercentTax = 5% of income from $15,000 to $25,000.  
    fivePercentTax = 0.05*10000;  
    //tenPercentTax = 10% of income over $25,000.  
    tenPercentTax = 0.10*(netIncome - 25000);  
    taxBill = (fivePercentTax + tenPercentTax);  
}
```

```
cout.setf(ios::fixed);  
cout.setf(ios::showpoint);  
cout.precision(2);  
cout << "Net income = $" << netIncome << endl;  
cout << "Tax bill = $" << taxBill << endl;
```

Sample Dialogue

```
Enter net income (rounded to whole dollars) $25100  
Net income = $25100.00  
Tax bill = $510.00
```



Chien-Nan Liu, NCTUEE

3-19

The switch Statement

- The switch-statement is an alternative for constructing multi-way branches

```
switch (controlling expression)  
{  
    case Constant_1:  
        Statement_Sequence_1  
        break;  
    case Constant_2:  
        Statement_Sequence_2  
        break;  
    . . .  
    case Constant_n:  
        Statement_Sequence_n  
        break;  
    default:  
        Default_Statement_Sequence  
}
```



Chien-Nan Liu, NCTUEE

3-20

Example: Letter Grade Using switch

- This example determines output based on a letter grade
 - Grades 'A', 'B', and 'C' each have a branch
 - Grades 'D' and 'F' use the same branch
 - If an invalid grade is entered, a default branch is used



Chien-Nan Liu, NCTUEE

A switch Statement (part 2 of 2)

Sample Dialogue 1

Enter your midterm grade and press Return: A
Excellent. You need not take the final.
End of program.

Sample Dialogue 2

Enter your midterm grade and press Return: B
Very good. Your midterm grade is now A.
End of program.

Sample Dialogue 3

Enter your midterm grade and press Return: D
Not good. Go study.
End of program.

Sample Dialogue 4

Enter your midterm grade and press Return: E
That is not a possible grade.
End of program.

3-21

Code Example for Letter Grade

DISPLAY 3.6 A switch Statement

//Program to illustrate the switch statement.

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

```
int main( )
```

```
{
```

```
    char grade;
```

```
    cout << "Enter your midterm grade and press Return: ";
    cin >> grade;
```

```
    switch (grade)
    {
        case 'A':
            cout << "Excellent. "
                  << "You need not take the final.\n";
            break;
```

```
        case 'B':
            cout << "Very good. ";
            grade = 'A';
            cout << "Your midterm grade is now "
                  << grade << endl;
            break;
        case 'C':
            cout << "Passing.\n";
            break;
        case 'D':
        case 'F':
            cout << "Not good. "
                  << "Go study.\n";
            break;
        default:
            cout << "That is not a possible grade.\n";
    }
```

```
    cout << "End of program.\n";
    return 0;
}
```



Chien-Nan Liu, NCTUEE

3-22

The Controlling Statement

- A switch statement's controlling statement must return one of these types
 - A **bool value**, ex: `switch (x>5) → case FALSE: ...`
 - An **enum constant**, ex: `switch (status) → case Playing: ...`
 - An **integer type**, ex: `switch (num) → case 5: ...`
 - A **character**, ex: `switch (grade) → case 'A': ...`
- The value returned is **compared** to the constant values after each "case"
 - When a match is found, the code for that case is used
- If no match is found, the statements following the **default label** are executed
 - Nothing happen if no default statement is provided
 - Better to prepare a default section for **unpredictable cases**



Chien-Nan Liu, NCTUEE

3-23

The break in switch Statement

- The **break** statement ends the switch-statement
 - Omitting the break statement will cause the code for the next case to be executed!
 - If multiple cases are executed together, it violates the mutual exclusive assumption (**unless you intend to do so**)
- Omitting a break statement allows the use of **multiple case labels** for a section of code
 - case 'A':
case 'a':

```
cout << "Excellent.";
break;
```
 - Runs the same code for either 'A' or 'a'



Chien-Nan Liu, NCTUEE

3-24

Switch Statements and Menus

- Nested if-else statements can be used in more cases than a switch statement
- However, switch statements can make code more clear in some cases
- Ex: Menu

Sample Dialogue

```
Choose 1 to see the next homework assignment.
Choose 2 for your grade on the last assignment.
Choose 3 for assignment hints.
Choose 4 to exit this program.
Enter your choice and press Return: 3
```

```
Assignment hints:
Analyze the problem.
Write an algorithm in pseudocode.
Translate the pseudocode into a C++ program.
```

```
Choose 1 to see the next homework assignment.
Choose 2 for your grade on the last assignment.
Choose 3 for assignment hints.
Choose 4 to exit this program.
Enter your choice and press Return: 4
End of Program.
```

The exact output will depend on the code inserted into the switch statement.



Chien-Nan Liu, NCTUEE

3-25

Creating a Menu Using switch

```
//DISPLAY 3.7 A Menu
//Program to give out homework assignment
information.
#include <iostream>
using namespace std;

int main( )
{
    int choice;

    do
    {
        cout << endl
            << "Choose 1 to see the next homework "
            << "assignment.\n"
            << "Choose 2 for your grade on the last "
            << "assignment.\n"
            << "Choose 3 for assignment hints.\n"
            << "Choose 4 to exit this program.\n"
            << "Enter your choice and press Return: ";
        cin >> choice;
```

```
        switch (choice)
        {
            case 1:
                // code to display the next assignment
                // on screen would go here.
                break;
            case 2:
                // code to ask for a student number and
                // give the corresponding grade.
                break;
            case 3:
                // code to display a hint
                break;
            case 4:
                cout << "End of Program.\n";
                break;
            default:
                cout << "Not a valid choice.\n"
                    << "Choose again.\n";
        }
    } while (choice != 4);
```

```
    return 0;
}
```



Chien-Nan Liu, NCTUEE

3-26

Blocks

- Each branch of a switch or if-else statement is a **separate sub-task**
 - Using **functional calls** (see Ch.4) instead of multiple statements can make the code much easier to read
 - If the action of a branch is too simple to warrant a function call, use **multiple statements between braces**
- A **block** is a section of code enclosed by **braces**
 - Variables **declared within a block** are **local** to the block or have the block as their scope
 - Variable names declared in the block **cannot be reused** outside the block
 - Bounded by the enclosed braces { }



Chien-Nan Liu, NCTUEE

3-27

Code Example for Sales Tax

```
const double TAX_RATE = 0.05; //5% sales tax.
```

```
int main( )
{
```

```
    char saleType;
    int number;
```

```
    double price, total;
```

```
    cout << "Enter price $";
    cin >> price;
    cout << "Enter number purchased: ";
    cin >> number;
    cout << "Type W if this is a wholesale purchase.\n"
        << "Type R if this is a retail purchase.\n"
        << "Then press Return.\n";
    cin >> saleType;
```

```
    if ((saleType == 'W') || (saleType == 'w'))
        total = price * number;
    else if ((saleType == 'R') || (saleType == 'r'))
    {
```

```
        double subtotal;
        subtotal = price * number;
        total = subtotal + subtotal * TAX_RATE;
```

```
    }
    else
        cout << "Error in input.\n";

    cout << number << " items at $" << price << endl;
    cout << "Total Bill = $" << total;
    if ((saleType == 'R') || (saleType == 'r'))
        cout << " including sales tax.\n";
    return 0;
}
```

```
Enter price: $10.00
Enter number purchased: 2
Type W if this is a wholesale purchase.
Type R if this is a retail purchase.
Then press Return.
R
2 items at $10.00
Total Bill = $21.00 including sales tax.
```



Chien-Nan Liu, NCTUEE

3-28

Overview

3.1 Using Boolean Expressions

3.2 Multiway Branches

3.3 *More about C++ Loop Statements*

3.4 Designing Loops



Chien-Nan Liu, NCTUEE

3-29

number++ vs ++number

- *(number++)* returns the current value of *number*, then increments *number*
 - An expression using *(number++)* will use the value of *number* **BEFORE** it is incremented
- *(++number)* increments *number* first and returns the new value of *number*
 - An expression using *(++number)* will use the value of *number* **AFTER** it is incremented
- Finally, *number* has **the same value after either version!**
- This rule is also applied on the **decrement operator (--)** that decreases the value of the variable by one
 - `cout << number--;` → display 8 (assume number = 8)
 - `cout << --number;` → display 7



Chien-Nan Liu, NCTUEE

3-30



Code Example for count++

```
int main( )
{
    int numberOfItems, count,
        caloriesForItem, totalCalories;

    cout << "How many items did you eat today? ";
    cin >> numberOfItems;

    totalCalories = 0;
    count = 1;
    cout << "Enter the number of calories in each of the\n"
        << numberOfItems << " items eaten:\n";

    while (count++ <= numberOfItems)
    {
        cin >> caloriesForItem;
        totalCalories = totalCalories
            + caloriesForItem;
    }
    cout << "Total calories eaten today = "
        << totalCalories << endl;
    return 0;
}
```

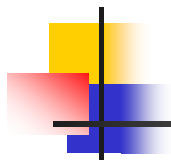
Sample Dialogue

```
How many items did you eat today? 7
Enter the number of calories in each of the
7 items eaten:
300 60 1200 600 150 1 120
Total calories eaten today = 2431
```



Chien-Nan Liu, NCTUEE

3-31



The for Statement

- A for-Statement (**for-loop**) is another loop mechanism in C++
 - Designed for common tasks such as adding numbers in a given range
 - Sometimes more convenient to use than a while loop
- The for loop uses the same components as the while loop in a more compact form
 - Looks like a special case of while loop
 - Ex: for (n = 1; n <= 10; n++) → The semicolons divide a for statement into 3 sections

Initialization Action

Update Action

Running Condition



Chien-Nan Liu, NCTUEE

3-32

for/while Loop Comparison

- `sum = 0;`
① `n = 1;`
② `while (n <= 10) // add the numbers 1 - 10`
{
 `sum = sum + n;`
 ③ `n++;`
}
- `sum = 0;`
① `for (n = 1; n <= 10; n++) //add the numbers 1 - 10`
 `sum = sum + n;`



Chien-Nan Liu, NCTUEE

3-33

Initialization in for Loop

- A for loop can also include a **variable declaration** in the initialization action
 - `for (int n = 1; n <= 10; n++)`
This line means
 - Create a variable, `n`, of type `int` and initialize it with 1
 - Continue to iterate the body as long as `n <= 10`
 - Increment `n` by one after each iteration
 - Variables declared within for loop are **local variables**
 - Such variables **cannot be reused outside** this for loop
- The initialization and increment expressions allow **multiple operations** separated by a **comma**
 - Ex: `for (int i=1, j=1; i <= 10; i++, j++)`



Chien-Nan Liu, NCTUEE

3-34

Code Example for Initialization

//DISPLAY 3.12 A for Statement

//Illustrates a for loop.

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

```
int main( )
{
    int sum = 0;

    for (int n = 1; n <= 10; n++) //Note that the variable n is a local
        sum = sum + n;           //variable of the body of the for loop!

    cout << "The sum of the numbers 1 to 10 is "
         << sum << endl;
    return 0;
}
```

Output

The sum of the numbers 1 to 10 is 55



Chien-Nan Liu, NCTUEE

3-35

for-loop Details

- Initialization and update actions of for-loops often contain more complex expressions, for example:
 - Vary the control variable from 1 to 100 in increments of 1
`for (int i = 1; i <= 100; i++)`
 - Vary the control variable from 100 down to 1 in decrements of 1
`for (int i = 100; i >= 1; i--)`
 - Vary the control variable from 7 to 77 in steps of 7
`for (int i = 7; i <= 77; i += 7)`
 - Vary the control variable from 20 down to 2 in steps of -2
`for (int i = 20; i >= 2; i -= 2)`
 - Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17
`for (int i = 2; i <= 17; i += 3)`



Chien-Nan Liu, NCTUEE

3-36

The for-loop Body

- The body of a for-loop can be

- A single statement
- A compound statement enclosed in braces

- Ex:

```
for(int number = 100; number >=0; number--)  
{  
    cout<<number  
        <<" bottles of beer on the shelf.\n";  
    if(number > 0)  
        cout<< "Take one down and pass it around.\n";  
}
```



Chien-Nan Liu, NCTUEE

Syntax

```
for (Initialization; Boolean_Expr; Update_Action)  
{  
    Statement_1  
    Statement_2  
    .  
    .  
    Statement_Last  
}
```

3-37

Extra Semicolon

- Placing a **semicolon after nothing** creates an empty statement that compiles but does nothing
- Placing a semicolon after the parentheses of a for loop creates an **empty statement** as the loop body

- Example:

```
for (int count = 1; count <= 10; count++)  
    cout << "Hello\n";
```

Do nothing
for 10 times!!

prints only one "Hello", but not as part of the loop!

- The empty statement is the body of the loop
- cout << "Hello\n"; is not part of the loop body!



Chien-Nan Liu, NCTUEE

3-38

Which Loop To Use?

- Choose the type of loop late in the design process
 - First design the loop concept using pseudocode
- **for-loops** are often selected in numeric calculations
 - When using a variable changed by equal amounts each time the loop iterates
- A **while-loop** is typically used when there are some cases that the loop body is not executed at all
 - while-loops can be applied on more general cases
- A **do-while-loop** is often adopted when the loop body must be executed at least once
 - Try at least once to obtain a result for making decision



The break and continue Statement

- There are times to exit a loop before it ends
 - If the loop checks for invalid input that would ruin a calculation, it is often best to end the loop
- The **break**-statement causes immediate exit from loop statement before normal termination
 - Used in while, for, do...while or switch statement
 - Be careful with nested loops! Using break only exits the loop in which the break-statement occurs (**1 level only**)
- The **continue**-statement skips the remaining statements in loop body and starts next iteration
 - Can be used in while, for, and do...while statement
 - Evaluate the loop-continuation test immediately



Code Example to Exit a Loop Early

```
int main( )
{
    int number, sum = 0, count = 0;
    cout << "Enter 10 negative numbers:\n";

    while (++count <= 10)
    {
        cin >> number;

        if (number >= 0)
        {
            cout << "ERROR: positive number"
                << " or zero was entered as the\n"
                << count << "th number! Input ends "
                << "with the " << count << "th number.\n"
                << count << "th number was not added in.\n";
            break;
        }
        sum = sum + number;
    }

    cout << sum << " is the sum of the first "
        << (count - 1) << " numbers.\n";

    return 0;
}
```

Enter 10 negative numbers:

-1 -2 -3 **4** -5 -6 -7 -8 -9 -10

ERROR: positive number or zero was entered as the
4th number! Input ends with the 4th number.

4th number was not added in.

-6 is the sum of the first 3 numbers.



Chien-Nan Liu, NCTUEE

3-41

Break vs Continue in Loops

```
int main( )
{
    int count;

    for (count = 1; count <= 10; count++)
    {
        if (count == 5)
            break; // break loop only if x is 5
        cout << count << ' ';
    }

    cout << "break loop at " << count << endl;
    return 0;
}
```

1 2 3 4
break loop at 5

```
int main( )
{
    int count;

    for (count = 1; count <= 10; count++)
    {
        if (count == 5)
            continue; // skip remaining code in loop
        cout << count << ' ';
    }

    cout << "count 5 is skipped" << endl;
    return 0;
}
```

1 2 3 4 6 7 8 9 10
count 5 is skipped



Chien-Nan Liu, NCTUEE

3-42



Overview

3.1 Using Boolean Expressions

3.2 Multiway Branches

3.3 More about C++ Loop Statements

3.4 Designing Loops



Chien-Nan Liu, NCTUEE

3-43



Designing Loops

- Designing a loop involves designing
 - The **body** of the loop
 - The **initializing** statements
 - The **conditions** for ending (or continuing) the loop
- Three general methods to control any loop
 - **Count controlled loops**
 - Repeat many times (**specify the number of iterations**)
 - List Headed By Size (**read in the problem size**)
 - **Ask before iterating**
 - Ended with the answer (**YES or NO**)
 - Ended with a sentinel value (**ex: -1 to finish**)
 - **Exit on flag condition**
 - Ended when a particular flag condition exists (**ex: sum > 100**)
 - Running out of input (**reaching End-Of-File (EOF)**)



Chien-Nan Liu, NCTUEE

3-44



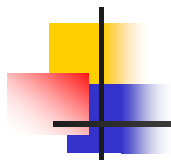
Count Controlled Loops

- Pseudocode containing the line
 repeat the following "this many times"
is often implemented with a for-loop
- A **for-loop** is generally the choice when there is a **predetermined number** of iterations
 - Example:
 for(int count = 1; count <= this_many; count++)
 Loop_body
- **Count controlled loops** are loops that determine the number of iterations **before the loop begins**
 - Specified in the program or given by users



Chien-Nan Liu, NCTUEE

3-45



Example: for-loop for Sum/Product

- Reading a list of numbers and computing the sum

```
int sum = 0;
for (int count=1; count <= this_many; count++) {
    cin >> next;
    sum = sum + next;
}
```

 - *sum* must be initialized to 0 prior to the loop body!
- Forming a product is very similar to the sum example

```
int product = 1;
for (int count=1; count <= this_many; count++) {
    cin >> next;
    product = product * next;
}
```

 - *product* must be initialized prior to the loop body
 - Notice that *product* is **initialized to 1**, not 0!



Chien-Nan Liu, NCTUEE

3-46

List Headed By Size

- We can determine the size of the list beforehand
- The **for-loops** provide a natural implementation of the list headed by size method of ending a loop

```
int items;
cout << "How many items in the list?";
cin >> items;
for (int count = 1; count <= items; count++)
{
    int number;
    cout << "Enter number " << count;
    cin >> number;
    cout << endl;
    // statements to process the number
}
```



Chien-Nan Liu, NCTUEE

3-47

Ask Before Iterating

- Ask if the user wants to continue before each iteration
- A **while loop** is used here to implement the ask before iterating method to end a loop

```
sum = 0;
cout << "Are there numbers in the list (Y/N)?";
char ans;
cin >> ans;
while ((ans == 'Y') || (ans == 'y'))
{
    // statements to read and process the number
    cout << "Are there more numbers (Y/N)? ";
    cin >> ans;
}
```



Chien-Nan Liu, NCTUEE

3-48

List Ended With a Sentinel Value

- Using a particular value to signal the end of the list
- A **while loop** is typically used to end a loop using the list ended with a sentinel value method

```
cout << "Enter a list of nonnegative integers.\n"
    << "Place a negative integer after the list.\n";
sum = 0;
cin >> number;
while (number > 0)
{
    // statements to process the number
    cin >> number;
}
```

- The sentinel value is read, but not processed!



Chien-Nan Liu, NCTUEE

3-49

Exit on Flag Condition

- Loops is ended when a particular flag condition exists
 - A variable that changes value to indicate that some event has taken place is a flag
- Ex: identify a student with a grade of 90 or better

```
int n = 1;
grade = computeGrade(n);
while (grade < 90)
{
    n++;
    grade = computeGrade(n);
}
cout << "Student number " << n
    << " has a score of " << grade << endl;
```



Chien-Nan Liu, NCTUEE

3-50

Correction to the Exit on Flag

- The loop on the previous slide might not stop if no student has a grade of 90 or higher
 - Use a second flag to ensure that there are still students to consider

```
int n = 1;
grade = computeGrade(n);
while ((grade < 90) && (n < numberOfStudents))
{
    // same as before
}
if (grade > 90)
    // same output as before
else
    cout << "No student has a high score";
```



Chien-Nan Liu, NCTUEE

3-51

Running Out of Input

- Using the **eof function** to indicate the end of a file
- The **while loop** is typically used to implement the running out of input method of ending a loop

```
ifstream infile;
infile.open("data.dat");
while (! infile.eof() )
{
    // read and process items from the file
    // File I/O covered in Chapter 6
}
infile.close();
```

- File operations are introduced in Chapter 6



Chien-Nan Liu, NCTUEE

3-52

Nested Loops

- The body of a loop may contain any kind of statement, including another loop
 - When loops are nested, all iterations of **the inner loop are executed for each iteration** of the outer loop
 - Give serious consideration to making the inner loop a function call to make it easier to read your program
- Similar to migrating 1-dimensional problems into multi-dimensional problems
 - One loop: $f(0), f(1), f(2), \dots$
 - Two loops: $f(0,0), f(0,1), \dots, f(0,n), f(1,0), f(1,1), \dots$
- The most important thing:
 - Obtain the **changing rules** of the running index



Chien-Nan Liu, NCTUEE

3-53

Nested Loops: Examples (1/3)

- Execute multi-dimensional operations

```
for (i=1; i<=4; i++) {  
    cout << "i=" << i << ":\n";  
    for (j=1; j<=3; j++) {  
        cout<<i<<"x"<<j<<"="<<i*j;  
    }  
    cout << endl;  
}
```

i=1:
1x1=1 1x2=2 1x3=3
i=2:
2x1=2 2x2=4 2x3=6
i=3:
3x1=3 3x2=6 3x3=9
i=4:
4x1=4 4x2=8 4x3=12

- Please pay special attention to the index changing sequence
 - Column first in this case
(1,1) -> (1,2) -> (1,3) -> (2,1) -> ...
 - So, column is changed in the inner loop



Chien-Nan Liu, NCTUEE

3-54

Nested Loops: Examples (2/3)

- Repeat a loop for n times

```
for (i=1; i<=5; i++) {
    for (j=1; j<=6; j++)
    { cout << "*"; }
    cout << endl;
}
```



- Inner loop control the repeated actions
 - Print 6 stars in this case
- Outer loop control the number of times
 - Print 5 rows of stars in this case



Chien-Nan Liu, NCTUEE

3-55

Nested Loops: Examples (3/3)

- Inner loop is controlled by outer loop

```
for (i=1; i<=5; i++)
{
    for (j=1; j<=i; j++)
    { cout << "*"; }
    cout << endl;
}
```



- Outer loop control the number of rows
 - Print 5 rows of stars in this case
- Inner loop control the number of stars
 - Change its termination condition
 - i=1 --> for (j=1; j<=1; j++) --> 1 star
 - i=2 --> for (j=1; j<=2; j++) --> 2 stars
 -



Chien-Nan Liu, NCTUEE

3-56

break in Nested Loops

- In nested loops, *break/continue* can only affect the most inner loop where the *break/continue* stands

```
for (i=1; i<=5; i++)
{
    for (j=1; j<=3; j++)
    {
        cout << '(' << i << ", " << j << ')';
        if (i==3) break;
    }
    cout << endl;
}
```

break inner
loop j only

```
(1,1) (1,2) (1,3)
(2,1) (2,2) (2,3)
(3,1)
(4,1) (4,2) (4,3)
(5,1) (5,2) (5,3)
```

Jump out the inner
loop and start from here

- If *break* is used to skip the following *switch* statements, it will not affect the outside loop
- One-time use only

```
/* loop until user types end-of-file key sequence */
while ( ( grade = getchar() ) != EOF ) {

    /* determine which grade was input */
    switch ( grade ) { /* switch nested in while */

        case 'A': /* grade was uppercase A */
        case 'a': /* or lowercase a */
            ++aCount; /* increment aCount */
            break; /* necessary to exit switch */
    }
}
```

3-57



Chien-Nan Liu, NCTUEE

Code Example for Nested Loop

```
int numberOfReports;
cout << "How many conservationist reports are there? ";
cin >> numberOfReports;
```

```
int grandTotal = 0, subtotal, count, next;
for (count = 1; count <= numberOfReports; count++)
```

outer loop:
listed head by size

```
{
    cout << endl << "Enter the report of "
        << "conservationist number " << count << endl;
    cout << "Enter the number of eggs in each nest.\n"
        << "Place a negative integer at the end of your list.\n";
    subtotal = 0;
    cin >> next;
```

```
while (next >= 0)
```

```
{
    subtotal = subtotal + next;
    cin >> next;
}
```

inner loop:
sentinel controlled loop

```
    cout << "Total egg count for conservationist "
        << "number " << count << " is "
        << subtotal << endl;
    grandTotal = grandTotal + subtotal;
```

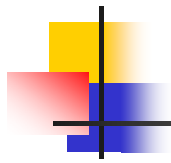
```
}
cout << endl << "Total egg count for all reports = "
    << grandTotal << endl;
```

For each report,
collect number of
eggs in each nest



Chien-Nan Liu, NCTUEE

3-58



Debugging Loops

- Common errors involving loops include
 - **Off-by-one errors** in which the loop executes one too many or one too few times
 - Check the **boundary cases** (start, end)
 - Infinite loops usually result from a mistake in the **Boolean expression** that controls the loop
 - Will the **termination condition** happen eventually?
- Trace the variable to **observe its value change** during execution
 - Many systems include utilities to help with this
 - Extra *cout* statements can be used to trace values



Chien-Nan Liu, NCTUEE

3-59



Fixing Off-by-One Errors

- Check your comparison: should it be $<$ or $<=$?
 - `for (i=0; i<10; i++)` v.s `for (i=0; i<=10; i++)`
 - $i = 0, 1, 2, \dots, 8, 9$ v.s $i = 0, 1, 2, \dots, 8, 9, 10$
- Ensure the **initialization** uses the correct value
 - `sum = 0;` `sum = sum + value;`
 - `prod = 1;` `prod = prod * value;` (what if start with 0?)
- Does the loop handle the **zero iterations** case?
 - If the first input is -1, what is the correct count? 0 or 1 ??

```
int count = 1;
cout << "-1 to finish\n";
cin >> grade;
while (grade >= 0)
```

.....



Chien-Nan Liu, NCTUEE

3-60

Fixing Infinite Loops

- Check the direction of inequalities: $<$ or $>$?
 - `while (grade < 90)` → finish loop when $\text{grade} \geq 90$
 - `while (grade > 90)` → finish loop when $\text{grade} \leq 90$
 - Is the condition for continuity or termination ??
- Test for $<$ or $>$ rather than equality ($==$)
 - `for (i=1; i != 10; i+=2)` → $i = 1, 3, 5, 7, 9, 11, \dots$ (X)
 - `for (i=1; i < 10; i+=2)` → $i = 1, 3, 5, 7, 9$ (O)
 - Remember that doubles are really **only approximations**
 - don't use a float-point number as the loop counter
 - `for (double f=1.0; f != 10; f+=1)` → $f = 9.9998?$ $f = 10.001?$
 - This loop may not stop as expected due to the approximation



Chien-Nan Liu, NCTUEE

3-61

Debugging Example

- The following erroneous code is supposed to obtain the product of the numbers 2 through 5 (Ans:120)
- Add temporary cout statements to trace variables

```
int next = 2, product = 1;
while (next < 5)
{
    next++;
    product = product * next;
}
```



```
int next = 2, product = 1;
while (next < 5)
{
    next++;
    product = product * next;
    cout << "next = " << next
    << "product = "
    << product << endl;
}
```

```
next = 3 product = 3
next = 4 product = 12
next = 5 product = 60
```



Chien-Nan Liu, NCTUEE

3-62

Fixing the Bugs

- The cout statements show that the loop never multiplied by 2
- Solve the problem by moving the statement next++
- Re-testing the loop shows that the loop never multiplies by 5
- The fix is to use <= instead of < in our comparison

```
int next = 2, product = 1;
while (next < 5)
{
    product = product * next;
    next++;
    cout << "next = " << next
        << "product = "
        << product << endl;
}
```

```
next = 3 product = 2
next = 4 product = 6
next = 5 product = 24
```

```
int next = 2, product = 1;
while (next <= 5)
{
    product = product * next;
    cout << "next = " << next
        << "product = "
        << product << endl;
    next++;
}
```

```
next = 2 product = 2
next = 3 product = 6
next = 4 product = 24
next = 5 product = 120
```



Chien-Nan Liu, NCTUEE

Loop Testing Guidelines

- Every time a program is changed, it must be **retested**
 - Changing one part may require a change to another
- Every loop should at least be tested using input to cause:
 - Zero iterations of the loop body
 - One iteration of the loop body
 - One less than the maximum number of iterations
 - The maximum number of iterations
- Be sure that the mistakes is really in the loop
 - **Tracing proper variables** is often required



Chien-Nan Liu, NCTUEE