

1 Week 3/4

1.1 Control Flow

These next few weeks we will go over control flow. Prior to this week you we read the code in **main** from top to bottom, where each line executes sequentially. Control flow alters the flow of the program by having blocks of code execute conditionally (if we find some condition **true**) or executed multiple times (after checking some condition is still **true**). These allow computers to make simple decisions for us and automate tedious processes.

1.1.1 Control flow basics

Here are the basic control flow and how they should be used:

Control Flow	Best Use Cases
if	Making decisions based on conditions.
for	Repeating a block of code a specific number of times.
while	Repeating a block of code until a condition is met, often an unknown number of times.
do-while	Repeating a block of code at least once and then continuing based on a condition.

The correct syntax for **if-else** looks like

```
if (condition1) {  
    //code1  
} else if (condition2) {  
    //code2  
} else {  
    //code3  
}
```

The computer processes this by checking **condition1**. If that's **true**, we execute **code1** and jump to the end of the **if-else** statement. If that's **false** we check if **condition2** is **true**. If that's **true**, we execute **code2** and jump to the end. If that's **false** we go check the next condition, continuing in that order. If we reach the **else** it means **condition1** and **condition2** were both **false** and that **code3** is executed. Note that the **else if** and **else** conditions are optional. For example, it's possible you could want to change a variable if a condition is met and leave it alone otherwise. Then a lone **if** statement would be suitable.

The condition in the statement can be any expression that can be evaluated to give a boolean. Here is a review of the Boolean expressions:

Logical operator	Outcome
&&	x && y is true if x and y are both true.
 	x y is true if at least one of x and y is true.
!	!x is true if x is false and false if x is true.

The other types of control flow repeat a block of code. Like the `if` statement, `for` and `while` loops first check a condition is `true` before executing some code. If the condition is `true` they run the code and then go back to the top of the loop to reevaluate the condition.

Here is basic syntax for the `for` loop.

```
for (initialization; condition; increment) {  
    //code  
}
```

The code begin with the initialization of a value to use as a counter; for example, a number or index. Then there is a condition to check based on that value. If that condition is `true`, then the code in the braces is executed. After the code in the braces is executed, the last thing to happen is the counter is incremented. Then the condition is checked again. If `true`, the code is executed again. The counter is incremented. The condition is checked. If `true`, the code is executed. The process continues until the condition is `false` and we break from the loop and continue with our sequential code.

Here is basic syntax for the `while` loop.

```
for (condition) {  
    //code  
}
```

The `while` loop is similar to the `for` loop. We don't use a counter. Instead we start the loop by checking if some condition is `true`. If it's `true`, the code in the braces is executed. Then we go back to the top and check if the condition is `true`. If `true`, the code is executed again. This process continues until the condition is `false`, we break from the loop, and continue on with our code.

Here is basic syntax for the `do-while` loop.

```
do {  
    //code  
} while (condition);
```

The `do-while` loop works a bit differently than the `for` and `while`. The first thing it does, even before checking the condition, is to run the code in the braces once. Then it behaves like a `while` loop. Essentially, it's a `while` loop that always runs at least once.

Here are some other tips concerning control flow:

- Always use braces around the body of a control flow statement. For example, try to predict this output:

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int idx = 0;
```

```

    if (idx == 2)
        cout << idx << endl;
        cout << idx + 1 << endl;

    return 0;
}

```

How can you add braces to make this code clearer?

Using braces appropriately with your control flow will help to prevent errors and make the code easier to read and understand.

- Avoid nesting control flow statements too deeply. If you find yourself nesting control flow statements deeply, it is often a good sign that you can refactor your code. For example, here is code that checks if variables `x`, `y`, `z` are greater than 10, 20, 30, respectively. If one is not, it will print relevant information.

```

#include <iostream>
using namespace std;

int main() {
    int x = 10;
    int y = 20;
    int z = 30;

    if (x > 5) {
        if (y > 10) {
            if (z > 20) {
                cout << "All conditions are met." << endl;
            } else {
                cout << "z is not greater than 20." << endl;
            }
        } else {
            cout << "y is not greater than 10." << endl;
            cout << "(z was not checked)." << endl;
        }
    } else {
        cout << "x is not greater than 5." << endl;
        cout << "(y and z were not checked)." << endl;
    }

    return 0;
}

```

This code can be refactored to reduce the triple nesting to no nesting:

```

#include <iostream>
using namespace std;

```

```

int main() {
    int x = 10;
    int y = 20;
    int z = 30;

    if (x <= 5) {
        cout << "x is not greater than 5." << endl;
    } else if (y <= 10) {
        cout << "y is not greater than 10." << endl;
    } else if (z <= 20) {
        cout << "z is not greater than 20." << endl;
    } else {
        cout << "All conditions are met." << endl;
    }

    return 0;
}

```

The refactored code is easier to follow than the code with nested statements.

- Although `goto` statements are allowed by the language, they should be avoided.
- If you're thinking about using `break` and `continue` statements, make sure you understand how these statements work first!

1.2 Exercises

1.2.1 Predict the output

Predict the output for the follow code (assume it runs error-free).

```

1. #include <iostream>
   using namespace std;

   int main() {
       bool b = false;
       int x = 4;

       if (x < 5) {
           cout << "x is less than 5" << "\n";
       }

       if (b) {
           cout << "x is greater than 6" << "\n";
       }
   }

```

```

    return 0;
}

```

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

```

int main() {
    bool b = false;
    int x = 24;

    if (x > 5 || b) {
        cout << "x is greater than 5 or b is true" << "\n";
    }
    else {
        cout << "x is less than or equal to 5 and b is false" << "\n";
    }

    return 0;
}

```

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

```

int main() {
    bool b = true;
    int x = 24;

    if (b)
    {
        cout << "b is true" << "\n";
        if (x < 20){
            cout << "x is less than 20" << "\n";
        }
        else {
            cout << "x is greater than or equal to 20" << "\n";
        }
    }
}

```

```

    return 0;
}

```

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

```

int main() {
    int x = 7;

    if (x > 20) {
        cout << "x > 20" << "\n";
    }
    else if (x > 10) {
        cout << "x > 10 but less than or equal to 20" << "\n";
    }
    else if (x > 5) {
        cout << "x > 5 but less than or equal to 10" << "\n";
    }
    else {
        cout << "x is less than or equal to 5" << "\n";
    }
    return 0;
}

```

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

```

int main() {
    if (true) {
        cout << "A" << endl;
    } else if (true) {
        cout << "B" << endl;
    } else {
        cout << "C" << endl;
    }

    return 0;
}

```

6. Evaluate the following boolean expressions,

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

int main() {
    cout << (x || y) << endl;
    cout << (y && x) << endl;
    cout << (!( x || y )) << endl;
    cout << ((x && y) || (x && !(x && y))) << endl;
    cout << (!(x && y) == (!x) || (!y)) << endl;

    return 0;
}
```

when we have declared the variables x and y as

- (a) `bool x = true;`
`bool y = false;`
- (b) `bool x = true;`
`bool y = true;`
- (c) `bool x = false;`
`bool y = false;`
- (d) `bool x = false;`
`bool y = true;`

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

```
int main() {
    for (size_t i = 0; i < 5; ++i) {
        for (size_t j = i; j <= 3; j=j+2) {
            cout << j << ' ';
        }
    }
    return 0;
}
```

8. Explain what this code is doing.

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

int main() {

    int n;
    int num;
    int minimum = 1e6;
    int maximum = -1e6;

    cin >> n;

    for (int i = 0; i < n; ++i) {
        cin >> num;
        minimum = min(minimum, num);
        maximum = max(maximum, num);
    }

    cout << "minimum: " << minimum << endl;
    cout << "maximum: " << maximum << endl;

    return 0;
}
```

1.2.2 Practice coding exercises

9. Look at the following code.

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

int main() {
    int idx = 0;
    while (idx < 10) {
        if (idx == 5) {
            break;
        }
        cout << idx << endl;
        ++idx;
    }

    return 0;
}
```

Using what you've learned about control flow, what are two different ways you can improve

this code? Hints:

- Try rewriting it without a **break** statement. Use logic.
- Try rewriting it without a **while** loop.

10. Rewrite this code as a **for** loop. Which is a better choice?

(a) `#include <iostream>`
`using namespace std;`

```
int main() {  
    double sum = 0.0;  
    double num = 2.0;  
  
    while (num <= 200.0) {  
        cout << num << endl;  
        sum += num;  
        num += 2.0;  
    }
```

```
    cout << "The sum of the first 100 positive even numbers is: " << sum << endl;  
  
    return 0;  
}
```

(b) `#include <iostream>`
`using namespace std;`

```
int main() {  
    int num, factorial = 1, idx = 1;  
  
    cout << "Enter a number: ";  
    cin >> num;  
  
    while (idx <= num) {  
        factorial *= idx;  
        idx++;  
    }
```

```
    cout << "The factorial of " << num << " is " << factorial << endl;
```

```

    return 0;
}

```

(c) `#include <iostream>`
`using namespace std;`

```

int main() {
    cout << "Type some letters then press enter. I'll capitalize them." << endl;
    char letter;
    while (cin.peek() != '\n'){
        cin >> letter;
        if (letter > 'Z'){
            letter += 'A' - 'a';
        }
        cout << letter;
    }
    cout << endl;

    return 0;
}

```

11. Rewrite these codes as `while` loops. Which is a better choice?

(a) `#include <iostream>`
`#include <string>`
`using namespace std;`

```

int main(){
    string text = "hello";
    for (int i = 0; i < text.size(); i++) {
        text[i] += 'A' - 'a';
    }
    cout << text << endl;

    return 0;
}

```

(b) `#include <iostream>`
`using namespace std;`

```

int main(){

```

```

    double x = 0.0;
    double y = 1.0;
    for (; y < 10.0; y *= 2.0) {
        x += y;
    }

    return 0;
}

```

(c) `#include <iostream>`
`using namespace std;`

```

int main(){
    cout << "Give me integers, and I'll keep a total." << endl;
    int total = 0;

    for (int current_number; cin >> current_number;) {
        total += current_number;
        cout << "Total:" << total << endl;
    }

    return 0;
}

```

12. An extra leap day occurs on every year that's a multiple of 4. It does not occur on years that are multiples of 100, with the exception of years that are multiples of 400. Ex: 1900 is not a leap year but 2000 is a leap year. The following code was written to determine if a year is a leap year. Rewrite this code without nested if-statements. Then out your code.

```

#include <iostream>
using namespace std;

int main() {
    int year = 1904;

    if (year % 400 == 0) {
        cout << year << " is a leap year." << endl;
    } else {
        if (year % 4 == 0){
            cout << year << " is a leap year." << endl;
        } else {
            if (year % 100 == 0){

```

```

        cout << year << " is NOT a leap year." << endl;
    } else {
        cout << year << " is a leap year." << endl;
    }
}

return 0;
}

```

- (a) By using logic.
- (b) By carefully ordering your conditional statements.

13. Write this code so it uses a do-while loop.

```

#include <iostream>
using namespace std;

int main() {
    cout << "Enter a list of integers separated by spaces" << endl;
    cout << "and I'll tell you their sum" << endl;

    int sum = 0;

    int i; cin >> i;
    sum += i;
    while (cin.get() != '\n') {
        cin >> i;
        sum += i;
    }

    cout << "Their sum is " << sum << endl;

    return 0;
}

```

14. Write a code that sings Old MacDonald with the user in the console. It will start singing and pause, waiting for the user to respond, when it gets to the animal. Once an animal is typed, the console will continue to sing until it's reached the animal noise. Then it will wait for the

user to respond. Then it will continue, using the response until it's finished the verse. It will then start again and expect the user to enter a new animal. The song continues indefinitely until the user gives the responds "no". Then the program ends.

Below is some code to start. You just need to add the control flow.

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

int main() {
    string animal, sound;
    cout << "Sing Old MacDonald!" << endl;

    cout << "Old MacDonald had a farm,\nE-I-E-I-O\n";
    cout << "And on that farm, he had a ";
    cin >> animal;

    cout << "E-I-E-I-O\n";

    cout << "With a ";
    cin >> sound;

    cout << " " << sound << " here, and a "
    << sound << " " << sound << " there,\n";
    cout << "Here a " << sound << ", there a "
    << sound << ", everywhere a " << sound << " " << sound << "\n";
    cout << "Old MacDonald had a farm,\nE-I-E-I-O\n\n";

    return 0;
}
```

15. Write a program that prints the numbers from 0 to 99.
16. Write a program that prints the odd numbers from 101 to 1 (decreasing).
17. Write a program that sums integers starting at 1 to an integer n . Try doing it by increasing the counter $(1 + 2 + 3 + \dots + n)$ and decreasing the counter $(n + (n - 1) + (n - 2) + \dots + 1)$.

18. Verify that the solution to the sum

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots = 1 + \sum_{n=1}^N \frac{1}{2^n} = 2.$$

Try this for large N . You will see how programming can be a powerful tool for checking analytic solutions. No way we'd want to do the sum 100 times. (How many times do we need to sum?)

19. Here's a few examples where it verifying math computationally goes wrong.

(a) It is a fact in mathematics that

$$\lim_{x \rightarrow 0} \frac{1 - \cos(x)}{x^2} = 0.5$$

What's the output of the following code?

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

int main() {
    double x, y;
    cout << boolalpha;

    x = 1e-7;
    y = (1 - cos(x)) / (x * x);
    cout << ((0.4 < y) && (y < 0.6)) << endl;

    x = 1e-8;
    y = (1 - cos(x)) / (x * x);
    cout << ((0.4 < y) && (y < 0.6)) << endl;

    return 0;
}
```

(b) `#include <iostream>`
`#include <cmath>`
`using namespace std;`

`int main() {`
 `double x, y;`
 `cout << boolalpha;`

```

double a, b, c;
a = 1e-18;
b = 1;
c = -1;

x = (-b + sqrt(b*b - 4*a*c)) / (2*a);
cout << ((0.9 < x) && (x < 1.1)) << endl;

return 0;
}

```

20. Write a code to compute the sum

$$1 - 2 + 3 - 4 + 5 - \dots + (-1)^{(n+1)}n$$

where n can be any integer you chose. Hint: when does the sign change?

21. Write code that counts the number of a's in a string like the one below.

```

string sentence = "We're given an input string and we want to count the
a's in that string.";

```

22. Write a program that starts with

```
int N = 4;
```

This is done so that we can change the value of N easily later.

Write the rest of the program so that (with N set to be 4) it prints:

```

*
**
***
****
***
**
*

```

- (a) First, declare an `int i = 10;` and write a `for` loop that uses `i` in its keep-going condition to print 10 *s. In the second part, you might remove the declaration of `i`.
- (b) In what we want to print, the rows are changing in size, starting with 1 *, maxing out at `N` *, and decreasing back down to 1 *. We need to use the code from the first part, but with different values of `i`. Use two `for` loops, one with increasing `i`, one with decreasing `i`, and the code from part (a) nested within each.
23. Use a nested `for` loop to produce the multiplication table for integers up to 10. To print things out nicely, use `setw` from the file `iomanip`. This allows you to set the field width so that each column of data aligns neatly. Here's an example of it being used:

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

int main() {
    cout << "Magic Square 3x3:" << endl;
    cout << "-----" << endl;

    cout << setw(3) << 2;
    cout << setw(3) << 7;
    cout << setw(3) << 6;
    cout << endl;

    cout << setw(3) << 9;
    cout << setw(3) << 5;
    cout << setw(3) << 1;
    cout << endl;

    cout << setw(3) << 4;
    cout << setw(3) << 3;
    cout << setw(3) << 8;
    cout << endl;

    return 0;
}
```

Console output:

```
Magic Square 3x3:
-----
 2  7  6
```

```
9 5 1
4 3 8
```

1.2.3 Error spotting exercises

Error spotting will be on the exam. Look at the follow code below and state:

- What errors occurred (build, run time, or logic) or if there are none.
- In the case of a logic error, what does the code currently accomplish?
- How the code can be fixed (use the line numbers to help with the description).

A description of how the code should perform will be put in a block comment. After you make your initial guess, type the code in your IDE and run it. Copying very carefully may help you find some of the errors. Play around with the code to fix it.

```
24. 1  #include <iostream>
    2  using namespace std;
    3
    4  /* This code should print:
    5  3
    6  4
    7  */
    8  int main() {
    9      for (size_t i = 0; i <= 4; ++i) {
   10          if (i > 2) {
   11              cout << i << "n";
   12          }
   13      }
   14      return 0;
   15  }
```



```
25. 1  #include <iostream>
    2  using namespace std;
    3
    4  /* This code is meant to print out 0 1 1 2 2 3 3 4. */
    5  int main() {
    6      for (size_t i = 0; i <= 3; ++i) {
    7          for (size_t j = 0; j < 2; ++j) {
    8              cout << j + i << ' ';
```

```

9      }
10     return 0;
11 }

```

26.

```

1  #include <iostream>
2  using namespace std;
3
4  /* The user is prompted to type a string of letters and then press Enter.
5     The code finishes after returning all the letters all capitalized. */
6  int main() {
7      cout << "Type some letters then press enter. I'll capitalize them. " << endl;
8      char letter;
9
10     while (letter != '\n') {
11         cin >> letter;
12         if (letter > 'Z') {
13             letter += 'A' - 'a';
14         }
15         cout << letter;
16     }
17     cout << endl;
18
19     return 0;
20 }

```

27.

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  /* Counts the number of vowels in a string s .
6     Output: The string "Hello, World!" contains 3 vowels. */
7  int main() {
8      string s = "Hello, World!";
9      int count = 0;
10     int idx = 0;
11     while (idx < s.length()) {
12         if (s[idx] == 'a' || s[idx] == 'e' || s[idx] == 'i' ||
13             s[idx] == 'o' || s[idx] == 'u') {
14             count++;

```

```
15         }
16
17     }
18     cout << "The string \"" << s << "\" contains " << count << " vowels.\n";
19
20     return 0;
21 }
```