CS2311 Computer Programming

LT5: Flow Control

Loops/Iterations

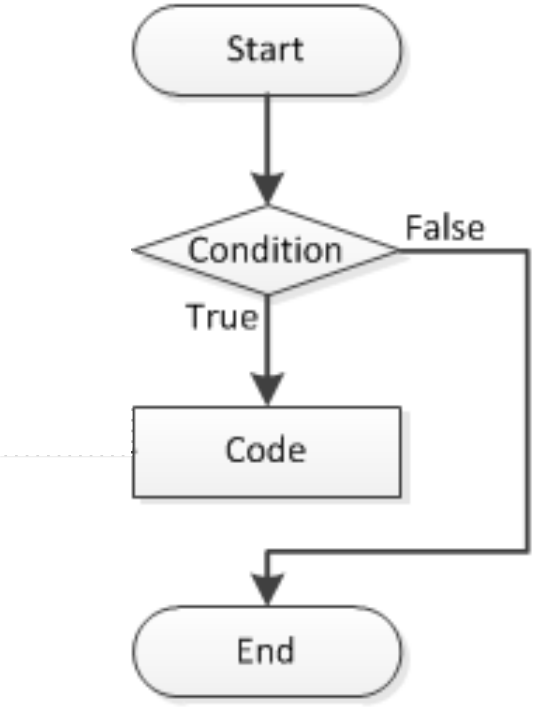
Loop

- When the execution enters a **loop**, it executes a block of code repeatedly until the condition is **NOT** met

- Beside ***sequential*** and ***branch*** execution, loop is another common control flow

- **`while`** loop

```
while(condition) {
    doAction;
}
```



Loops

- In general, a program loop consists of:
  - **Initialization** statements
  - **Body**
  - Exit/Loop **condition**
  - **Post** loop statements (stepping forward to exit loop condition)

An Example to use `while`

- Ensure a positive number input?

```
#include <iostream>
using namespace std;
int main() {
   int     cnt = 0, n;
   float   max, x;

   cout << "The maximum value will be computed." << endl;
   cout << "How many numbers do you wish to enter? ";
   cin >> n;
   while (n <= 0) {  // ensure a positive number is entered
      cout << "ERROR: Positive integer required." << endl << endl;
      cout << "How many numbers do you wish to enter? ";
      cin >> n;
   }
   // to be continued on next page
   return 0;
}
```

An Example to use `while`

- Repetitive execution based on user's input

```
int main() {
   // :
   // continue from last page
   cout << "Enter " << n << " real numbers:  ";
   cin >> x;  // read 1st number
   max = x;
   // pick the largest number in while-loop
   while (++cnt < n) {
      cin >> x;  // read another number
   // cout << "The " << cnt << "-th num is: " << x << endl;
      if (max < x)
         max = x;
   }
   cout << "Maximum value: " << max << endl;

   return 0;
}
```

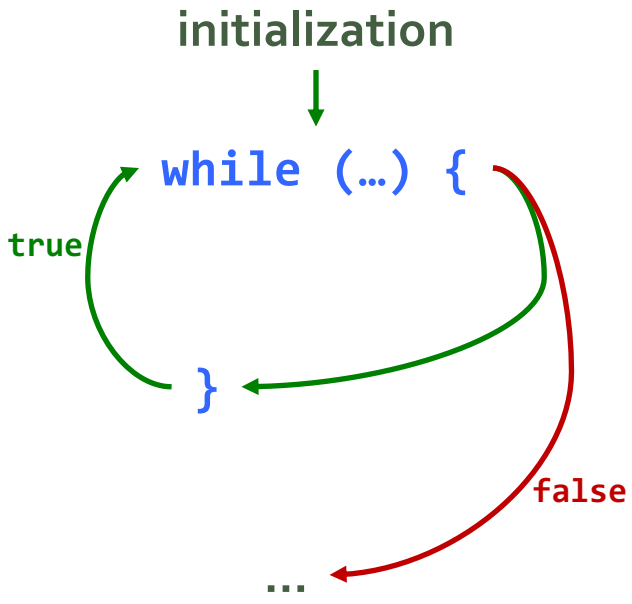
do while statement

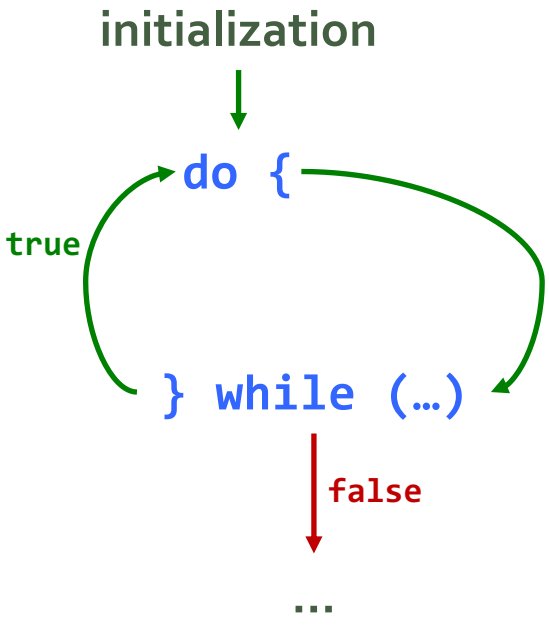
- General form of **`do while`**

```
do
    statement(s);
while (expression);
```

- Semantics:
  - ***statement*** is executed first; thus the loop body is **run at least once**
  - If the value of ***expression*** is non-zero (true), the loop repeats; otherwise, the loop terminates

while vs do .. while





Example of do..while

```
bool error;
int n;
do {
   cout << "Input a positive integer: ";
   cin >> n;
   if (error = (n <= 0))
      cout << "Error: non-positive input! " << endl;
} while (error);
...
```

for-loop statement

```
expr1;
while(expr2) {
    loop statements;
    expr3;
}
```

```
for(expr1; expr2; expr3) {
    loop statements;
}
```

The loop statements is executed as long as **expr2** is *true*.

When **expr2** becomes *false*, the loop ends (e.g., ***i<10***).

expr1: Executed before entering the loop. Often used for variable initialization (e.g., ***int i=0***).

expr3: For each iteration, **expr3** is executed after executing the loop body. Often used to update the counter variables (e.g., ***i++***).

Examples of for

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

int main() {
   for (int i=0; i<10; i++)
      cout << i << endl;
   return 0;
}
```

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

int main() {
   for (int i=0; i<10; i++) {
      if (i%2 == 0)
         cout << i << endl;
   }
   return 0;
}
```

index variable

for Loop: Syntax Error

```
for (int k = 1; k <= 8; k++) {
   cout << "log(" << k << ") = "
      << log(double(k)) << endl;
}
cout << k << endl;  // out of scope; SYNTAX ERROR
```

The variable k is declared within the for-loop statement. It is not visible/accessible outside the for-loop.

```
// Variable k can be declared before the for-loop
int k = 0;
for (k = 1; k <= 8; k++) {
   cout << "sqrt(" << k << ") = "
      << sqrt(k) << endl;
}
cout << k << endl;
```

Poor Practice

Usually the index variable k acts as a counter; so don't access it outside the loop.

Example

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

int main() {

   int x;
   cout << "Enter a number: ";
   // for-loop;
   for (cin >> x; x <= 0; cin >> x) {
      cout << "Input must be positive." << endl;
      cout << "Enter number: ";
   }
   // the while-loop equivalent to the above for-loop:
   cout << "Enter a number: ";
   cin >> x;
   while (x <= 0) {
      cout << "Input must be positive." << endl;
      cout << "Enter number: ";
      cin >> x;
   }
   cout << "The square of " << x << " is " << x*x << endl;
   return 0;
}
```

```
Enter a number: -3
Input must be positive.
Enter number: 2
Enter a number: -3
Input must be positive.
Enter number: 2
The square of 2 is 4
```

for loop

expr1 and **expr3** can contain multiple statements. Each statement is separated by a comma ','

```
for(expr1; expr2; expr3) {
   loop statements;
}
```

Example

```
for (int i=0, j=0; i < 10; i++, j++)
   ……
  expr1: i=0, j=0
  expr2: i<10
  expr3: i++, j++
```

Comma operator (,)

- It has the *lowest* precedence of all operators in C++ and is evaluated from *left to right*
- General form is

```
expr1, expr2
```

- The comma expression *as a whole* has the value and type of its *right operand*
- Sometimes used in **for** statements to allow multiple initializations and multiple processing of indices
- The comma operator is *rarely* used
- **Not** all commas in a C++ program are comma operators

Common Errors

- Mix up assignment **=** with equality operator **==**
- Mix up the comma operator with the semi-colon
- Unaware of extra semi-colons, e.g.,

```
int sum = 0;
for (int j=1; j<=10; j++)
   sum += j;
```

is **not** the same as

```
int sum = 0;
for (int j=1; j<=10; j++);
  sum += j;
```

Common Errors (cont'd)

- Fail to ensure that the termination condition of a loop is reachable → *infinite* loop
- Misuse of relational operators

```
int k = 8;
if (2 < k < 7)✗
  cout >> "true"; // print true
else
  cout >> "false";
```

  - Use `(2 < k && k < 7)` for the correct answer
  - Use `(k > 2 && k < 7)` for the correct answer

Nested for loop

```
#include <iostream>
using namespace std;
int main() {
   for (int i=0; i<3; i++) {
      cout << "Outer for:" << endl;
      for (int j=0; j<2; j++) {
         cout << "Inner for: ";
         cout << "i=" << i << ", j=" << j << endl;
      }
   cout << endl;
   }
   return 0;
}
```

```
Outer for:
Inner for: i=0, j=0
Inner for: i=0, j=1

Outer for:
Inner for: i=1, j=0
Inner for: i=1, j=1

Outer for:
Inner for: i=2, j=0
Inner for: i=2, j=1
```

The outer loop is executed 3 times.
For each iteration of the outer loop, the inner loop is executed 2 times

Exercise

- Write a program to generate a multiplication of `n` rows and `m` column (where `n` and `m` is input by the user).
  - Assume `n > 1` and `m <= 9`.
  - E.g. when `n=4`, `m=3`, the following table is generated

```
1  2  3
2  4  6
3  6  9
4  8  12
```

Use the Loop Counters in Nested Loop Cleverly

```
Enter diagonal length: 5
x
 x
  x
   x
    x
```

```
for (int row = 0; row < length; row++) {
// print (row-1) spaces
  for (int col = 0; col < row; col++)
    cout << ' ';
  cout << 'x' << endl;  // print x on diagonal
}
...
```

break statement

- The **break** statement causes an exit from the ***innermost*** enclosing loop or `switch` statement (discussed already)

```
while (1) {
   cin >> n;
   if (n < 0)
      break;  // exit loop if n is negative
   cout << n << endl;
}
// if break is run, jumps to here
```

✗ *not recommended*

✗ *not recommended in* ***if statement***

`continue` Statement

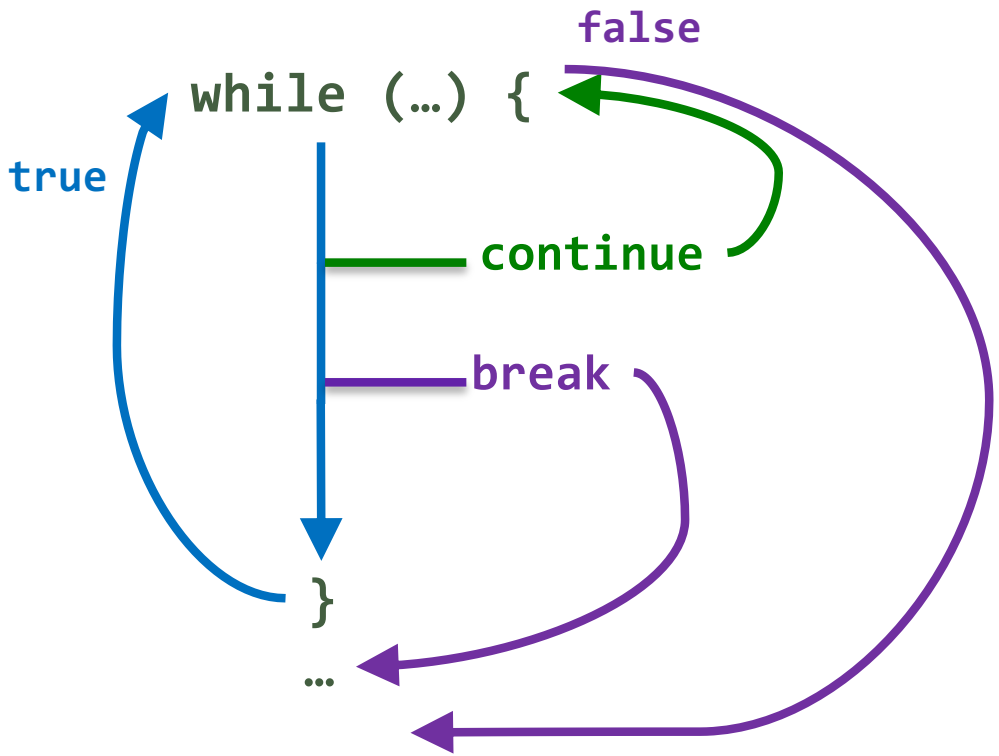
- **`continue`** statement causes the current iteration of a loop to ***stop*** and the next iteration to begin immediately
- It can be applied in a **`while`**, **`do..while`** or **`for`** statement

Example of continue

```
// read in and sum 10 not-too-small numbers.
int cnt = 0;
double x, sum = 0;
while (cnt < 10) {
   cin >> x;
   if (x > -0.01 && x < 0.01)
      continue; // discard small values
   ++cnt;
   sum += x;
   // continue transfers control here
}
```

✗✔ *use with discretion*

continue, break



Example: output the following matrix

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Example: output the following matrix

```
1  //  print a matrix - practise nested for loops
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  const int MaxSize = 10;
7  const int space2 = 2;
8  const int space5 = 5;
9
10 int main() {
11     int cnt = 0;
12     // first row
13     cout << setw(space2) << ' ';
14     for (int i = 1; i <= MaxSize; i++)
15         cout<< setw(space5) << i;
16     cout << endl;
17
18     for (int i = 1; i <= MaxSize; i++) {
19         cout << setw(space2) << i;
20         for (int j = 1; j <= MaxSize; j++)
21             cout << setw(space5) << i * j;
22         cout << endl;
23     }
24
25     return 0;
26 }
```

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

first row

first column

elements

Further remarks

- Use a ***relational*** expression, if possible, rather than an *equality* expression to control a loop or a selection statement, e.g.,

  variable j, increasing from 0 to 3

Don't use

```
while (j != 4) {
   ...
   j++;
}
```

Use

```
while (j < 4) {
   ...
   j++;
}
```

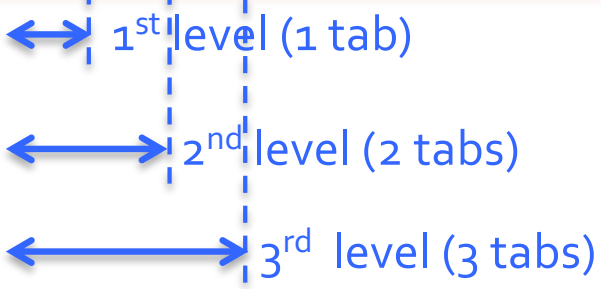
Further Remarks

- Don't use a variable of any **floating** point data type to control a loop because real numbers are represented in their *approximate* values internally
- ***Infinite*** loop can be made

Programming style

- Indent code in a consistent fashion to indicate the flow of control (*use the tab key*)
  - Note the multiple levels of **indentation**

```
void main() {
   int i;
   for(i = 0; i < 100; i++) {
      if(i > 3)
         cout << i;
   }
   return 0;
}
```



Formatting Programs

- ***Indent*** the code properly as you write the program to reflect the **structure** of the program.
  - Improve readability and increase the ease for debugging the program
  - In assignment, marks will be allocated for indentation
- To indent in Visual Studio, you may press the ***tab*** button
- You may select ***multiple*** lines of statements and press ***tab*** to indent all of them
- To move back one level to the left, press ***shift+tab***
- **Make good use of the Text Editor of the IDE!**

Summary

- In C++, repeating tasks could be expressed with

```
while (…) {…}
do {…} while (…);
for (…;…;…) {…}
```

- A complete looping structure may consist of
  - Loop initialization statements
  - Loop Body
  - Exit condition
  - Post Loop statements