1. The increment (++) and decrement (--) operators increase or decrease a variable's value by one, respectively. They are great if all you want to do is increment (or decrement) a variable: "`i++;`".

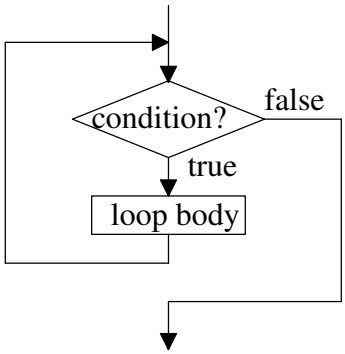
HOWEVER, don't use them for anything else. Predict the output resulting from this program.

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

int main() {
      int i;
      i = 5;

      cout << "before: i = " << i << endl;
      cout << "i++ = " << i++ << "  i++ = " << i++ << endl;
      cout << "after: i = " << i << endl;

} // end main
```

```
before: i = 5
i++ =
```

2. A `while` loop allows code to be executed repeatedly as long as some condition is satisfied. The `while` loop is an example of a *pre-test loop*. The syntax of a `while` loop is:

```
while (condition) {
    // loop "body"
    statement1;
    statement2;
    statement3;
} // end while
```



If the `condition` evaluates to `true`, then the statements of the body are executed. If the `condition` is `false`, then the body is skipped. If the body is executed, loop back and re-evaluate the `condition`.
NOTE: The statements in the body of the `while` should be indented.

Typically, the `condition` involves comparing "stuff" using relational operators (`<, >, ==, <=, >=, !=`) and Boolean operators (`!(not), || (or), && (and)`). For example, we might want to perform *input validation* on a user's menu selection until they "get it right."

a) Complete the `condition` of the while loop:

```
cout << "Menu choices\n";
cout << "1. First choice\n";
cout << "2. Second choice\n";
cout << "3. Third choice\n\n";
cout << "Enter your choice: ";
cin >> choice;
while (                                                    ) {
      cout << endl << "Your menu choice of " << choice <<" is invalid."<< endl;
      cout << "Please enter 1, 2, or 3" << endl << endl;
      cout << "Menu choices\n";
      cout << "1. First choice\n";
      cout << "2. Second choice\n";
      cout << "3. Third choice\n\n";
      cout << "Enter your choice: ";
      cin >> choice;
} // end while
```

b) What happens if the user types in the string "one" at the prompt?

3. A `while` loop can be combined with a *counter* variable to loop a specified number of times. The counter variable is also known as a *loop-control* variable. The following code generates a single-digit multiplication table for 3 x 1, 3 x 2, ..., 3 x 9.

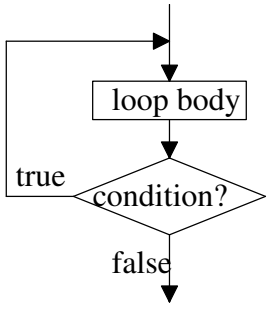
```
#include <iostream>
using namespace std;

int main() {
    int counter;

    counter = 1;
    while (counter <= 9) {
        cout << "3 x " << counter << " = " << counter * 3 << endl;
        counter++;
    } // end while

} // end main
```

```
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
```

Write a `while` loop that allows the user to enter a positive starting value and count down to zero. Write each value in the count down. When zero is reached, write the string "BLAST OFF!!!".

4. A `do-while` loop is a *post-test loop* that executes the body once before checking the `condition` to see if the loop body should be repeated. While the `condition` is `true` the loop is repeated. The syntax of a `do-while` loop is:

```
do {
    // loop "body"
    statement1;
    statement2;
    statement3;
} while (condition);
```



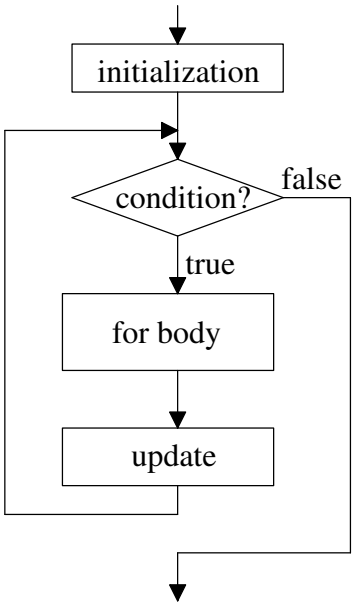
NOTE: The '`;`' after the `condition` is required.

a) Why is the `do-while` loop useful for validating user input?

b) Why is the `do-while` loop useful for menu-driven user input?

5. A `for` loop is useful for counter-controlled loops. The `for` loop is an example of a *pre-test loop*. The syntax of a `for` loop is:

```
for (initialization; condition; update) {
    // loop "body"
    statement1;
    statement2;
    statement3;
} // end for
```



The `initialization` code is run once at the start of the for loop. If the `condition` evaluates to `true`, then the statements of the body are executed. If the `condition` is `false`, then the loop is existed. If the body is executed, the `update` code runs before looping back and re-evaluate the `condition`. NOTE: The `initialization` and `update` can be several statements separated by commas (instead of the normal semi-colons).

The `for` loop code below generates a single-digit multiplication table for 3 x 1, 3 x 2, ..., 3 x 9.

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

int main() {
    int counter;

    for (counter = 1; counter <= 9; counter++) {
        cout << "3 x " << counter << " = " << counter * 3 << endl;
    } // end for

} // end main
```

```
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
```

Write a `for` loop that allows the user to enter a positive starting value and count down to zero. Write each value in the count down. When zero is reached, write the string "BLAST OFF!!!".

Using a loop, we can process data repeatedly, but how do we know when to stop? Two main ways can be used:

- counter controlled - the user tells us **before the loop** how many data items will follow. A `for` loop can then be used to loop that many times.
- sentinel controlled - the user might not know how many data items will follow, so we loop until a special *sentinel* value that cannot be confused with a valid datum is entered, *e.g.*, -999 for a test score. A `while` loop or `do-while` loop are used for sentinel controlled processing.

Suppose I wanted to average the scores on an assignment for a class.

a) Complete the `for` loop of the counter-controlled program.

```
How many scores do you have? 5
Enter a score: 10
Enter a score: 12
Enter a score: 14
Enter a score: 16
Enter a score: 18
The average is 14.0
```

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

int main() {
   int counter;
   int numberOfScores;
   double totalOfScores;
   double score;
   double average;

   totalOfScores = 0.0;
   cout << "How many scores do you have? ";
   cin >> numberOfScores;

   for (                                                    ) {

      cout << "Enter a score: ";
      cin >> score;
      totalOfScores = totalOfScores + score;
   } // end for

   average = totalOfScores / numberOfScores;
   cout << "The average is " << setprecision(1) << fixed << average << endl;

} // end main
```

b) Complete the `while` loop of the sentinel-controlled program.

```
Enter a score (or -999) to exit: 10
Enter a score (or -999) to exit: 12
Enter a score (or -999) to exit: 14
Enter a score (or -999) to exit: 16
Enter a score (or -999) to exit: 18
Enter a score (or -999) to exit: -999
The average is 14.0
```

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

int main() {
   int numberOfScores;
   double totalOfScores;
   double score;
   double average;

   totalOfScores = 0.0;
   numberOfScores = 0;
   cout << "Enter a score (or -999) to exit: ";
   cin >> score;

   while (                              ) {

      totalOfScores = totalOfScores + score;
      numberOfScores++;
      cout << "Enter a score (or -999) to exit: ";
      cin >> score;
   } // end while

   average = totalOfScores / numberOfScores;
   cout << "The average is " << setprecision(1) << fixed << average << endl;

} // end main
```

Operator Precedence and Associativity		
Operator	**Associativity**	**Usage(s)**
`::`	unary: left-to-right binary: right-to-left	
`() [] -> .`	left-to-right	parenthesis index object pointer/structure pointer dot operator
`++ -- - + ! ~ (type) * & sizeof`	right-to-left	increment and decrement unary negation and plus logical negation one's complement operator type cast indirection address-of/reference
`* / %`	left-to-right	multiply, division, remainder
`+ -`	left-to-right	addition and subtraction
`<< >>`	left-to-right	io: insertion and extraction bit-wise shift left and right
`< <= > >=`	left-to-right	comparisons for inequality
`== !=`	left-to-right	comparison for equality
`&`	left-to-right	bit-wise AND
`^`	left-to-right	bit-wise exclusive-OR
`|`	left-to-right	bit-wise OR
`&&`	left-to-right	logical AND
`||`	left-to-right	logical OR
`?:`	right-to-left	conditional
`= += -= *= /= %= &= ^= |= <<= >>=`	right-to-left	assignment
`,`	left-to-right	comma operator