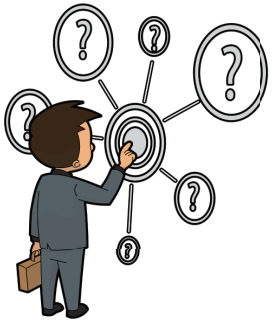
COMP 2012H Honors Object-Oriented Programming and Data Structures

Topic 3: Program Flow Control

Dr. Desmond Tsoi

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

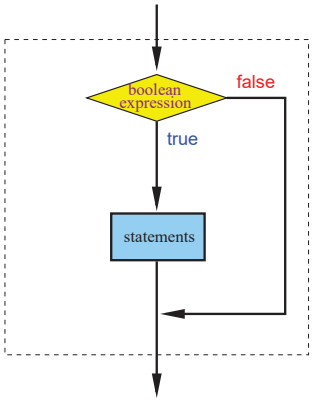
You Have a Choice: **if**

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

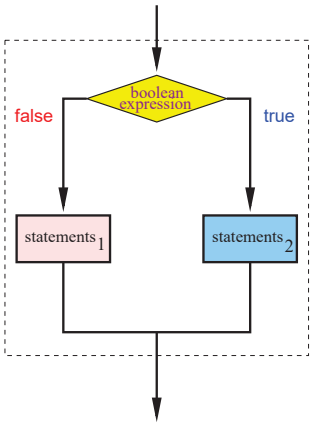
```
#include <iostream>      /* File: swap.cpp */
using namespace std;

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;            // Numerical mark between 0 and 100
    cin >> mark;

    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 */
- $5 * 15 + 4 == 13$ && $12 < 19$ || !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:

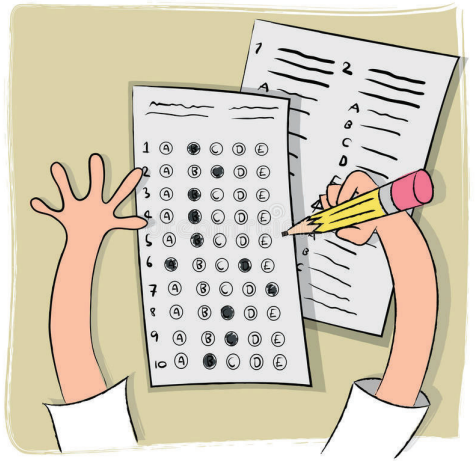
```
int x = 15;

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.

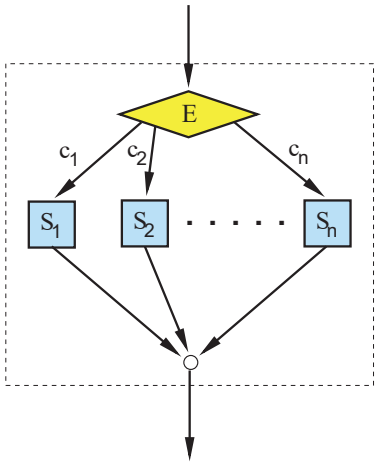
Part II

Let's **switch**: C++ Multiple Choices



switch Statement

switch statement is a variant of the if-else-if statement, that allows multiple choices based on the value of an integral expression.



Syntax: switch Statement

```
switch (integral expression)
{
    case constant-1:
        statement-sequence-1;
        break;
    case constant-2:
        statement-sequence-2;
        break;
    ...
    case constant-N:
        statement-sequence-N;
        break;
    default: // optional
        statement-sequence-(N+1);
}
```


Example: switch on Integers

```
#include <iostream>      /* File: switch-find-comp2012h-instructor.cpp */
using namespace std;

int main()               // To determine your instructor
{
    cout << "Enter the COMP2012H section number to find its instructor: ";
    int section;         // COMP2012H section number: should be 1, 2, 3, or 4
    cin >> section;      // Input COMP2012H section number

    switch (section)
    {
        case 1:
            cout << "Sergey Brin" << endl; break;
        case 2:
            cout << "Bill Gates" << endl; break;
        case 3:
            cout << "Steve Jobs" << endl; break;
        case 4:
            cout << "Jeff Bezos" << endl; break;
        default:
            cerr << "Error: Invalid lecture section " << section << endl;
            break;
    }

    return 0;
}
```

Example: switch on Characters

```cpp
#include <iostream>     /* File: switch-char-bloodtype.cpp */
using namespace std;

int main()              // To find out who may give you blood
{
    cout << "Enter your blood type (put 'C' for blood type AB): ";
    char bloodtype; cin >> bloodtype;

    switch (bloodtype)
    {
        case 'A':
            cout << "Your donor must be of blood type: O or A\n";
            break;
        case 'B':
            cout << "Your donor must be of blood type: O or B\n";
            break;
        case 'C':
            cout << "Your donor must be of blood type: O, A, B, or AB\n";
            break;
        case 'O':
            cout << "Your donor must be of blood type: O";
            break;
        default:        // To catch errors
            cerr << "Error: " << bloodtype << " is not a valid blood type!\n";
            break;
    }
    return 0;
}
```


Example: switch with Sharing Cases

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

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

    switch (mark/10)
    {
        case 10:        // Several cases may share the same action
        case 9:
            grade = 'A'; break; // If mark >= 90
        case 8: case 7: case 6: // May write several cases on 1 line
            grade = 'B'; break; // If 90 > mark >= 60
        case 5:
        case 4:
        case 3:
        case 2:
            grade = 'C'; break; // If 60 > mark >= 20
        case 1:
            grade = 'D'; break; // If 20 > mark >= 10
        default:
            grade = 'F'; break;
    }

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


Remarks on switch

- The expression for switch must evaluate to an integral value (integer, char, bool in C++).
- NO 2 cases may have the same value.
- On the other hand, several cases may share the same action statements.
- When a case constant is matched, the statements associated with the case are executed until either
  - a break statement.
  - a return statement.
  - the end of the switch statement.
- Difference between a switch statement and a if-else-if statement:
  - switch statement can only test for equality of the value of one quantity.
  - each expression of the if-else-if statement may test the truth value of different quantities or concepts.

Example: Give me a break

```cpp
#include <iostream>     /* File: switch-no-break.cpp */
using namespace std;

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

    /* What happens if you forget to break? What is the output? */
    switch (mark/10)
    {
        case 10: case 9:
            cout << "Your grade is A" << endl;
        case 8: case 7: case 6:
            cout << "Your grade is B" << endl;
        case 5: case 4: case 3: case 2:
            cout << "Your grade is C" << endl;
        case 1:
            cout << "Your grade is D" << endl;
        default:
            cout << "Your grade is F" << endl;
    }

    return 0;
}
```

New Data Types with enum

- One way to define a new data type is to use the keyword enum.

Syntax: enum Declaration

enum new-datatype { identifier1 [=value1], identifier2 [=value2], ··· };

Example

```
enum weekday { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
               SATURDAY, SUNDAY };          // 0,1,2,3,4,5,6

enum primary_color { RED = 1, GREEN, BLUE }; // 1,2,3

enum bloodtype { A, B, AB = 10, O };         // 0,1,10,11
```


User-defined enum Type

- An enumeration is a type that can hold a finite set of symbolic objects.
- The symbolic (meaningful) names of these objects follow the same rule as identifier names.
- The symbolic names make your program easier to read/understand.
- Internally, these objects are represented as integers.
- By default, the first object is given the value zero, then each subsequent object is assigned a value one greater than the previous object's value.
- The integral values of the enumerated objects may be assigned other integral values by the programmer.
- Thus, the objects of an enum type act like named integer constants.

Example: enum with switch

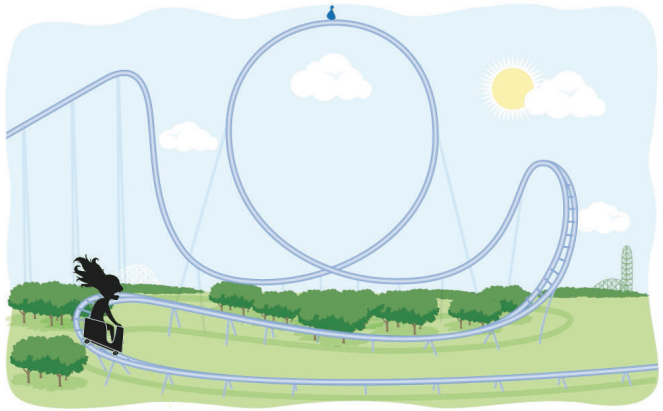
```
#include <iostream>     /* File: enum-shapes.cpp */
using namespace std;

int main()
{
    enum shapes { TEXT, LINE, RECT, CIRCLE };
    cout << "supported shapes: "
         << " TEXT = " << TEXT << " LINE = " << LINE
         << " RECT = " << RECT << " CIRCLE = " << CIRCLE << endl;
    int myshape; // Why the type of myshape is not shape?
    cin >> myshape;

    switch (myshape)
    {
        case TEXT:
            cout << "Call a function to print text" << endl; break;
        case LINE:
            cout << "Call a function to draw a line" << endl; break;
        case RECT:
            cout << "Call a function to draw a rectangle" << endl; break;
        case CIRCLE:
            cout << "Call a function to draw a circle" << endl; break;
        default:
            cerr << "Error: Unsupported shape" << endl; break;
    }

    return 0;
}
```


Example: Mixing Colors

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

int main()
{   // Declare color variables immediately after the enum definition
    enum color { RED, GREEN, BLUE, YELLOW, CYAN, PURPLE } x, y;
    int xint, yint;     // Input variables for the color variables

    cin >> xint >> yint;
    x = static_cast<color>(xint); // Convert an int to a color quantity
    y = static_cast<color>(yint); // Convert an int to a color quantity

    if ( (x == RED && y == GREEN) || (y == RED && x == GREEN) )
        cout << YELLOW << endl;

    else if ( (x == RED && y == BLUE) || (y == RED && x == BLUE) )
        cout << PURPLE << endl;

    else if ( (x == GREEN && y == BLUE) || (y == GREEN && x == BLUE) )
        cout << CYAN << endl;

    else
        cerr << "Error: only support mixing RED/GREEN/BLUE!" << endl;

    return 0;
} // Check what is really printed out
```

Part III

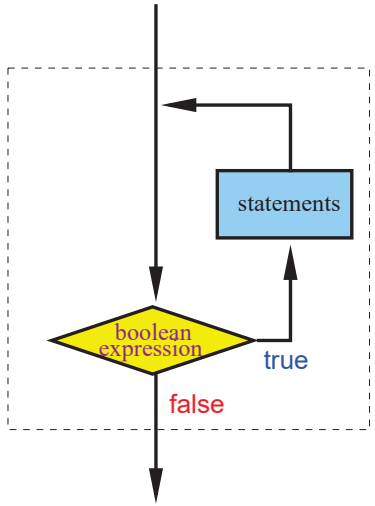
Loops or Iterations



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.

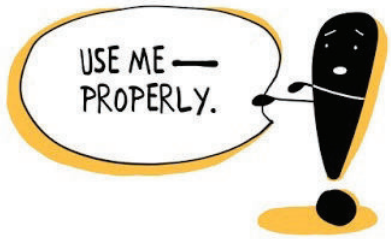
Example: Factorial using while Loop

```
#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



A Good Programming Practice on Loops

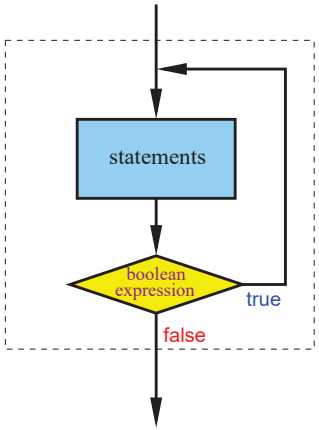
After you have written the codes for a loop, try verifying the following cases:

- 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.

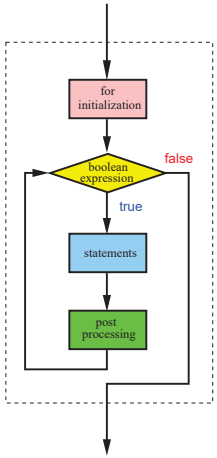
Example: Factorial using do-while Loop

```
#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;
}
```


Remarks on for Statement

- Notice that the variable j in the above example is 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.

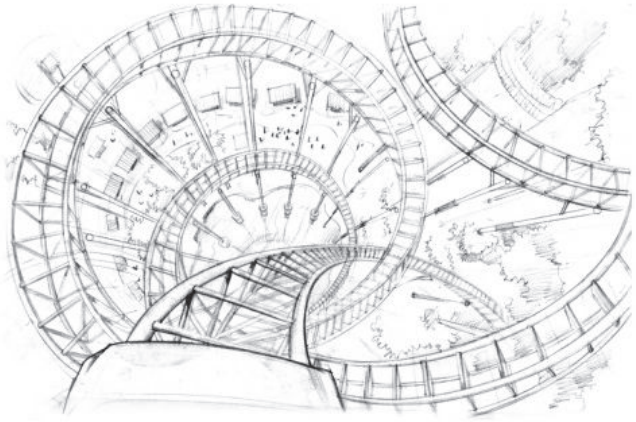
while loop :
- You want to repeat an action but do not know exactly how many times it will be repeated.
- The number of iterations is determined by a boolean condition. e.g.

```
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 IV

Nested Looooooops

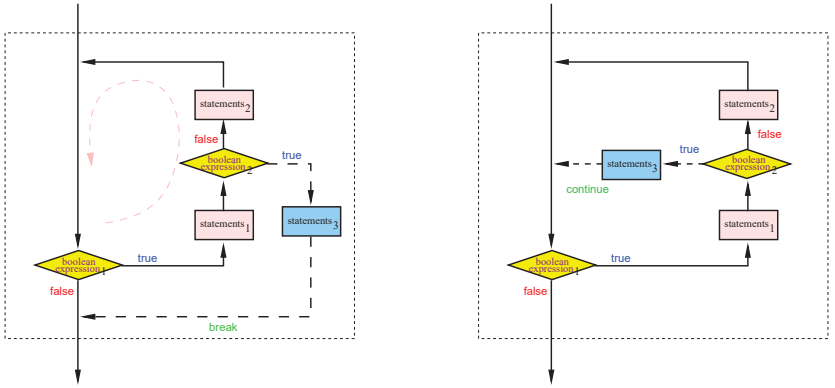


Nested Loops Example: Multiplication Table

```cpp
#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: Difference between break and continue

```cpp
/* 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;
}
```

```cpp
/* 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?

```cpp
#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++;
}
```


That's all!

Any questions?