Programming with C++

COMP2011: Program Flow Control

Cecia Chan
Cindy Li
Brian Mak

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology
Hong Kong SAR, China

Introduction

- So far, our C++ program consists of only the main() function.
- Inside main() is a sequence of statements, and all statements are executed once and exactly once.
- Such sequential computation can be a big limitation on what can be computed. Therefore, we have
  - selection
  - iteration

Part I

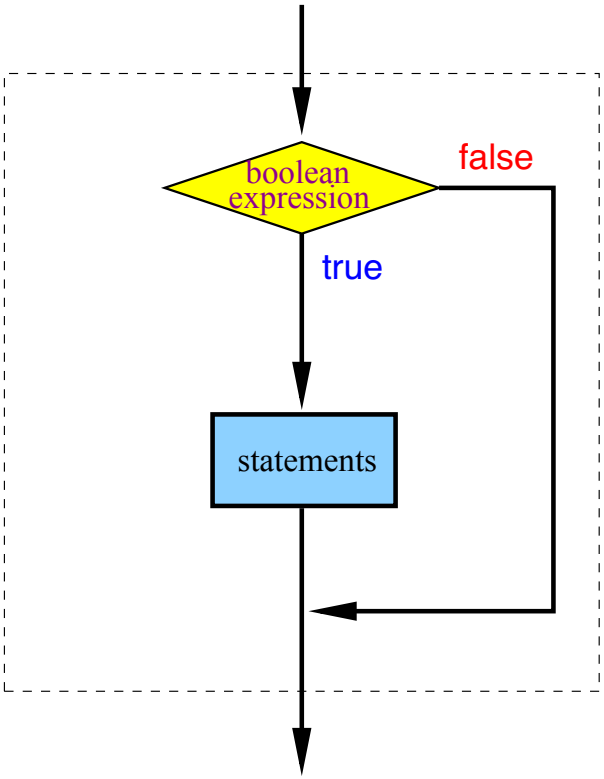
You Have a Choice: **if**

if Statement

Syntax: if Statement

if (<boolean expression>) <statement>

if (<boolean expression>) { <sequence of statements> }



- Example: Absolute value $|x|$ of x.

```
int x;
cin >> x;

if (x < 0)
{
    x = -x;
}
```

Example: To Sort 2 Numbers

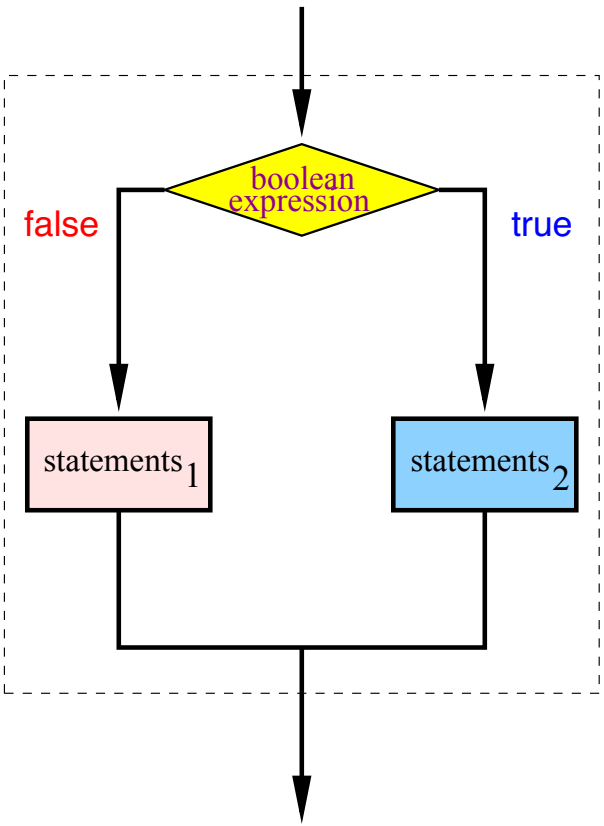
```
#include <iostream>      /* File: swap.cpp */
using namespace std;

int main() /* To sort 2 numbers so that the 2nd one is larger */
{
    int x, y;            // The input numbers
    int temp;            // A dummy variable for manipulation

    cout << "Enter two integers (separated by whitespaces): ";
    cin >> x >> y;

    if (x > y)
    {
        temp = x;        // Save the original value of x
        x = y;           // Replace x by y
        y = temp;        // Put the original value of x to y
    }

    cout << x << '\t' << y << endl;
    return 0;
}
```

if-else Statement

Syntax: if-else Statement

if (<bool-exp>) <stmt> else <stmt>

if (<bool-exp>) { <stmts> } else { <stmts> }



- Example: To find the larger value.

```
int x, y, larger;

cin >> x >> y;

if (x > y)
    larger = x;
else
    larger = y;
```

if-else-if Statement

Syntax: if-else-if Statement

if (<bool-exp>) <stmt>
else if (<bool-exp>) <stmt>
else if (<bool-exp>) <stmt>
⋮
else < *stmt* >

if (<bool-exp>) { <stmts> }
else if (<bool-exp>) { <stmts> }
else if (<bool-exp>) { <stmts> }
⋮
else { <stmts> }

Example: Conversion to Letter Grade

```
#include <iostream>      /* File: if-elseif-grade.cpp */
using namespace std;

int main()              /* To determine your grade (fictitious) */
{
    char grade;             // Letter grade
    int mark; cin >> mark; // Numerical mark between 0 and 100

    if (mark >= 90)
        grade = 'A';    // mark >= 90
    else if (mark >= 60)
        grade = 'B';    // 90 > mark >= 60
    else if (mark >= 20)
        grade = 'C';    // 60 > mark >= 20
    else if (mark >= 10)
        grade = 'D';    // 20 > mark >= 10
    else
        grade = 'F';    // 10 > mark

    cout << "Your letter grade is " << grade << endl;
    return 0;
}
```

Relational Operators

Math	C++	Meaning
$=$	==	equal to
$<$	<	less than
$\leq$	<=	less than or equal to
$>$	>	greater than
$\geq$	>=	greater than or equal to
$\neq$	!=	not equal to

- Relational operators are used to compare two values.
- The result is boolean indicating if the relationship is true or false.
- Don't mix up the 2 following different expressions:

```
x = y      // This is an assignment
x == y     // This is an equality comparison
```

Logical Operators

- Logical operators are used to modify or combine boolean values.
- C++ has 3 logical operators:
  - !: logical NOT
  - ||: logical OR
  - &&: logical AND
- Boolean values
  - true: internally represented by 1; ANY non-zero number is also considered true
  - false: internally represented by 0

p	q	!p	p && q	p || q
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

Precedence and Associativity of Boolean Operators

Operator	Description	Associativity
( )	parentheses	—
++ −− ! −	increment, decrement, logical NOT, unary minus	Right-to-Left
* / %	multiply, divide, mod	Left-to-Right
+ −	add, subtract	Left-to-Right
> >= < <=	relational operator	Left-to-Right
== !=	equal, not equal	Left-to-Right
&&	logical AND	Left-to-Right
||	logical OR	Left-to-Right
=	assignment	Right-to-Left

- Operators are shown in decreasing order of precedence.
- When you are in doubt of the precedence or associativity, use extra parentheses to enforce the order of operations.

Quiz

What is the value of each of the following boolean expressions:

- $4 == 5$
- $x > 0$ && $x < 10$ /* if int x = 5 */
- $5 * 15 + 4 == 13$ && $12 < 19$ || $!\text{false} == 5 < 24$
- true && false || true
- x /* if int x = 5 */
- $x++ == 6$ /* if int x = 5 */
- $x = 9$
- $x == 3 == 4$ /* assume that x is an int */

= != ==

- Both x = y and x == y are valid C++ expressions
  - `x = y` is an assignment expression, assigning the value of y to x. The expression has a result which is the final value of x. (That is why the cascading assignment works.)
  - `x == y` is a boolean expression, testing if x and y are equal, and the result is either true or false.
- But since C++ also interprets integers as boolean, so
  - in `if (x = 3) { <stmts> }`, <stmts> are always executed because (x = 3) evaluates to 3 — a non-zero value — which is interpreted as true.
  - in `if (x = 0) { <stmts> }`, <stmts> are always NOT executed because (x = 0) evaluates to 0 which is interpreted as false.
- It is not recommended to use an assignment expression as a boolean expression.

if-else Operator: ?:

Syntax: if-else Expression

(<bool-exp>) ? <then-exp> : <else- exp>;

- The ternary if-else operator: ?: is used.
- Unlike an if-else statement, an if-else expression has a value!

Example

```
/* Example: get the larger of two numbers */
larger = (x > y) ? x : y;

/* Example: get the letter grade from the percentile */
grade = (percentile >= 85) ? 'A'
        : ((percentile >= 60) ? 'B'
           : ((percentile >= 15) ? 'C'
              : ((percentile >= 5) ? 'D': 'F'
                )
             )
          );
```

Nested if

- In the if or if-else statement, the $<$ *stmts* $>$ in the if-part or else-part can be any statement, including another if or if-else statement. In the latter case, it is called a nested if statement.
- "Nested" means that a complete statement is inside another.

```
if (condition1)
{
    if (condition2)
        if (condition3)
            cout << "conditions 1,2,3 are true." << endl;
        else
            cout << "conditions 1,2 are true." << endl;
    else
        cout << "condition1 true; condition2 false." << endl;
}
```

"Dangling else" Problem

What is the value of x after the following code is executed?

Program code:

```
int x = 15;

if (x > 20)
if (x > 30)
x = 8;
else
x = 9;
```

Interpretation 1:

```
int x = 15;

if (x > 20)
{
    if (x > 30)
        x = 8;
    else
        x = 9;
}
```

Interpretation 2:

```
int x = 15;

if (x > 20)
{
    if (x > 30)
        x = 8;
}
else
    x = 9;
```

"Dangling else" Problem ..

- C++ groups a dangling else with the most recent if.
- Thus, for the code in the previous page, interpretation 1 is used.
- It is a good programming practice to use extra braces "{ } "
  - to control how your nested if statements should be executed.
  - to clarify your intended meaning, together with proper indentation.

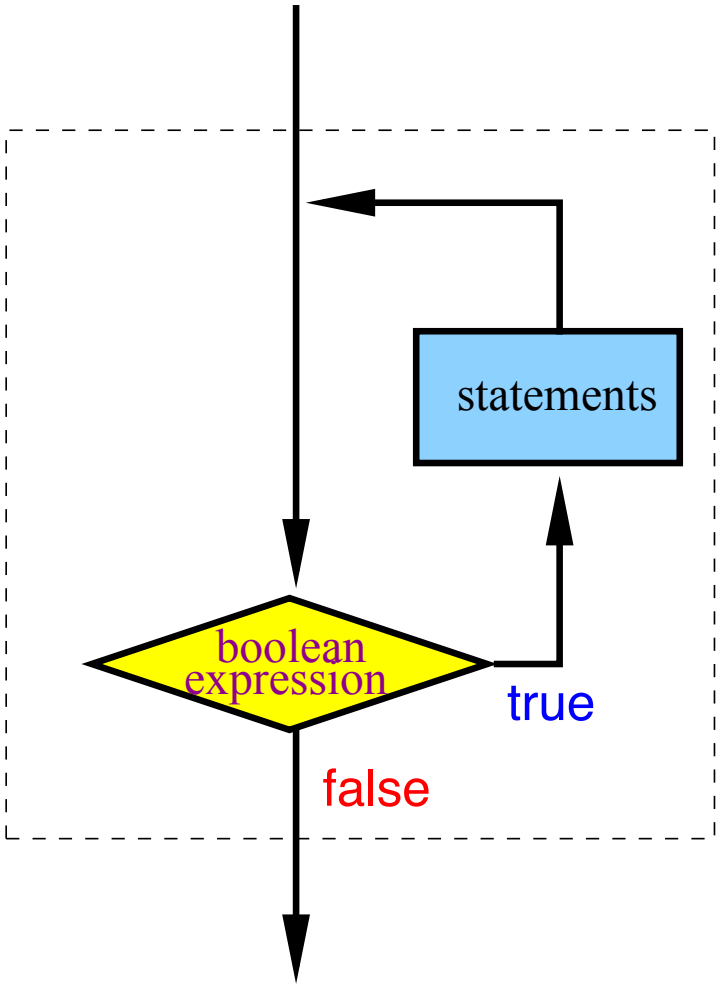
Part II

Loops or Iterations

while Loop (Statement)

Syntax: while Statement

while (<bool-exp>) { <stmts> }



- <stmts> will be repeated as long as the value of <bool-exp> is true.
- As usual, <stmts> can be a single statement, or a sequence of statements (including another while statement), or even no statement!
- What does `while (x > 0) ;` do?
- In general, while statement only makes sense if the value of <bool-exp> may be changed by <stmts> inside the while loop.

Example: Factorial using while Loop

```
#include <iostream>     /* File: while-factorial.cpp */
using namespace std;

/* To compute x! = x(x-1)(x-2)...1, where x is a non -ve integer */
int main()
{
    int factorial = 1;
    int number;

    cout << "Enter a non-negative integer: ";
    cin >> number;

    while (number > 0)
    {
        factorial *= number; // Same as: factorial = factorial*number
        --number;            // Same as: number = number-1
    }

    cout << factorial << endl;
    return 0;
}
```

Example: Factorial using while Loop ..

(assume the user enters 4 for the variable *number*)

Iteration	factorial	number	(number $>$ 0)
0	1	4	true
1	4	3	true
2	12	2	true
3	24	1	true
4	24	0	false

Example: Find the Maximum using while Loop

```
#include <iostream>     /* File: while-max.cpp */
using namespace std;

// To find the maximum of a list of +ve integers. Stop by inputting a
// character that is not a digit. Assume there is at least one number.
int main()
{
    cout << "Enter a number: ";
    int x; cin >> x;    // Input integers

    int max = x;        // Result initialized with the first number

    cout << "Enter the next number: ";
    while (cin >> x)    // If there is input, cin returns TRUE else FALSE
    {
        if (x > max)
            max = x;
        cout << "Enter the next number: ";
    }

    cout << endl << "The maximum number = " << max << endl;
    return 0;
}
```

A Good Programming Practice on Loops

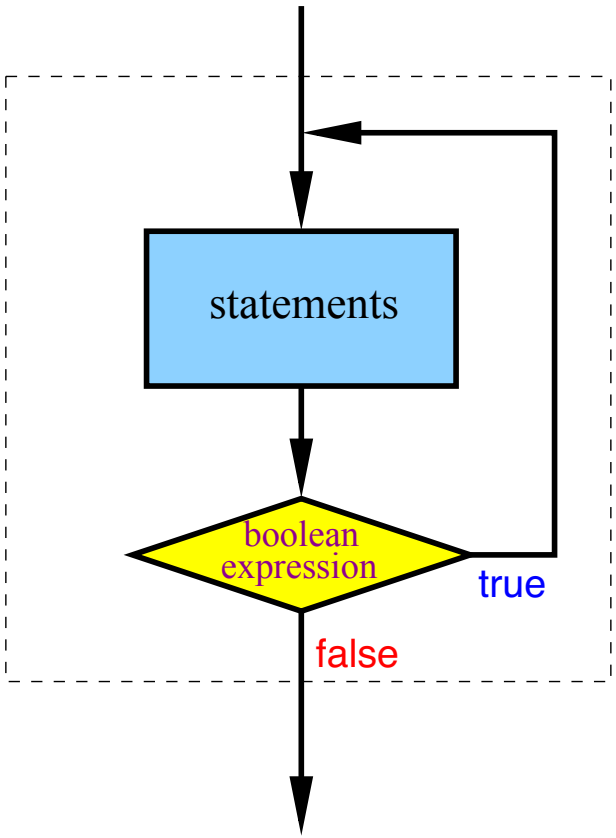
After you have written the codes for a loop, try verifying the following cases:

- The first iteration.
- The second iteration.
- The last iteration.
- Do you know exactly how many iterations will be performed?
- How can the loop terminate? Otherwise, you have an infinite loop! And the program runs forever!

do-while Loop (Statement)

Syntax: do-while Statement

do { <stmts> } while (<bool-exp>);



- Again, like the while statement, <stmts> will be repeated as long as the value of <bool-exp> is true.
- However, unlike the the while statement, the <bool-exp> is evaluated after <stmts> at the bottom of do-while statement.
- That means, <stmts> in do-while loop will be executed at least once, whereas <stmts> in while loop may not be executed at all.

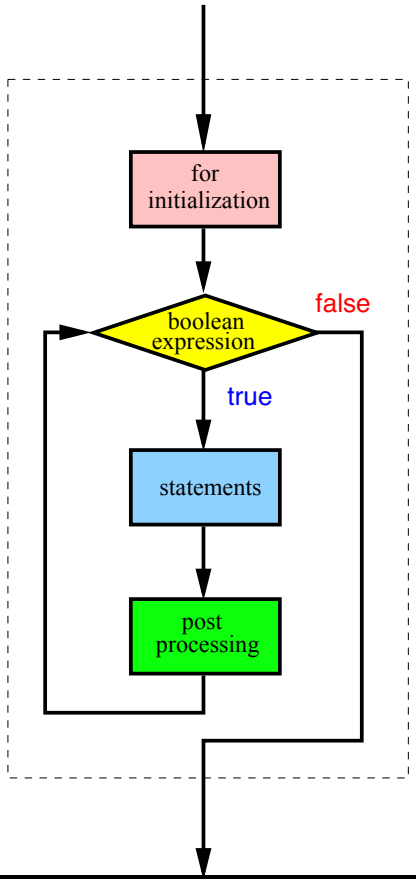
Example: Factorial using do-while Loop

```
#include <iostream>  /* File: do-factorial.cpp */
using namespace std; // Compute x! = x(x-1)(x-2)...1; x is non -ve

int main()
{
    int factorial = 1, number;
    cout << "Enter a non-negative integer: ";
    cin >> number;

    if (number > 0)
    {
        do
        {
            factorial *= number; // Same as: factorial = factorial*number
            --number;            // Same as: number = number-1
        } while (number > 1);
    }

    cout << factorial << endl;
    return 0;
}
```

for Loop (Statement)

Syntax: for Statement

for (<for-initialization> ; <bool-exp> ; <post-processing>)
{ <stmts> }



- for statement is a generalization of the while statement. The idea is to control the number of iterations, usually by a counter variable.
- <for-initialization> sets up the initial values of some variables, usually a counter, before executing <stmts>.
- <stmts> are iterated as long as <bool-exp> is true.
- At the end of each iteration, <post-processing> will be executed. The idea is to change some values, again usually the counter, so that <bool-exp> may become false.

Example: Factorial using for Loop

```
#include <iostream>     /* File: for-factorial.cpp */
using namespace std;

/* To compute x! = x(x-1)(x-2)...1, where x is a non -ve integer */
int main()
{
    int factorial = 1;
    int number;

    cout << "Enter a non-negative integer: ";
    cin >> number;

    for (int j = 1; j <= number; ++j) // Set up a counter to iterate
        factorial *= j;

    cout << number << "! = " << factorial << endl;
    return 0;
}
```

Example: x^n using for Loop

```
#include <iostream>      /* File: for-power.cpp */
using namespace std;

/* To compute x^n, where x and n are integers, and n >= 0 */
int main()
{
    int x;
    int n;               // Power or exponent
    int result = 1;      // Need to initialize it to 1. Why?

    cout << "Enter a number followed by its +ve power: ";
    cin >> x >> n;

    if (n < 0)
        cerr << "Error: n < 0!" << endl;
    else
    {
        for (int j = 1; j <= n; j++)
            result *= x;

        cout << x << " to the power of " << n << " = " << result << endl;
    }

    return 0;
}
```

Remarks on for Statement

- Notice that the variable *j* in the above 2 examples are only defined inside the for loop. When the loop is done, *j* disappears, and you cannot use that *j* anymore.
- Don't mis-type a ";" after the first line of the for loop. E.g., what is the result of the following code?

```
for (int j = 1; j <= n; j++);
    result *= x;
```

- while statement is a special case of for statement. How can you simulate while using for?
- Sometimes, if the for-body is short, you may even further compact the code as follows:

```
for (int j = 1; j <= number; factorial *= j++)
    ;
```

Which Loop to Use?

for loop :

- When you know how to specify the required number of iterations.
- When the counter variable is also needed for computation inside the loop.
- e.g. To compute sums, products, and to count.

while loop :

- You want to repeat an action but do not know exactly how many times it will be repeated.
- The number of iterations is determined by a boolean condition. e.g.

```
while (cin >> x) { ... }
```

do-while loop :

- The associated actions have to be executed at least once.
- Otherwise, do-while and while are used in similar situations.

Part III

Nested Looooooops

Nested Loops Example: Compute Average Score

One may put a while loop inside another while loop.

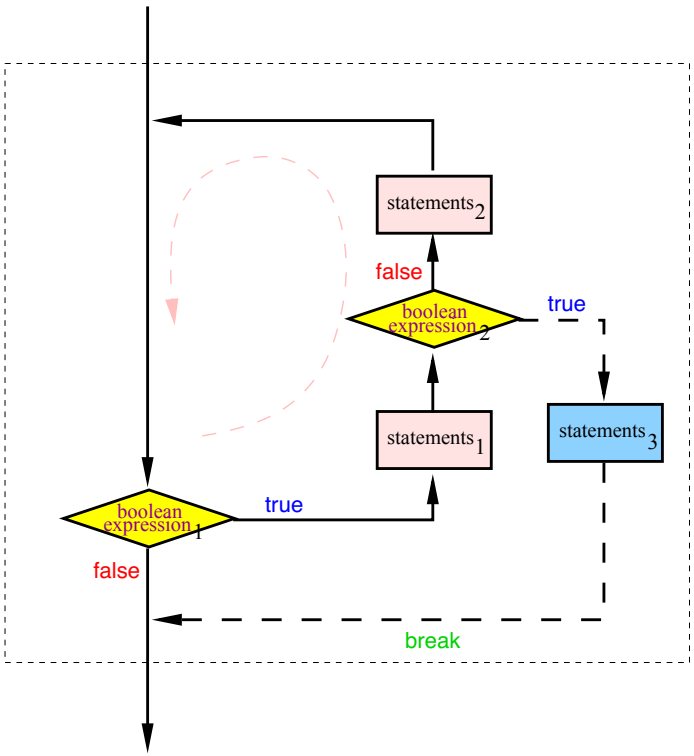
```
#include <iostream>                                          /* File: nested-while-avg.cpp */
using namespace std;

int main( )
{
    int NUM_ASSIGNMENTS = 5;                                 // Uppercase variable doesn't change
    int j;                                                   // Assignment counter
    int score, sum_of_scores;
    char reply = 'y';                                        // 'y' for yes, 'n' for no; initialized to yes

    cout << "Enter scores for the first student?  (y/n) " << endl;

    while ((cin >> reply) && (reply == 'y' || reply == 'Y'))
    {
        sum_of_scores = 0;                                   // Reset the accumulator to zero
        j = 1;                                               // Reset the assignment counter to 1

        while (j <= NUM_ASSIGNMENTS)
        {
            cout << "Enter student's score for assignment #" << j << " :  ";
            cin >> score;                                    // Remark: one should check input errors here
            sum_of_scores += score;
            j++;
        }

        cout << "The average score = " << sum_of_scores/NUM_ASSIGNMENTS << endl;
        cout << "Enter scores for another student?  (y/n) " ;
    }

    return 0;
}
```

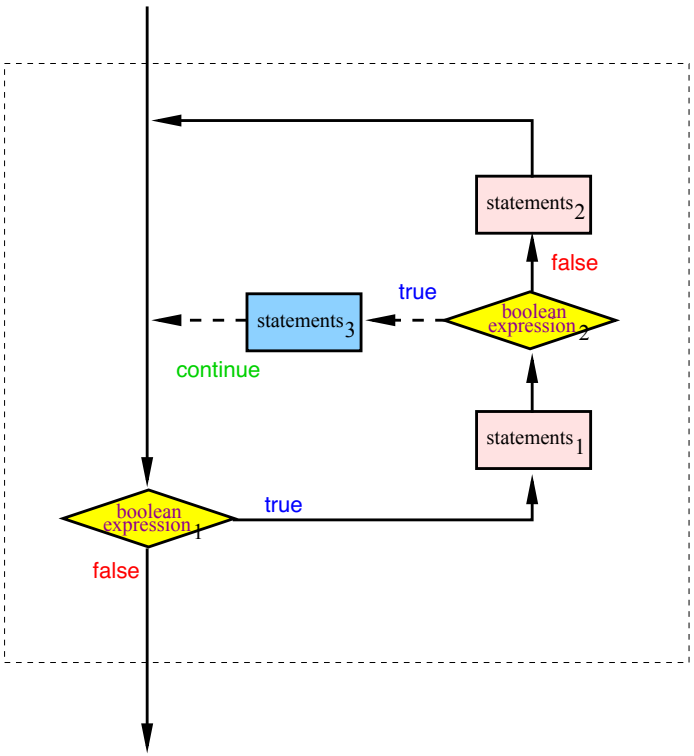
Nested Loops Example: Multiplication Table

```
#include <iostream>     /* File: multiplication-table.cpp */
#include <iomanip>      // a library that helps control input/output formats
using namespace std;

int main()
{
    // To print out products of j*k where j, k = 1,...,10
    for (int j = 1; j <= 10; ++j)
    {
        for (int k = 1; k <= 10; ++k) // Reset k=1 for each j. Why?
            cout << setw(4) << j*k;   // Set the length of output field to 4

        cout << endl;
    }

    return 0;
}
```

break and continue

- A break causes the innermost enclosing loop to exit immediately.
- A continue causes the next iteration of the enclosing loop to begin.
- That is, in the while loop, control passes to test the boolean expression again immediately.





Example: Stop Inputs with break

```
#include <iostream>          /* File: break-avg.cpp */
using namespace std;

int main()
{
    int NUM_ASSIGNMENTS = 5; // Uppercase variable doesn't change
    int j;                   // Assignment counter
    int score, sum_of_scores;
    char reply = 'y';        // 'y' for yes, 'n' for no; initialized to yes

    cout << "Enter scores for the first student? (y/n) " << endl;
    while ((cin >> reply)  &&  (reply == 'y' || reply == 'Y'))
    {
        sum_of_scores = 0;   // Reset the accumulator to zero
        j = 1;               // Reset the assignment counter to 1

        while (j <= NUM_ASSIGNMENTS)
        {
            cout << "Enter student's score for assignment #" << j << " : ";
            cin >> score;    // Remark: one should check input errors here

            if (score < 0)
                break;

            sum_of_scores += score;
            j++;
        }
        cout << "The average score = " << sum_of_scores/NUM_ASSIGNMENTS << endl;
        cout << "Enter scores for another student? (y/n) " ;
    }
    return 0;
} // Question: What is the output with the input: 4, 5, -6, 7, 8?
```

Example: Ignore Negative Inputs with continue

```
#include <iostream>     /* File: continue-avg.cpp */
using namespace std;

int main()
{
    int NUM_ASSIGNMENTS = 5; // Uppercase variable doesn't change
    int j;                   // Assignment counter
    int score, sum_of_scores;
    char reply = 'y';        // 'y' for yes, 'n' for no; initialized to yes

    cout << "Enter scores for the first student? (y/n) " << endl;
    while ((cin >> reply)  &&  (reply == 'y' || reply == 'Y'))
    {
        sum_of_scores = 0;   // Reset the accumulator to zero
        j = 1;               // Reset the assignment counter to 1

        while (j <= NUM_ASSIGNMENTS)
        {
            cout << "Enter student's score for assignment #" << j << " : ";
            cin >> score;    // Remark: one should check input errors here

            if (score < 0)
                continue;

            sum_of_scores += score;
            j++;
        }
        cout << "The average score = " << sum_of_scores/NUM_ASSIGNMENTS << endl;
        cout << "Enter scores for another student? (y/n) " ;
    }
    return 0;
} // Question: What is the output with the input: 4, 5, -6, 7, 8 ?
```

Example: Difference between break and continue

```
/* File: break-example.cpp */
#include <iostream>
using namespace std;

int main()
{
    int j = 0;

    while (j < 3)
    {
        cout << "Enter iteration "
             << j << endl;

        if (j == 1)
            break;

        cout << "Leave iteration "
             << j << endl;
        j++;
    }

    return 0;
}
```

```
/* File: continue-example.cpp */
#include <iostream>
using namespace std;

int main()
{
    int j = 0;

    while (j < 3)
    {
        cout << "Enter iteration "
             << j << endl;

        if (j == 1)
            continue;

        cout << "Leave iteration "
             << j << endl;
        j++;
    }

    return 0;
}
```

Question: What are the outputs of the 2 programs?

Where Does continue; Continue in a for Loop?

```
#include <iostream>      /* File: for-continue.cpp */
using namespace std;

int main()
{
    for (int j = 1; j <= 10; j++)
    {
        cout << "j = " << j << endl;

        if (j == 3)
        {
            j = 8;
            continue;   // What if it is replaced by break;
        }
    }

    return 0;
}
```

Common Loop Errors

What is the error in each of the following cases?

- Case 1:

```
int sum;
while (cin >> x)
    sum += x;
```

- Case 2:

```
int j;
while (j < 10)
{
    cout << "hello again!" << endl;
    j++;
}
```

- Case 3:

```
int j = 0;
while (j < 10);
{
    cout << "hello again!" << endl;
    j++;
}
```