CS2311 Computer Programming

LT5: Flow Control

Loops/Iterations

Outline

- **while** and **do while**
- **for**

Loops

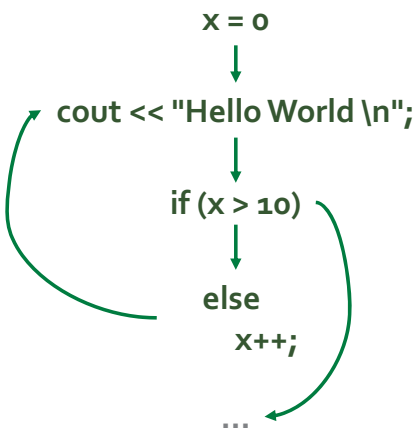
- Beside sequential execution and branch execution, looping is another common control flow in programming.
- When the execution enters a loop, it will execute a block of code repeatedly until the exit condition is matched

Loops

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

Loops

1. Set x=0;
2. cout << "Hello World\n"
3. If (x>10) then exit the loop
4. Add 1 to x
5. Loop back



while

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

    cout << "The maximum value will be computed. \n";
    cout << "How many numbers do you wish to enter?  ";
    cin >> n;
    while (n <= 0) {  /* ensure a positive number is entered */
        cout << "\nERROR: Positive integer required. \n\n";
        cout << "How many numbers do you wish to enter?  ";
        cin >> n;
    }
    cout << "\nEnter " << 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 */
        if (max < x)
            max = x;
    }
    cout << "\nMaximum value: " << max << endl;
    return 0;
}
```

do while statement

- General form of **do while**

  do

  statement

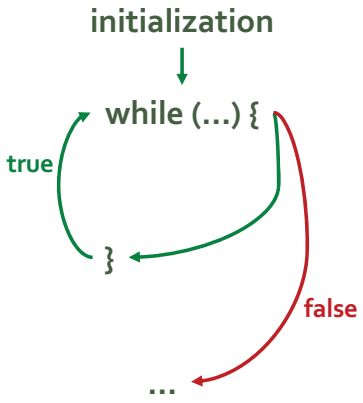
  while (*expression*);

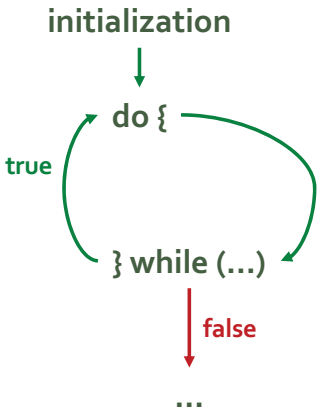
- Semantics:
  - ***statement*** is executed first; thus the loop body is **run at least once**
  - If the value of ***expression*** is non-zero (true), the loop repeats; otherwise, the loop terminates

Example of do .. while

```
int error;
int n;
do {
   cout << "Input a positive integer: ";
   cin >> n;
   if (error = (n <= 0))
       cout << "\nError: negative input! \n";
} while (error);
...
```

while vs do .. while





for-loop Statement

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

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

When **expr2** becomes false, the loop ends.

expr1: Executed before entering the loop. Often used for variable initialization.

expr3: For each iteration, **expr3** is executed after executing the loop body. Often used to update the counter variables.

for loop

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

Example

```
for (i=0, j=0; i<10; i++, j++)
    ......
```

expr1: i=0, j=0

expr2: i<10

expr3: i++, j++

Examples of for

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

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

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

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

Nested for loop

```
#include <iostream>
using namespace std;
int main() {
    int i,j;
    for (i=0; i<3; i++) {
        cout << "Outer for:\n";
        for (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

```
int main() {
    int m, n;   // n is no. of rows, m no. of columns
    int i, j;       // i is row index, j is column index

    for (i=1; i<=n; i++) {    // print each row
        for (j=1; j<=m; j++) { // print each column
            cout << i*j << "\t";
        }
    cout << endl;  // each row ends with an endl
    }
    return 0;
}
```

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 */
```

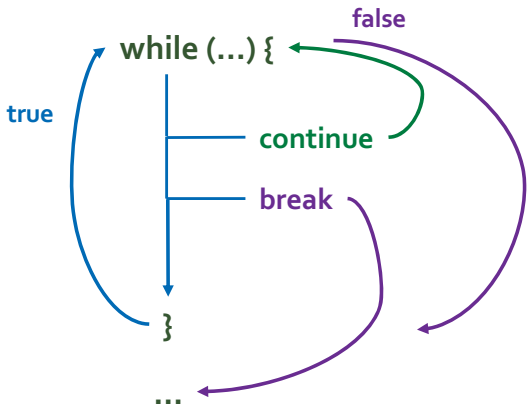

continue Statement

- **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. */
cnt = 0;
while (cnt < 10) {
    cin >> x;
    if (x > -0.01 && x < 0.01)
        continue; /* discard small values */
    ++cnt;
    sum += x;
    /* continue transfers control here */
}
```


continue, break



Example: prime numbers

- Given a positive integer, we want to know if it's a prime or composite number
- A number is prime if and only if it has only two factors, 1 and itself
  - 5 is prime, 6 is composite
  - 1 is not prime nor composite by definition

Version 1.0

```
int n, m, i;
cout << "Enter a positive number: ";
cin >> n;
m = 0;      // m is an indicator variable

for (i=2; i<n; i++) {
    if (n%i == 0)
        m = 1;
}
if (m == 0)
    cout << n << " is prime.\n";
else
    cout << n << " is composite.\n";
```

Version 1.1

```
int n, m, i;
cout << "Enter a positive number: ";
cin >> n;
m = 0;        // m is an indicator variable

for (i=2; i<n; i++) {
    if (n%i == 0) {
        m = 1;
        break;
    }
}
if (m == 0)
    cout << n << " is prime.\n";
else
    cout << n << " is composite.\n";
```


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

Examples of Comma Operator

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

and

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

and

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

are equivalent

Common Errors

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

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

is not the same as

```
sum=0;
for (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

Further Remarks

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

Don't use

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

Use

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

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

- Follow the normal rules of English
  - e.g. putting a space after a comma
- Put one space on each side of a binary operator for readability
  - (e.g. a == b a => b a < b)
- Indent code in a consistent fashion to indicate the flow of control
  - (use the IDE editor/tab key)
  - Note the multiple levels of indentation

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**

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