The While Loop

1E3
Topic 6

Objectives

- To recognise when a WHILE loop is needed.
- To be able to predict what a given WHILE loop will do.
- To be able to write a correct WHILE loop.
- To be able to use a WHILE for handling input terminated with a sentinel.
Textbook

- This topic is covered in the first part of Chapter 5.

- You should now read up to the end of the Fibonacci number programming example.

WHILE Loops

- WHILE some condition holds DO some action repeatedly
  - WHILE there is more dessert eat a spoonful;
  - WHILE investment has not doubled leave it for another year;
  - WHILE notes not understood study them some more;
**WHILE loops in C++**

- `while ( condition )
  action;
- `while ( condition )
  {
    multiple statements
  }
- Note there is no ; after the condition’s ) or after the closing ]

**Example**

```cpp
while (countdown > 0)
{
  cout << "Hello\n";
  countdown = countdown - 1;
}
```

Note: We will see later that this behaviour would be more suited to a FOR loop.
Flowchart for while

while (<Condition>)
<Statement>

If the <Condition> evaluates to true, the flow of program execution exits via the T (for True) branch, flowing to the <Statement>. When the <Statement> completes, execution flow loops around to perform the whole while construct again. If the condition evaluates to false, execution flows to the Out terminal, on to the statement following the while construct.

Do zero or more times

- Note that a while loop causes the body to be executed zero or more times.
- From the flowchart you can see that if the condition is initially true control goes straight through, without executing statement.
- Later we'll look at a related construct that carries out the loop body one or more times.
Infinite Loops

- Make sure your loop body changes the loop condition.
- This loop will never end:

```c++
countdown = 5;
while (countdown > 0) {
    cout << "Hello\n";
}
```

Correct WHILE loops

- Initialise the condition variables
  - countdown = 5
- Test the condition
  - countdown > 0
- Modify the condition variables
  - countdown = countdown - 1
  - This should generally move the condition closer to being false.
What will this loop print?

A. Nothing
B. 2 4 6 8 10
C. 2 4 6 8 10 12
D. 2 4 6 8 10 12 14 16 18 ...

x = 2;
while (x != 12) {
    cout << x << " ";
    x = x+2;
}

What will this loop print?

Answer

- 2 4 6 8 10
- It doesn’t print the 12 because once x becomes 12, “x not equal to 12” (x !=12) becomes false and the loop ends.
- It does terminate – x will reach 12
What will this loop print?

```cpp
x = 2;
while (x != 13) {
    cout << x << " ";
    x = x+2;
}
```

A. Nothing  
B. 2 4 6 8 10  
C. 2 4 6 8 10 12  
D. 2 4 6 8 10 12 14 16 18 ...

What will this loop print?  
Answer

`x = 2;  
while (x != 13) {
    cout << x << " ";
    x = x+2;
}`

- 2 4 6 8 10 12 14 16 18 ...
- `x` never becomes equal to 12

Be careful with loop conditions that expect equality – especially doubles are unlikely to ever be equal:

- `while (p != 66.666) ...` it’s very unlikely that any computation will give a `p` exactly equal to 66.666
Double an investment

- How many years will it take to double 1000 euro if the annual interest is 5%?
- The basic loop is
  - while value < (2 * initial-value)
    - add another year's interest to value
- But this doesn't give us a value for the number of years it takes.
- See doubleinvestment.cpp
- Exercise: Turn this into a program that reads in the initial investment and interest rate.

Quick self-test

- Show the output of this code if x is of type int
  
  ```
  x = 10;
  while (x > 0)
  {
    cout << x << " \n";
    x = x - 3;
  }
  ```

- Show the output if the condition is changed from (x > 0) to (x < 0).
Counter controlled while loops

- A while loop can be used to
  - read 20 numbers, and average them
  - printout the value of an investment for each of 35 years
- These are counter controlled loops.
- But we will prefer FOR loops for loops that are to be executed a specific number of times.

Counter-controlled

counter = 0; //initialize the loop control variable
while (counter < N) //test the loop control variable
{
  .
  .
  counter++; //update the loop control variable
  .
  .
}

**Sentinel controlled loops**

- We’ll use these a lot so pay attention!
- While loops are perfect for handling sequences of input of unknown length.
- If a special “sentinel” value terminates the sequence, then we use a sentinel controlled loop.
  - E.g. use 0 to terminate series of student numbers
  - Use a negative number to terminate a sequence of exam results

**Sentinel controlled loop**

```cpp
cin >> variable;    //initialize the loop control variable
while (variable != sentinel)  //test the loop control variable
{
    
    cin >> variable;    //update the loop control variable
    
}
```

Deal with the variable
Input is 10 5 8 7 -1
Which outputs 30?

- **A**
  ```
  int num, sum;
  cin >> num; sum = num;
  while (num > 0) {
    cin >> num;
    sum = sum + num;
  }
  cout << sum;
  ```

- **B**
  ```
  int num, sum = 0;
  cin >> num;
  while (num > 0) {
    sum = sum + num;
    cin >> num;
  }
  cout << sum;
  ```

- **C**
  ```
  int num, sum = 0;
  while (num > 0) {
    cin >> num;
    sum = sum + num;
  }
  cout << sum;
  ```

**B** (below) is the correct answer. Note it carefully.

- Both A and C will give 29. They will add all the numbers, including the -1: it gets added before it gets checked for (num > 0).
- C has the additional problem of not initialising num for the first time it tests (num > 0).
Flag controlled loops

found = false; //initialize the loop control variable
while (!found) //test the loop control variable
{
  
  
  if (expression)
    found = true; //update the loop control variable
  
  
}

Number-guessing game

- Example 5.6 in textbook
- Program generates a random number between 1 and 100, and then asks the user to guess the number. Responds
  - “Guess a lower number” or
  - “Guess a higher number”
  - till user enters the correct number.
Random numbers

- A random integer between 1 and 100 is generated using
  - \((\text{rand()} + \text{time(0)}) \% 100\)
- \text{rand()} may produce the same number each time!
  - So add an integer based on the current time:
    - \text{time(0)}
- Convert to 1..100 range by getting the remainder of dividing the random integer by 100 i.e. \% 100
- \text{rand()} is in \text{cstdlib} library; \text{time()} is in \text{ctime}

Sentinel controlled Exercise

- Add a sequence of numbers terminated by a 0.
  - See add-sequence.cpp
  - Alter it to get average, min and max.
- Test whether a sequence of numbers is all even.
  - This involves a boolean flag to keep track of whether a non-even has been found.
  - See testsequence.cpp
Which loop does not terminate?
Assume \( x \) is an int

A. \[\text{while (} x \neq 999\text{) } \]
\[\text{cin} \gg x;\]
\[\text{sum} = \text{sum} + x; \]

B. \[x = 0;\]
\[\text{while (} x \neq 12\text{) } \]
\[x = x + 3; \]

C. \[\text{cin} \gg x;\]
\[\text{while (} x \neq 999\text{) } \]
\[\text{sum} = \text{sum} + x; \]

D. \[\text{bool done} = \text{false};\]
\[\text{while (!done)} \]
\[\text{cin} \gg x;\]
\[\text{if (} x < 0\text{)}\]
\[\text{done} = \text{true};\]

Which loop does not terminate?
Answer is C

A. \[\text{while (} x \neq 999\text{) } \]
\[\text{cin} \gg x;\]
\[\text{sum} = \text{sum} + x; \]

B. \[x = 0;\]
\[\text{while (} x \neq 12\text{) } \]
\[x = x + 3; \]

C. \[\text{cin} \gg x;\]
\[\text{while (} x \neq 999\text{) } \]
\[\text{sum} = \text{sum} + x; \]

D. \[\text{bool done} = \text{false};\]
\[\text{while (!done)} \]
\[\text{cin} \gg x;\]
\[\text{if (} x < 0\text{)}\]
\[\text{done} = \text{true};\]

- C is the answer – nothing in the loop body changes \( x \); it cannot terminate unless initial input is 999.
- A is not ideal – it will add 999 to the sum, but it can terminate.
- B is fine.
- D is fine; it will terminate so long as user enters a negative.
Which loop definitely terminates?
Assume x is an int

A. cin >> x;
   while (x != 999) {
      sum = sum + x;
   }

B. cin >> x;
   while (x != 999) {
      sum = sum + x;
      cin >> x
   }

C. x = 0;
   while (x != 12) {
      x = x + 3;
   }

D. x = 1;
   while (x != 12) {
      x = x + 2;
   }

Which loop definitely terminates?
Answer is C
Which makes sense?

char ans = ‘n’;  A
while (ans != ‘n’) { ...
cin >> ans;}

bool f = false;  B
while (!f) { ...
f = false;}

string ans = “yes”;  C
while (ans != “no”) { ...
cin >> ans;}

bool f = false;  D
while (f) { ...
f = true;}

Which makes sense? Answer is C

- C makes sense;
- A won’t enter loop;
- B doesn’t look like it can terminate;
  - f will always be false, “!f” always true.
- D won’t enter loop
  - “while (f)” means while f is true.
Another WHILE exercise

- Convert your feet to centimetres program into one which asks the user if he’d like to do another conversion.

```cpp
char answer = 'y'; ← initialise
while (answer == 'y') ... ← test
    { get and convert another length ... ...
        . . .
        cout << "Again? (y/n):";
        cin >> answer; ← update
    }
```

Next ...

- The next construct we’ll see is the FOR loop.
- FOR is another way of doing something repeatedly
  - But usually for a predictable number of times
  - Counter controlled loops.