6 Simple Flow Controls

- Branch
  - Lets program choose between two alternatives
- Flow of control
  - The order in which statements are executed

6.1 'if' statement

'if' statement makes a branch.

```
        int num;
        std::cout << "input number :";
        std::cin >> num;
        if (num % 2 == 0){
                std::cout << "The number is even\n";
        }
```

6.1.1 Boolean Expressions

Boolean expressions are expressions that are either **true** or **false.** Comparison operators such as '>' (greater than) are used to compare variables and/or numbers. Including the parentheses, **(num % 2 == 0)** is the boolean expression. Comparison operators are:

- > greater than
- < less than
- >= greater than or equal to
- <= less than or equal to
- != not equal or inequality
- == equal or equivalent (Not '=' !!!!!)

A few of the comparison operators that use two symbols (No spaces allowed between the symbols!)

6.1.2 'if-else' flow control

When the boolean expression is true, only the true statements enclosed in { } are executed . When the boolean expression is false, only the false statements enclosed in { } are executed.

```
        int num;
        std::cout << "input number :";
        std::cin >> num;
        if (num % 2 == 0){
                std::cout << "The number is even\n";
        }else{
                std::cout << "The number is odd\n";
        }
```

6.1.3 AND/OR/NOT

Boolean expressions can be combined into more complex expressions.

&& – The AND operator.

```
        if ((2 < x) && (x < 7)){
                // true;
        }
```

- This goes to the true statement only if x is between 2 and 7.
- Inside parentheses are optional but enhance meaning.
- If the first expression is false, the second expression won't be evaluated.

| | – The OR operator (no space!)

```
if (( x == 1) || ( x == y)){
        // true;
}
```

- This goes to the true statement if x is 1 or x is equal to y, also if both comparisons are true.

! – negates any boolean expression.

```
if (!( x < 1)) {
        // true;
}
```

This goes to the true statement if x is greater than or equal to 1. This can be rewritten as

```
if ( x >= 1) {
        // true;
}
```

! Operator can make expressions difficult to understand. . . use only when appropriate.

6.2 'while' loop

```
while(expression){
        //Loop body : do here while expression is true
}
```

- First, the expression is evaluated.
  - If false, the program skips to the line following the while loop.
  - If true, the body of the loop is executed.
- During execution, some item from the expressions changed.
- After executing the loop body, the expression is checked again repeating the process until the expression becomes false.
- A while loop might not execute at all if the expression is false on the first check.

6.2.1 sample05

```
/*
 CPP sample05.cpp
 */

#include <iostream>

using namespace std;

int main(int argc, char *argv[]) {
        int a;

        while (1) {
                cout << "input a number:";
                cin >> a;

                if (a == 0)
                        break;

                if (a % 2) {
                        cout << a << " is odd number" << endl;
                } else {
                        cout << a << " is even number" << endl;
                }
        }
        cout << "bye" << endl;

        return 0;
}
```

6.3 'do-while' loop

```
        do{
                //Loop body
        } while(expression);
```

- A do-while loop is always executed at least once.
- The body of the loop is first executed.
- The boolean expression is checked after the body has been executed.

6.4 'for' loop

```
        for (initialize ; expression ; increment){
                // do here while expression is true
        }
```

- The initialize step is executed first, and only once.
- Next, the expression (condition) is evaluated.
  - If it is true, the body of the loop is executed.
  - If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the increment statement.

- This statement allows you to update any loop control variables.
- The condition is now evaluated again.
- If it is true, the loop executes and the process repeats.

Example:

```
//
// Example
//
#include <iostream>
using namespace std;
int main (int argc, char *argv[]) {
        for (int i = 1 ; i <= 3 ; i++){
                for (int j = 1 ; j <= 3 ; j++){
                        cout << i << " x " << j << " = " << i * j << endl;
                }
        }
        return 0;
}
```

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
```

6.4.1 sample06

```cpp
/*
 CPP sample06.cpp
 */

#include <iostream>
#include <iomanip>

using namespace std;

int main(int argc, char *argv[]) {
        int n;

        cout << "number of terms : ";
        cin >> n;
        cout << "n=" << n << endl;
        ;

        double x = 0.0;
        for (int i = 0; i < n; i += 2) {
                x += 1.0 / (2 * i + 1);
                x -= 1.0 / (2 * i + 3);
        }

        cout << "Pi=" << setprecision(16) << 4.0 * x << endl;

        return 0;
}
```

6.5 break, continue

```cpp
        int a;
        int i = 0;
        while(i < 5){
                std::cout << "input a number:";
                std::cin >> a;
                if (a == 0) break;
                if (a < 0) continue;
                std::cout << "i=" << i << std::endl;
                i++;
        }
        std::cout << "bye" << std::endl;
```

When a=0, get out from the loop. when a<0, go to the end of loop without executing **cout** and **i++**.