C++ PROGRAMMING

DR. YIHSIANG LIOW (JULY 22, 2025)

Contents

12. while Loops

OBJECTIVES

- Write while-loops
- Use sentinel values to exit a while-loop

Like the `for`-loop, the purpose of the `while`-loop is to repeat a block of statements (or just a statement). While there is a difference, once you know how the `for`-loop works, the while-loop is easy: the `while`-loop is a simplification of the `for`-loop.

`while`-loop

First run this. I' m not going to explain it since this is old stuff.

```
int sum = 0;
for (int i = 1; i <= 10; i++)
{
    sum += i;
    std::cout << "i:" << i << " sum:" << sum
              << std::endl;
}
std::cout << "sum:" << sum << std::endl;
```

Now try this:

```
int sum = 0;
int i = 1;

for (; i <= 10;)
{
    sum += i;
    std::cout << "i:" << i << " sum:" << sum
              << std::endl;
    i++;
}
std::cout << "sum:" << sum << std::endl;
```

The program works the same right? One final modification:

```
int sum = 0;
int i = 1;
while (i <= 10)
{
     sum += i;
     std::cout << "i:" << i << " sum:" << sum
               << std::endl;
i++;
}
std::cout << "sum:" << sum << std::endl;
```

That's all there is to it.

The `while`-loop is just the `for`-loop without the initialization and the update part; the `while`-loop only has the boolean condition from the `for`-loop. So if a `while`-loop must perform an initialization and/or an update, you can put them in the following spots:

```
int sum = 0;
int i = 1;                  // INITIALIZATION
while (i <= 10)             // BOOLEAN CONDITION
{
    sum += i;
    i++;                    // UPDATE
}
```

By the way, there is a slight difference between the `while`-loop and `for`-loop. The difference has nothing to do with the flow of execution. It has to do with where we declare variables in loops. (This was already mentioned earlier in the notes on `for`-loops).

Look at these:

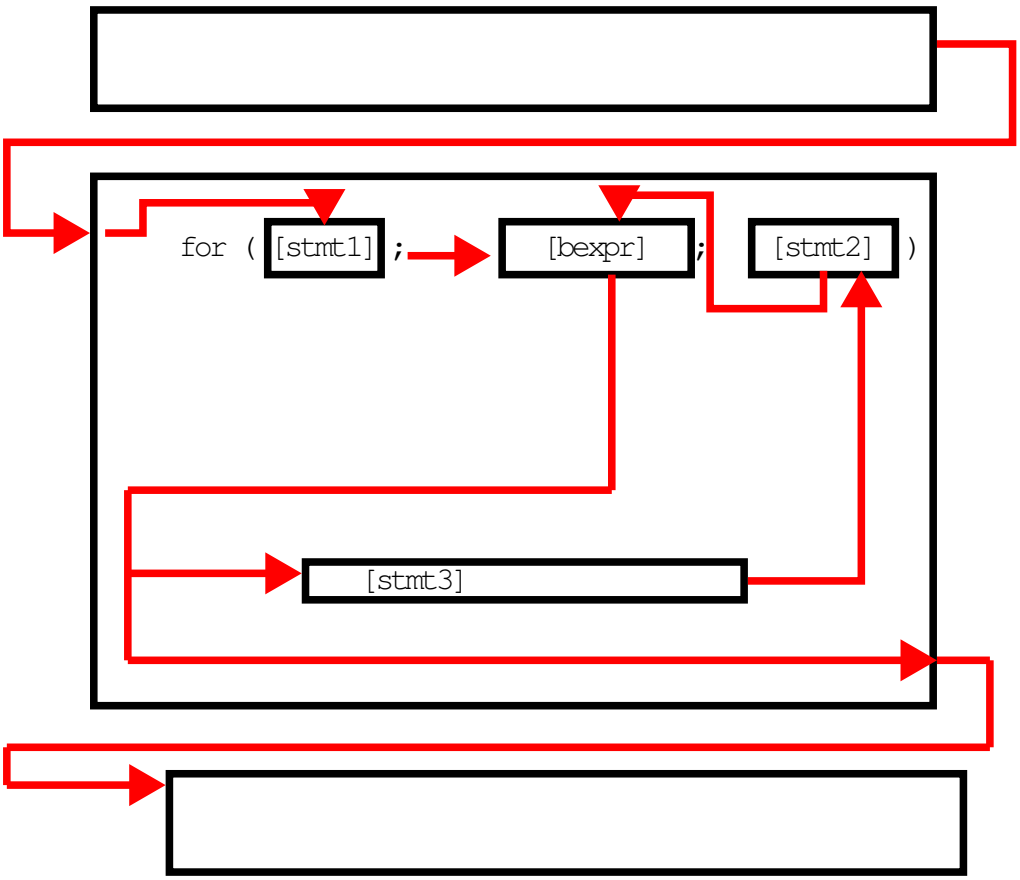
```
int sum = 0;
for (int i = 1; i <= 10; i++)
{
    sum += i;
    std::cout << "i:" << i << " sum:" << sum
              << std::endl;
}
std::cout << "i:" << i << " sum:" << sum
          << std::endl;
```

```
int sum = 0;
int i = 1;
while (i <= 10)
{
      sum += i;
      std::cout << "i:" << i << " sum:" << sum <<
      std::endl;
      i++;
}
std::cout << "i:" << i << " sum:" << sum
          << std::endl;
```

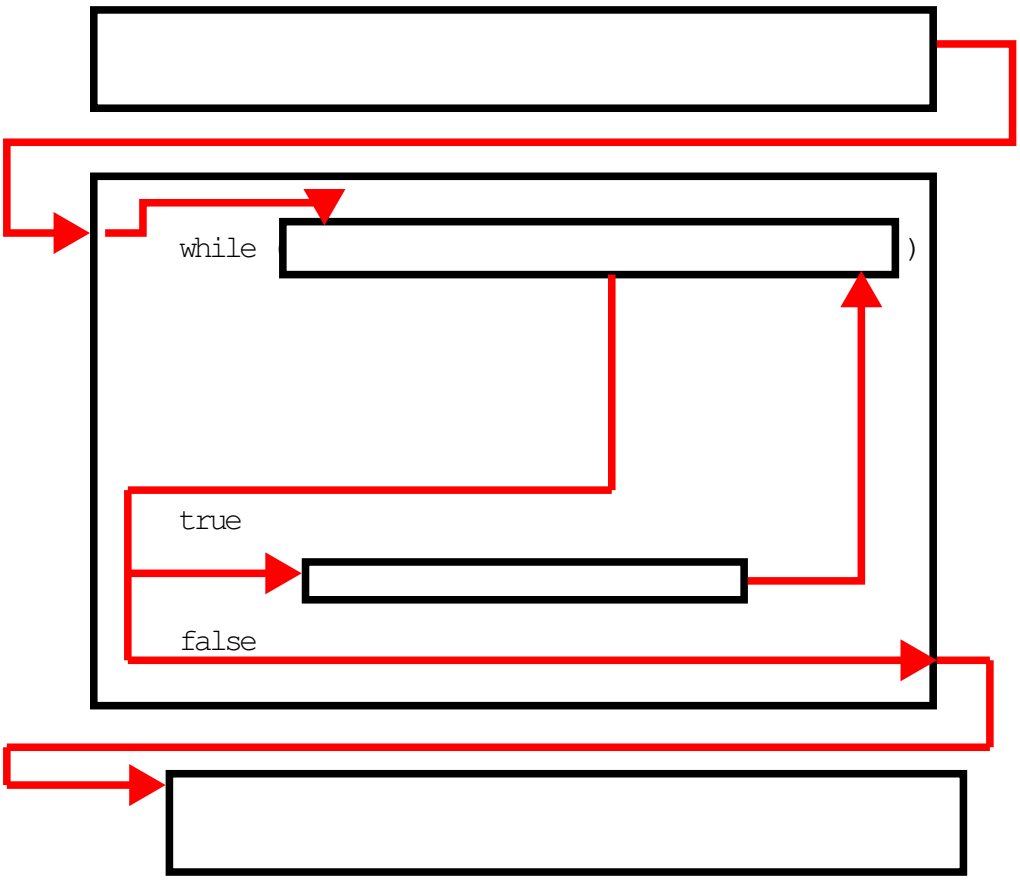
Because the variable `i` in the second example is declared outside the `while`-loop, it is still available after exiting the `while`-loop. In the case of the `for`-loop, the variable `i` is destroyed when you exit the `for`-loop.

Flow of execution in a `while` loop

Recall that the flow of execution for the for-loop looks like this:



Here's the flow of execution of the while-loop:


while (
)
true
false

More Examples

Here's a typical example that uses the `while`-loop.

```
int i = 0;
std::cout << "gimme a number ...";
std::cin >> i;
while (i != 42)
{
      std::cout << "so 42 is not your fav number?";
                << std::endl;
      std::cout << "gimme a number ...";
      std::cin >> i;
}
```

Notice something about this `while`-loop?

Hmmmm ....there is **no control variable** unlike our `for`-loop examples. Now try this:

```
int sum = 0;
int i = 0;
std::cout << "gimme a number ...";
std::cin >> i;
while (i != 42)
{
      sum += i;
      std::cout << "sum: " << sum
                << std::endl;
      std::cout << "gimme a number ...";
      std::cin >> i;
}
std::cout << "final sum: " << sum << std::endl;
```

In general, if you want to process a bunch of numbers, you can run over these numbers in two different ways: Either the program knows how to compute them (example: the numbers are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10) or the program does not (example: it's based on user input).

For the first case, you should use the **for-loop** with a **control or index or counter variable** running over the values.

For the second case, you should use the **while-loop**. To let the user stop the data processing, you have to pick a special value for the user to use to indicate end-of-data. This special value is called the **sentinel value**.

Now I want you to step back and look at the “structure”of the above program:

initialization
prompt user for data
while value of data is not sentinel value:
    process data
    prompt user for data

Note that there is code duplication:

initialization
prompt user for data
while value of data is not sentinel value:
    process data
    prompt user for data

Exercise -1.0.1. Write a program that continually “rolls a die”and prints the face value of the die until 6 is reached. (Use rand() to generate a random integer between 1 and 6.)

Exercise -1.0.2. Write a program that computes the product of integers entered by the user. Use a sentinel value of 0 to indicate the end of data. Here’s the skeleton code:

```
int product = _____;
int i = 0;

std::cout << "gimme a number ...";
std::cin >> i;
while (__________)
{
     ________________________;
     std::cout << "product: " << product
               << std::endl;
     std::cout << "gimme a number ...";
     std::cin >> i;
}
std::cout << "final product: " << product << std::endl;
```

Exercise -1.0.3. Write a program that computes the cost of head grafting surgery. Each head costs $100.42. Display the total number of heads requested. The program exits the loop when the number of heads requested is zero. Here is the skeleton code:

```
int heads = 0;
double cost = 0.0;
int totalHeads = _____;
double totalCost = _____;

std::cout << "how many heads? ";
std::cin >> heads;
while (__________)
{
      cost = ____________________________________;
      totalHeads = ____________________________________;
      totalCost = ____________________________________;
      std::cout << "total heads requested: "
                << totalHeads << std::endl
                << "total cost: "
                << totalHeads <<
                '\n' << std::endl;
      // prompt for heads
}

std::cout << "your final bill:" << totalCost << '\n'
          << "i hope you enjoy your extra "
          << totalHeads << " heads" << std::endl;
```

Exercise -1.0.4. Due to popular demand and lack of staff and surgeons (and donors), we have to limit the number of extra heads per visit. Modify the above program so that the program exits the while-loop when the number of heads requested is zero or when the total number of heads requested exceeds 42.

Exercise -1.0.5. Write a simple number guessing game. The program generates a random integer from 1 to 10 (inclusive), and continually prompts the user to guess the number until the correct answer is given. When you're done with that, get the program to print the number of tries.

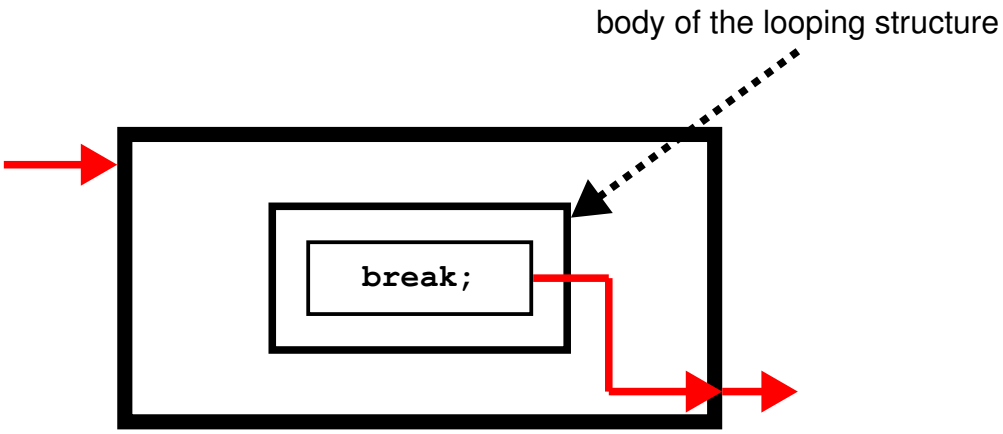
Exercise -1.0.6. Modify our multiplication game so that it keeps posing questions until the user answers 5 questions correctly. While prompting the user, print the number of correct answers. (Once

you're done with that, make a slight modification to prompt the user for the number of correct answers before terminating the program.)

Exercise -1.0.7. Modify the above program so that the program terminates when 5 **`consecutive`** correct answers are given.

break

Fortunately the `break` statement works the same way for the `while`-loop as it does for the for-loop.




```
int i = 0;
while (i < 1000)
{
      std::cout << i << std::endl;
      if (i == 100) break;
      i++;
}
std::cout << "out ... " << i << std::endl;
```

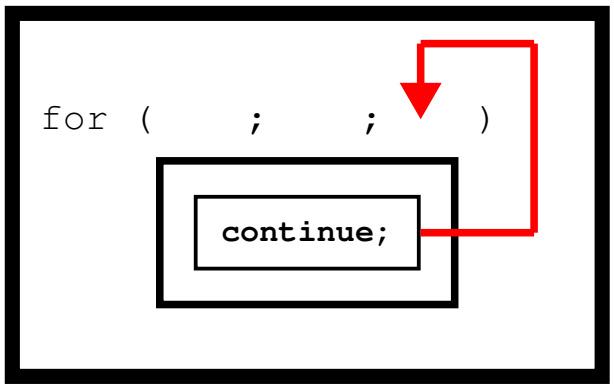
Again, the same advice for using the `break` statement in the `for`-loop applies: Don't overuse it.

Exercise -1.0.8. What is the output? Work it out by hand and then check it by running the program.

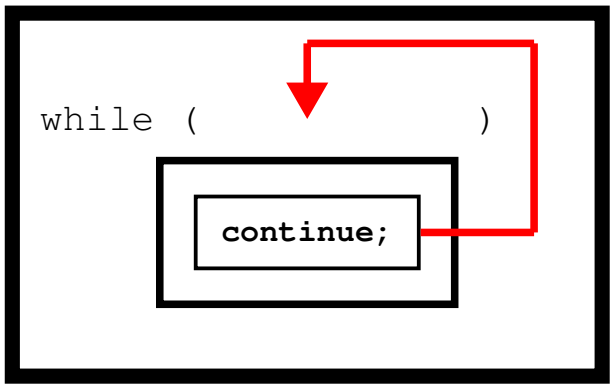
```
int i = 0;
while (i <= 10)
{
      std::cout << "A " << i << ' ';
      i += 2;
      std::cout << "B " << i << ' ';
      if (i > 5) break;
      std::cout << "C " << i << ' ';
      i += 3;
      std::cout << "D " << i << '\n';
}
std::cout << i << std::endl;
```

continue

Recall that this is the flow of execution for `continue` in the `for`-loop:



The `continue` statement works like this for the `while`-loop:



Exercise -1.0.9. What is the output? Work it out by hand and then check it by running the program.

```
int i = 0;
while (i <= 10)
{
      i += 5;
      if (i > 5) continue;
      i += 3;
}
std::cout << i << std::endl;
```

Exercise -1.0.10. Here's an example. Run it.

```
int numHeads = 0;
std::cout << "how many heads do you have? ";
std::cin >> numHeads;
while (numHeads != 0)
{
      std::cout << "cool!" << std::endl;
      std::cout << "how many heads do you have?";
      std::cin >> numHeads;
}
```

Now insert code that prints “rubbish”and `continue` to the next iteration of the loop if `numHeads` is negative:

```
int numHeads = 0;
std::cout << "how many heads do you have? ";
std::cin >> numHeads;
while (numHeads != 0)
{
      // INSERT CODE HERE
      std::cout << "cool!" << std::endl;
      std::cout << "how many heads do you have?";
      std::cin >> numHeads;
}
```

More examples

Remember our example on computing the maximum of a list of values?

```
int x, y, z;
std::cin >> x >> y >> z;

int max = x;
if (max < y) max = y;
if (max < z) max = z;
std::cout << max << std::endl;
```

Let's write a program that prompts the user for integers and computes the maximum of all integers entered. The program stops prompting when the user enters -99999. Here's the skeleton code:

```
int x = 0;
std::cin >> x;
int max = x;

while (x != -99999)
{
      // process x
      std::cout << max << std::endl;
      std::cin >> x;
}
std::cout << "final max: " << max
          << std::endl;
```

Note that we need a "special value"of -99999 to stop the loop. What do we do with x? We check if it's larger than what we have in max:

```
int x = 0;
std::cin >> x;
int max = x;

while (x != -99999)
{
      if (max < x) max = x;
      std::cout << max << std::endl;
      std::cin >> x;
}
std::cout << "final max: " << max << std::endl;
```

Exercise -1.0.11. Modify the above program so that it computes the minimum instead of the maximum.

Exercise -1.0.12. Modify the above program so that it computes the sum of all values entered by the user until 0 is entered. When 0 is entered, the program prints the final sum and stops.

Exercise -1.0.13. Write a program that continually prompts the user for numbers and prints the running average. The program stops prompting when the user enters `-99999`.

Summary

The format of the while-loop looks like this:

```
while ([bool expr])
    [stmt]
```

where `[bool expr]` is a boolean expression and `[stmt]` is a single statement or a block of statements.

The `while`-loop is just the `for`-loop without the initialization and the update parts.

In general, if your loop requires a variable that runs over a sequence of numbers that can be computed by the program, you should use the `for`-loop. Otherwise you should use the `while`-loop. (There are exceptions. But this advice will do for now).

The `break` statement, if executed in the body of the `while`-loop, will exit the `while`-loop and continues with the execution of the statement immediately after the `while`-loop.

The `continue` statement, if executed in the body of the `while`-loop, will immediately bring the execution to the boolean condition of the `while`-loop.

Exercises

Q1. Write a program that computes the gas mileage of a car. The program continually prompts the user for the relevant data. In each iteration of the loop, the program prompts the user for the miles traveled and the gas used. When the user entered 0 for miles, the program stops prompting the user, and displays the gas mileage (i.e. total miles divided by the total gas usage).

Q2. Modify our multiplication game so that it keeps posing questions until the user answers 80% of the questions correctly. While prompting the user, print the percentage of his/her correct answers.

Q3. Investigate the following problem: How many die rolls do you need to get the value 6? What about two 6's? What about three 6's? ... What is the likelihood of getting a 6? What is the likelihood of getting two 6's? ... What about the case of two consecutive 6's? Three consecutive 6's?