CIS 129
Advanced Computer Programming

Chapter 3: Flow of Control

Mr. Horence Chan

Control Structures

- **Control structures** are portions of program code that contain statements within them and, depending on the circumstances, execute these statements in a certain way.
- There are typically two kinds:
  - _______________
  - _______________

Conditionals

- **Conditionals** allow the program to check the values of variables and to execute (or not execute) certain statements.
- C++ has _____ and __________ conditional structures.

If / else
Switch
Corporate needs you to find the differences between this picture and this picture.
Compilers
They're the same picture.

Operators

- Conditionals use two kinds of special operators:
  - Relational
  - Logical
- These are used to determine whether some condition is true or false.
- The ________________ **operators** are used to test a relation between two expressions
- The ________________ **operators** are often used to combine relational expressions into more complicated Boolean expressions

Operators

Relational Operator

Operators	Meaning
>	Greater than
	Greater than or equal to
<	Less than
	Less than or equal to
	Equal to
	Not equal to

Logical Operator

Operators	Meaning
	And
	Or
	Not

- An expression using operators return a Boolean value of either `true` or `false`, indicating whether the relation tested for holds, which is also called a Boolean expression.
- E.g. if the variables `x` and `y` have been set to 6 and 2, respectively, then `x > y` returns `true`. Similarly, `x < 5` returns `false`.

Operators

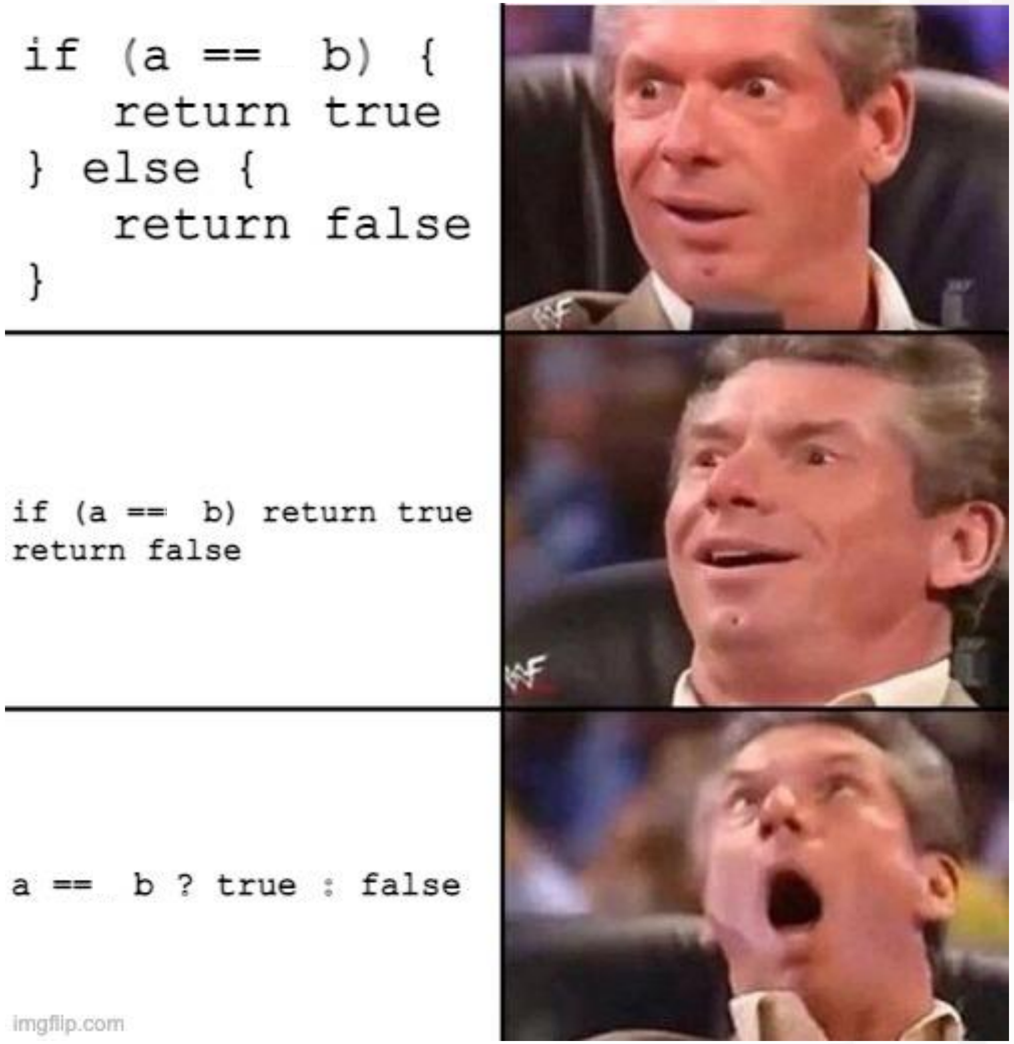
- More example: (assume x = 6 and y = 2)
    - `!(x > 2) → ___________`
    - `(x > y) && (y > 0) → ___________`
    - `(x < y) && (y > 0) → ___________`
    - `(x < y) || (y > 0) → ___________`
- In fact, any kind of value can be used in a Boolean expression in C++:
    - `False`: represented by 0
    - `True`: anything that is not 0
- Any variable holding a non-zero value is true.
    - `"Hello, world!" → ___________`
    - `2 → ___________`
    - `!x → ___________`
    - `x && y → ___________`

if, then, else

```
if(condition1)
{
        statementA1
        statementA2
        …
}
else if(condition2)
{
        statementB1
        statementB2
        …
}
else
{
        statementC1
        statementC2
        …
}
```

- If `condition1` is met, the block corresponding to the `if` is executed.
- ______________ is used in each block.
- If not, then *only if* `condition2` is met is the block corresponding to the `else if` executed.
- If none of the previous conditions are met, the `else` block is executed.
- There may be more than one `else if`, each with its own condition.
- Once a block whose condition was met is executed, any `else if`s after it are ignored.
- In this structure, one of the blocks must execute.

if, then, else



```
if (a ==  b) {
    return true
} else {
    return false
}
```

```
if (a ==  b) return true
return false
```

```
a ==  b ? true : false
```

imgflip.com

- _______ can be omitted if there is only one statement

- For beginner in C++, it is recommend to write { } in each block at the first time
- This help to reduce complication error during editing

if, then, else

```
#include <iostream>
using namespace std;
          int main() {
          int x = 6;
          int y = 2;
                    if(x > y){
                    cout << "x is greater than y\n";
                    }
                    else if(y > x){
                    cout << "y is greater than x\n";
                    }
                    else{
                    cout << "x and y are equal\n";
                    }
          return 0;
          }
```

- Output : `x is greater than y`

- If we replace `int x = 2; int y = 6`
- Output: ______________________________

- If we replace `int x = 2; int y = 2`
- Output: ______________________________

switch-case

```
switch(expression)
{
        case constant1:
        statementA1
        statementA2
        ...
        break;
        case constant2:
        statementB1
        statementB2
        ...
        break;
        ...
        default:
        statementZ1
        statementZ2
        ...
        break;
}
```

- If `expression` is equal to `constant1`, then the statements below `case constant1:` are executed until a _____________ is encountered.
- If `expression` is not equal to `constant1`, then it is compared to `constant2`. If these are equal, then the statements below `case constant2:` are executed until a _____________ is encountered.
- If not, then the same process repeats for each of the constants, in turn.
- If none of the constants match, then the statements below `default:` are executed.
- _____________ are not necessary for cases.

switch-case

```
#include <iostream>
using namespace std;
        int main() {
        int x = 6;
                switch(x) {
                case 1:
                cout << "x is 1\n";
                break;
                case 2:
                case 3:
                cout << "x is 2 or 3";
                break;
                default:
                cout << "x is not 1, 2, or 3";
                break;
                }
        return 0;
        }
```

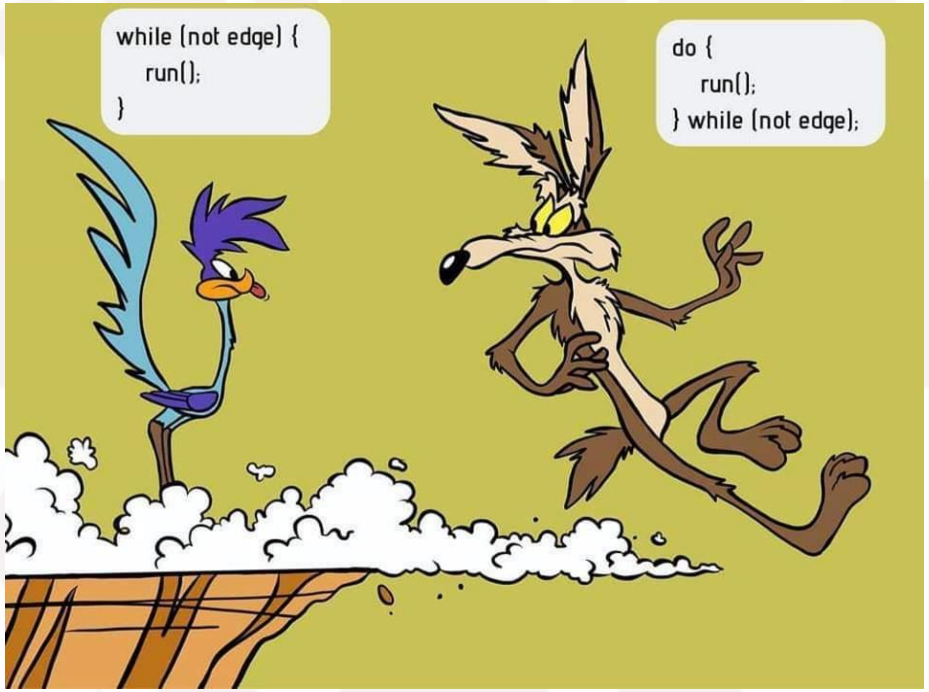
- Output: `x is not 1, 2, or 3`

- If we replace with `int x = 2`
- Output: ____________________

- Note how to write the expression when two cases have the __________ output

While and do-while loop

```
while(condition)
{
        statement1
        statement2
        …
}
```



```
do
{
        statement1
        statement2
        …
}
while(condition);
```

- As long as condition holds, the block of statements will be repeatedly executed
- *do-while* loop is a variation that guarantees the block of statements will be executed at least __________

While loop

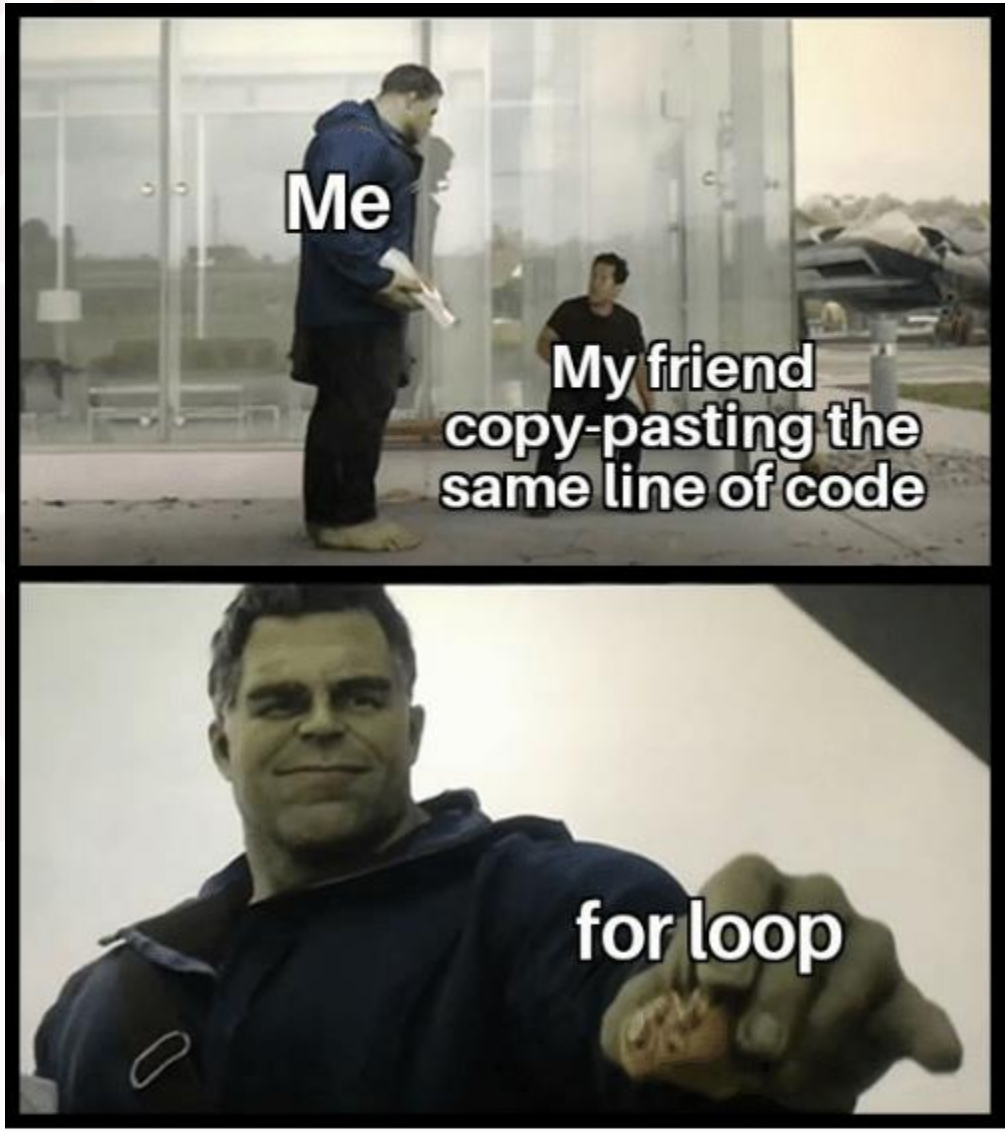
```
#include <iostream>
using namespace std;
    int main() {
    int x = 0;
        while(x < 10){
        x = x + 1;
        }
cout << "x is " << x << "\n";
return 0;
}
```

Output: ______________________

for loop

```
for(initialization; condition; incrementation)
{
        statement1
        statement2
        …
}
```

for loop is designed to allow a counter variable that is initialized at the beginning of the loop and incremented (or decremented) on each iteration of the loop.



Programming_irl

for loop

```
#include <iostream>
using namespace std;
int main() {
    for(int x = 0; x < 5; x = x + 1)
    {
    cout << x << "\n";
    }
return 0;
}
```

Output:

for loop

```
#include <iostream>
using namespace std;
int main() {
     int x = 0;
     for(; x < 5; x = x + 1)
     {
     cout << x << "\n";
     }
return 0;
}
```

- If the counter variable is already defined, there is no need to define a new one in the initialization portion of the for loop.
- Note that the _________________ inside the for loop's parentheses is still required.

for and while loop

```
#include <iostream>
using namespace std;
int main() {
    int x = 0;
    for(; x < 5; x = x + 1)
    {
    cout << x << "\n";
    }
return 0;
}
```

```
#include <iostream>
using namespace std;
int main() {
    int x = 0;
    while(x < 5) {
    cout << x << "\n";
    x = x + 1;
    }
return 0;
}
```

Both have the same output!

for and while loop

```
#include <iostream>
using namespace std;
int main() {
    int x = 0;
    int y = 0;
    while(y < 5) {
    cout << x << "\n";
    x = x + 1;
    }
    cout << "Why can’t print me :(";
return 0;
}
```

- Please pay attention when writing the condition in for / while loop !!!
- Please make sure a loop has exit condition. (or entry condition)

when you forget to write an exit condition for your while loop
This little maneuver is gonna cost us 51 years

Nested if conditionals

```
#include <iostream>
using namespace std;
      int main() {
      int x = 6;
      int y = 0;
            if(x > y) {
            cout << "x is greater than y\n";
                  if(x == 6) {
                  cout << "x is equal to 6\n";
                  } else{
                  cout << "x is not equal to 6\n";
                  }
            } else
            cout << "x is not greater than y\n";
return 0;
}
```

Output:

Nested loops

```
#include <iostream>
using namespace std;
      int main() {
      for(int x = 0; x < 4; x++) {
           for(int y = 0; y < 4; y++){
           cout << y;
           }
      cout << "\n";
      }
return 0;
}
```

- ‘x++’ means ___________

Output:

Simple file input and output

- We have a text file showing the price of each products

- We want to covert the price list into full sentence

- Input file

```
burger      15
fries       11
ice-cream   9
```

- Output file

```
The price of burger is $15.
The price of fries is $11.
The price of ice-cream is $9.
```

Simple file input and output

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
using namespace std;
int main(){
         //Declare variables
         ifstream inFile;
         ofstream outFile;
         string burger_name, fries_name, icecream_name;
         int burger_price, fries_price, icecream_price;
         //Open the input file and output file
         inFile.open("price.txt");
         if (!inFile) {
                  cout << "Cannot open the input file."
                       << "The program terminates." << endl;
         return 1;
         }
         outFile.open("price_output.out");
         cout << "Processing data" << endl;
```

- `fstream`: Stream class to both _______ and _______ from/to files.
- `string`: for using `string` variable
- `ifstream`: variable to _______ from files
- `ofstream`: variable to _______ on files
- `.open()`: open a file
- Check if the file “price.txt” exists, if not the program terminates

Simple file input and output

```
//Read file word by word
inFile >> burger_name >> burger_price;
inFile >> fries_name >> fries_price;
inFile >> icecream_name >> icecream_price;
//Output file
outFile << "The price of " << burger_name
        << " is $" << burger_price <<"." << endl;
outFile << "The price of " << fries_name
        << " is $" << fries_price << "." << endl;
outFile << "The price of " << icecream_name
        << " is $" << icecream_price << "." << endl;
inFile.close(); // .close(): close a file
outFile.close();
cout << "Processing completed" << endl;
return 0;
}
```

- Input file

```
burger      15
fries       11
ice-cream   9
```

- Output file

```
The price of ________ is _____.
The price of ________ is _____.
The price of ________ is _____.
```

- Remember to close the input and output file at the end!