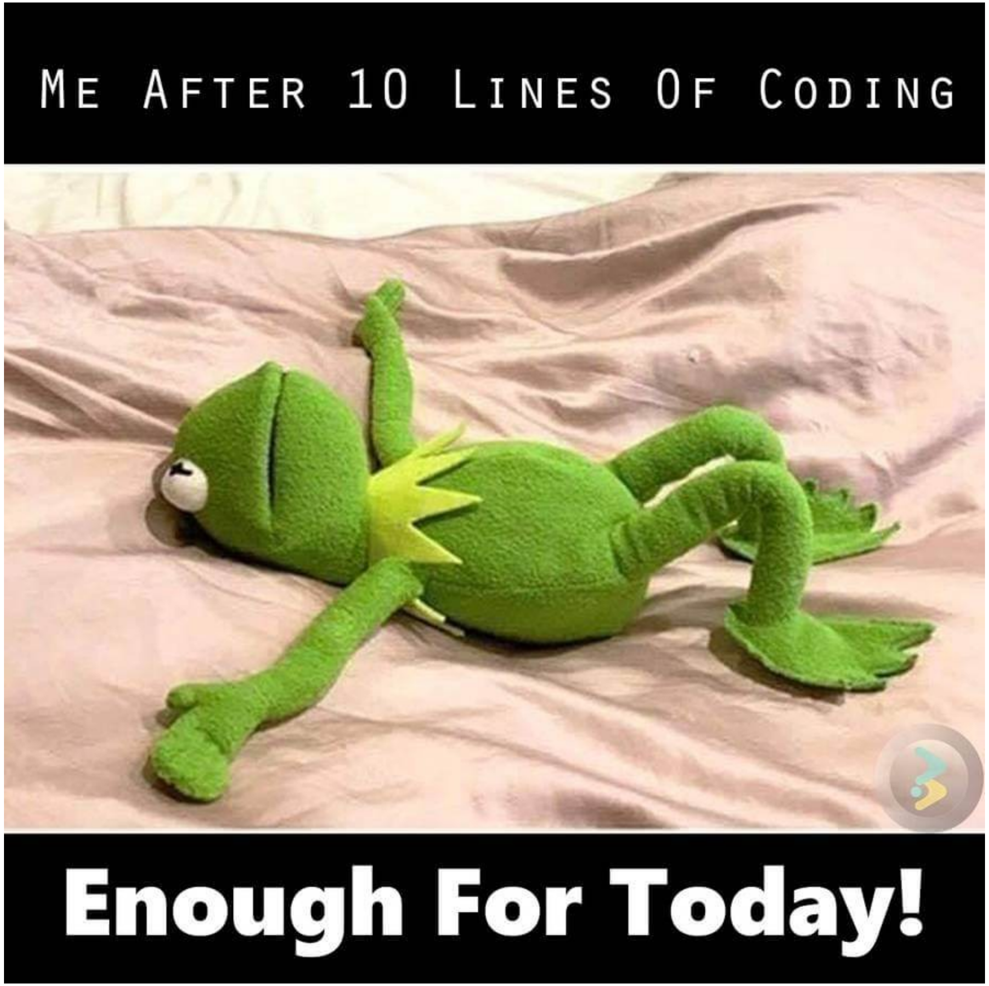

Me After 10 Lines Of Coding
Enough For Today!

CS1428
Foundation of Computer Science

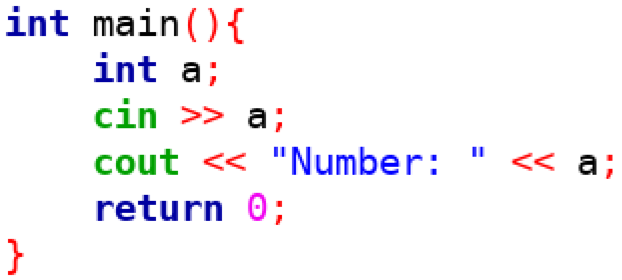
Lecture 4: Branching

Branching:

- Programs can choose to execute some instructions or not based on a “condition”.
- This makes programs responsive to user input.
- Programs can change their output based on computed conditions.
- The **if...else** block is the simplest form of branching.

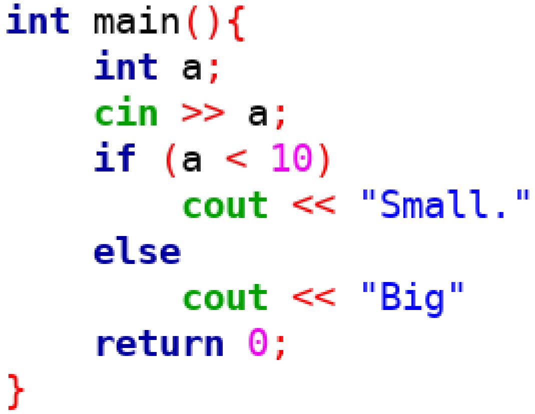
Branching:

Un-branched Program

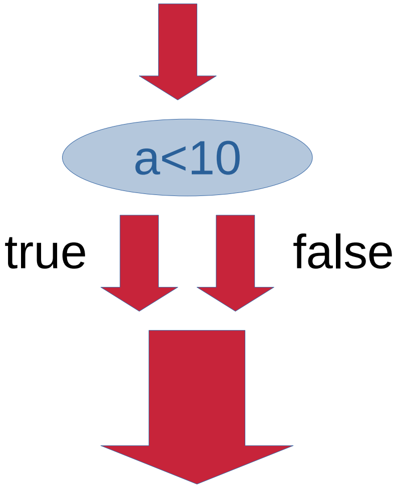


```
int main(){
    int a;
    cin >> a;
    cout << "Number: " << a;
    return 0;
}
```

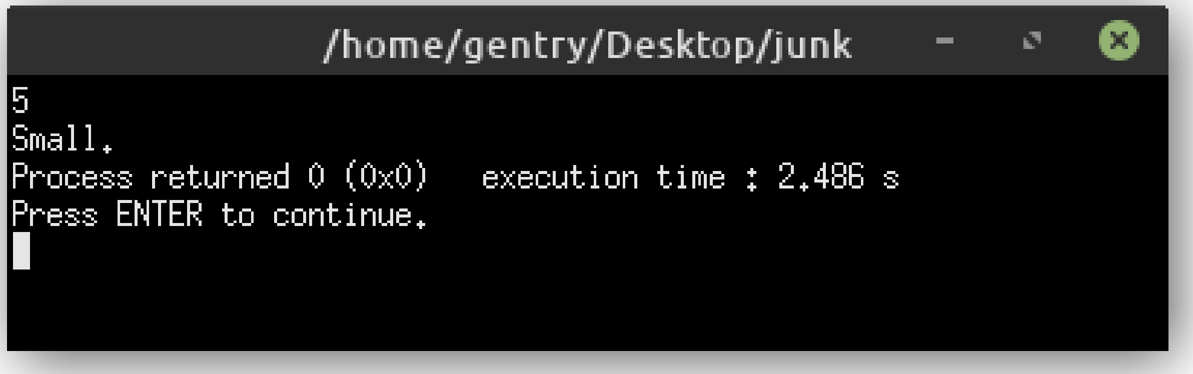
Branched Program



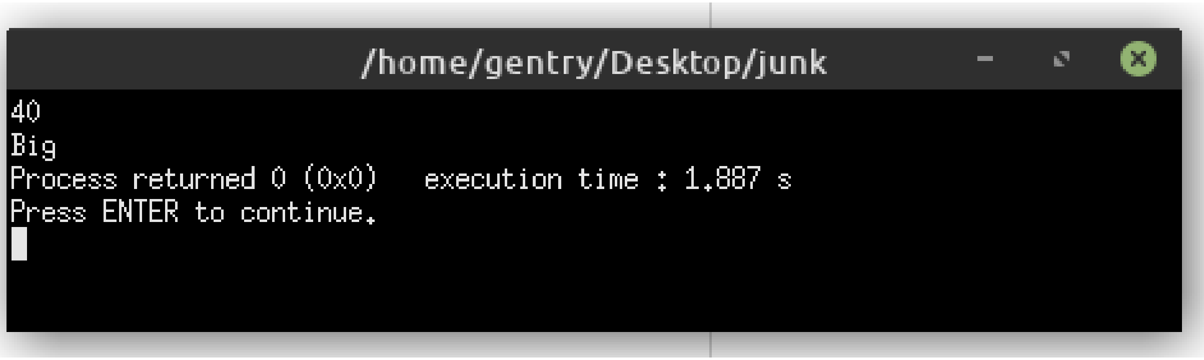
```
int main(){
    int a;
    cin >> a;
    if (a < 10)
        cout << "Small."
    else
        cout << "Big"
    return 0;
}
```



Branched Example Output:



```
5
Small.
Process returned 0 (0x0)   execution time : 2.486 s
Press ENTER to continue.
```

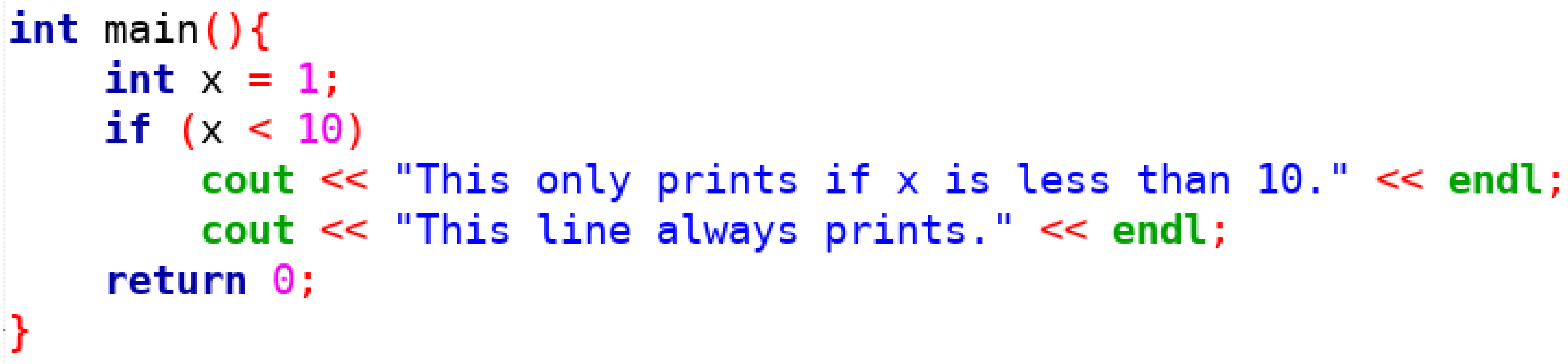


```
40
Big
Process returned 0 (0x0)   execution time : 1.887 s
Press ENTER to continue.
```

Syntax of if...else:

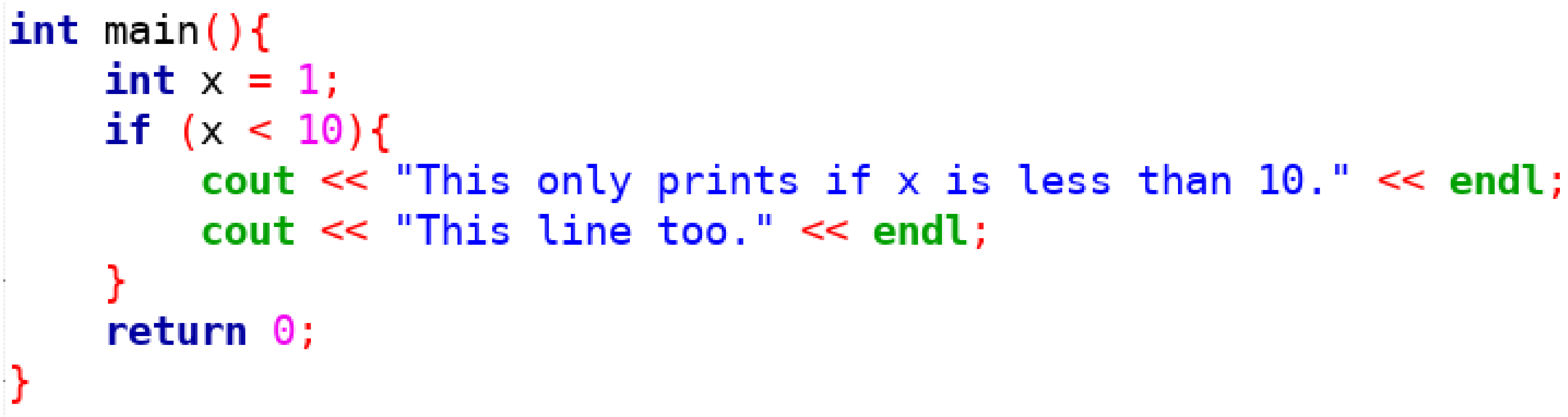
- if is a reserved keyword in C++ that begins a conditional statement.
- Every conditional statement has to have a condition in parenthesis.
- Every condition must resolve to **true** or **false** (boolean values).
- The if applies to everything in the {curly brackets} that follow it or just the one line following if there are no curly brackets
- An else can follow the if block optional. The statements in the else block are executed only if the condition is **false**.

if blocks without brackets:



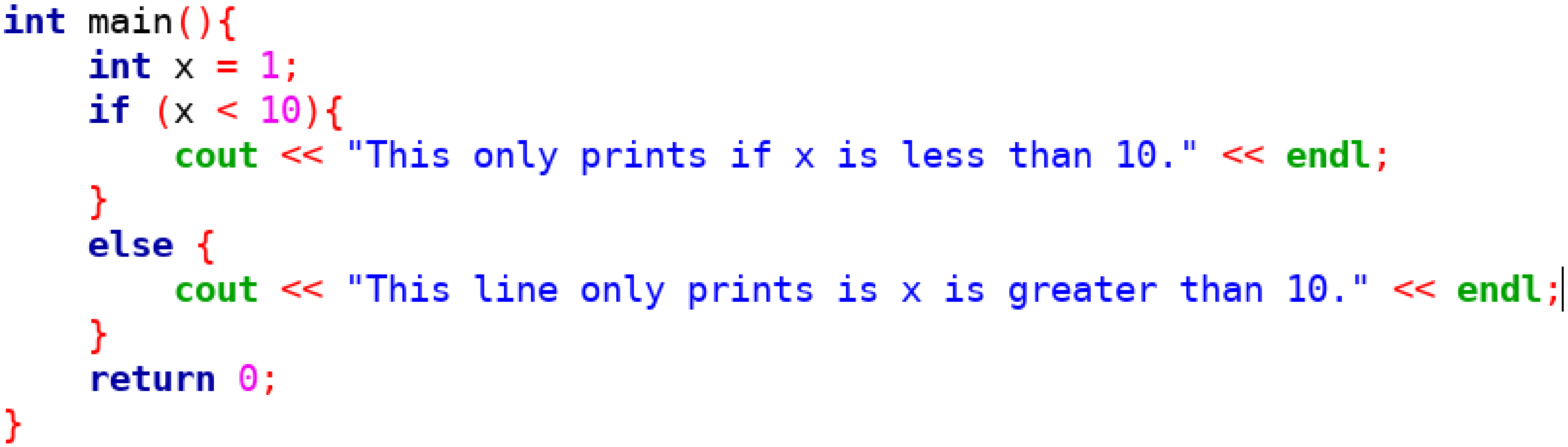
```
int main(){
    int x = 1;
    if (x < 10)
        cout << "This only prints if x is less than 10." << endl;
        cout << "This line always prints." << endl;
    return 0;
}
```

if blocks with brackets:



```
int main(){
    int x = 1;
    if (x < 10){
        cout << "This only prints if x is less than 10." << endl;
        cout << "This line too." << endl;
    }
    return 0;
}
```

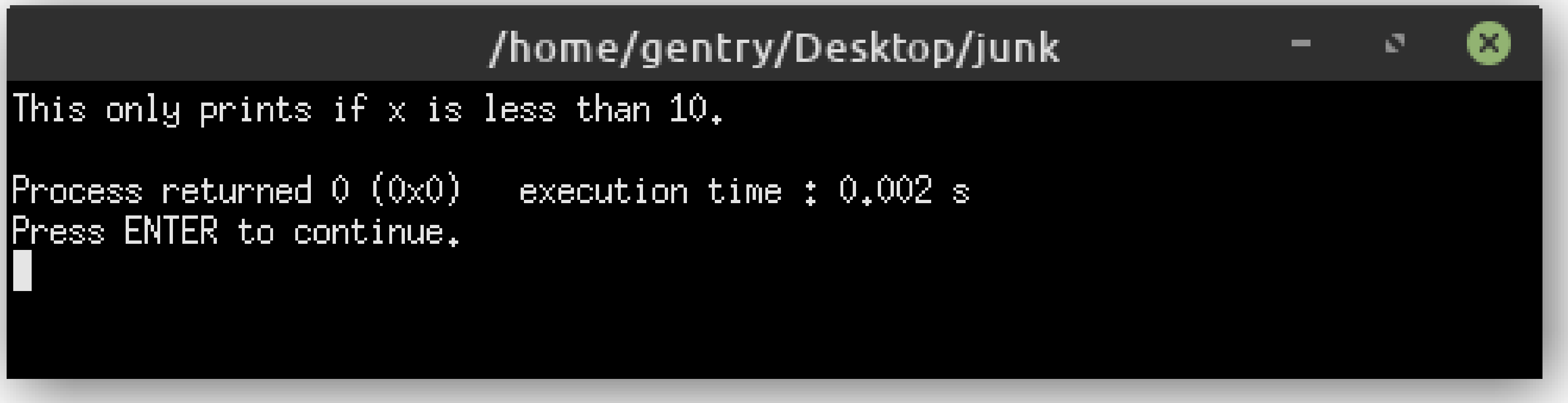
if...else blocks:



```
int main(){
    int x = 1;
    if (x < 10){
        cout << "This only prints if x is less than 10." << endl;
    }
    else {
        cout << "This line only prints is x is greater than 10." << endl;
    }
    return 0;
}
```

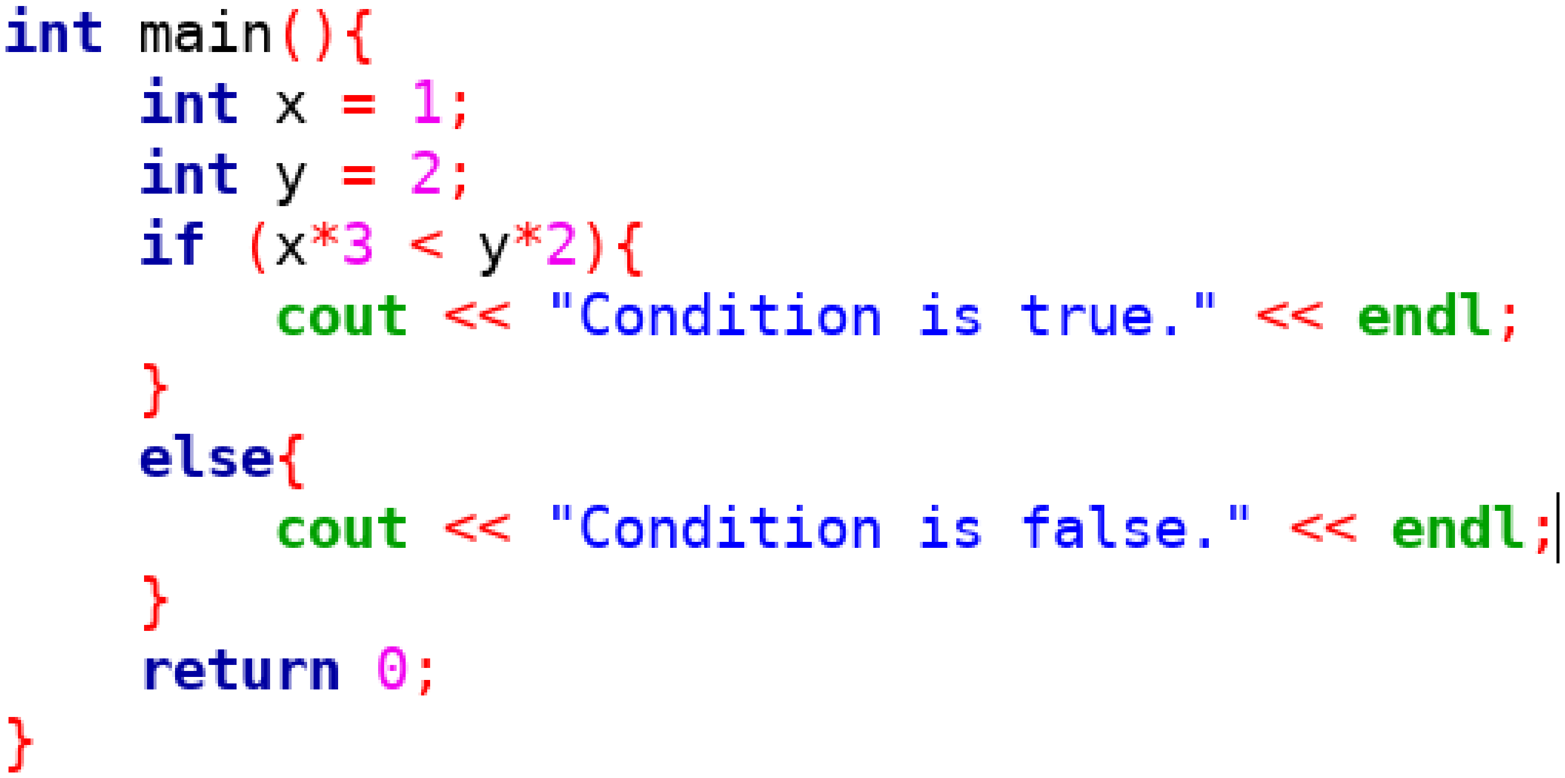
The else is actually >=. Do you see why?

if...else Output:



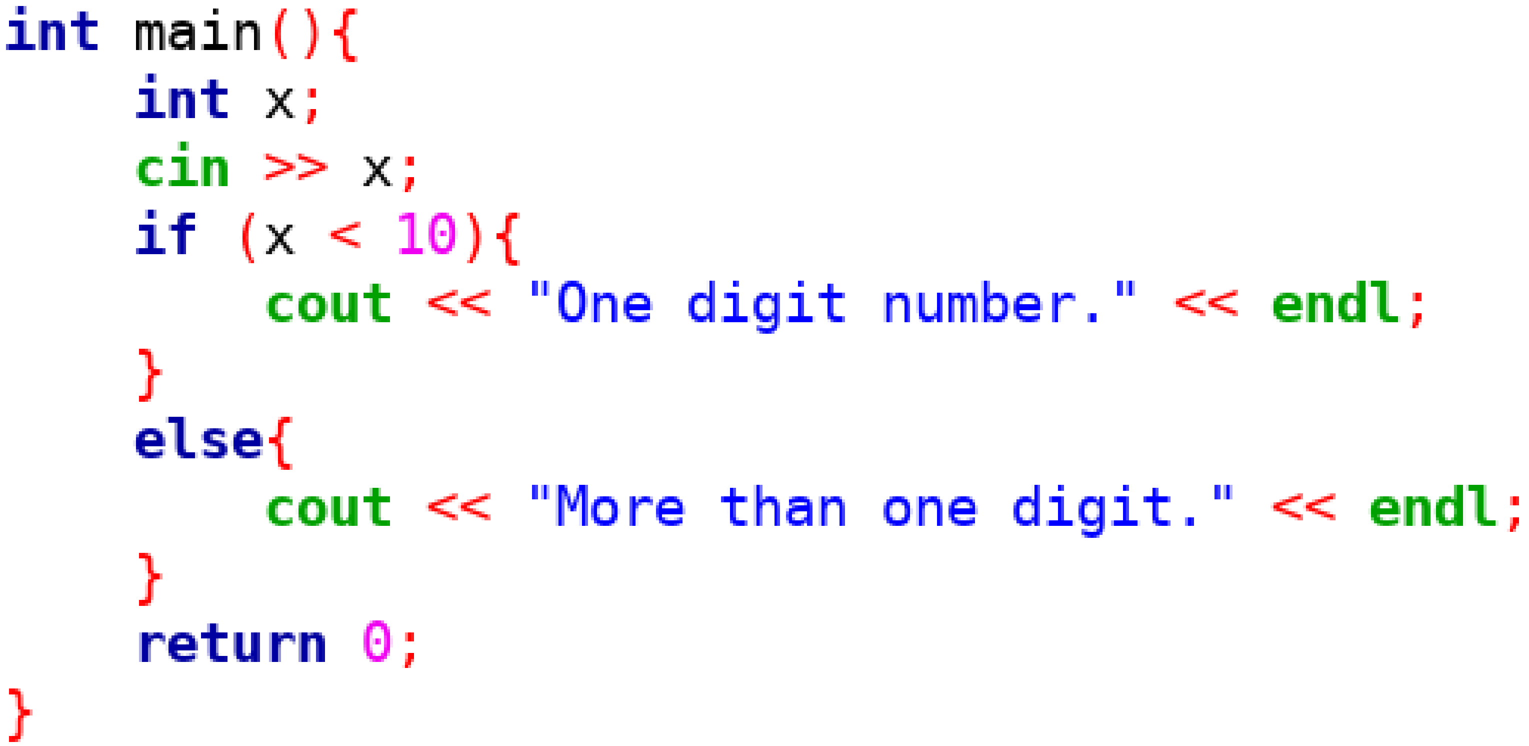
```
This only prints if x is less than 10.

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Conditions can be operations:



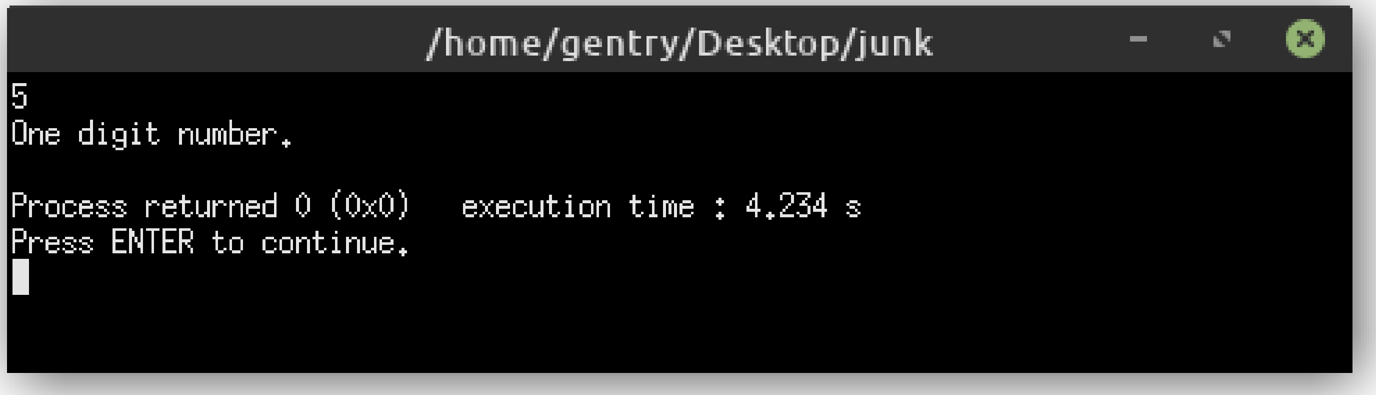
```
int main(){
    int x = 1;
    int y = 2;
    if (x*3 < y*2){
        cout << "Condition is true." << endl;
    }
    else{
        cout << "Condition is false." << endl;
    }
    return 0;
}
```

Conditions can be user input:

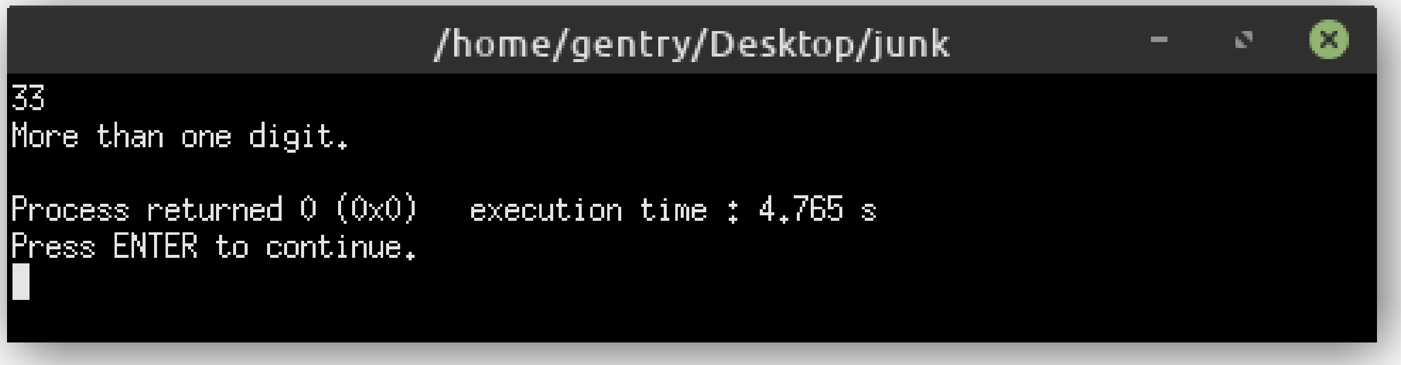


```
int main(){
    int x;
    cin >> x;
    if (x < 10){
        cout << "One digit number." << endl;
    }
    else{
        cout << "More than one digit." << endl;
    }
    return 0;
}
```

Output from User Input:



```
5
One digit number.

Process returned 0 (0x0)   execution time : 4.234 s
Press ENTER to continue.
```



```
33
More than one digit.

Process returned 0 (0x0)   execution time : 4.765 s
Press ENTER to continue.
```

Comparison Operators:

- Always produce a true or false value.
- Resolved after arithmetic operators (after PEMDAS)
- Provided by C++:
  - **<** less than
  - **>** greater than
  - **<=** less than or equal to
  - **>=** greater than or equal to
  - **==** equal to
  - **!=** not equal to

Comparison Examples:

1 < 3	true
3 < 3	false
3 <= 3	true
4 > 3	true
4 != 3	true
4 != 4	false
4 == 4	true
(3 – 1) < 3	true

= vs. ==:

=

- assignment operator
- changes values stored in memory
- Think of it as “takes the value”

==

- comparison operator
- evaluates values for equality and returns true of false.
- Think of it as “is equal to”

Logical Operators:

- **&&** “and” returns true if both sides are true and false otherwise.
- **||** “or” returns false if both sides are false and true otherwise.
- Both sides of **&&** or **||** have to be valid logical expressions.
- **!** “not” returns the opposite of its one operand. This makes **not** a “unary” operator.

AND examples:

```
1 < 3 && 3 > 1
(1 < 3) && (3 > 1)
true && true
true
```

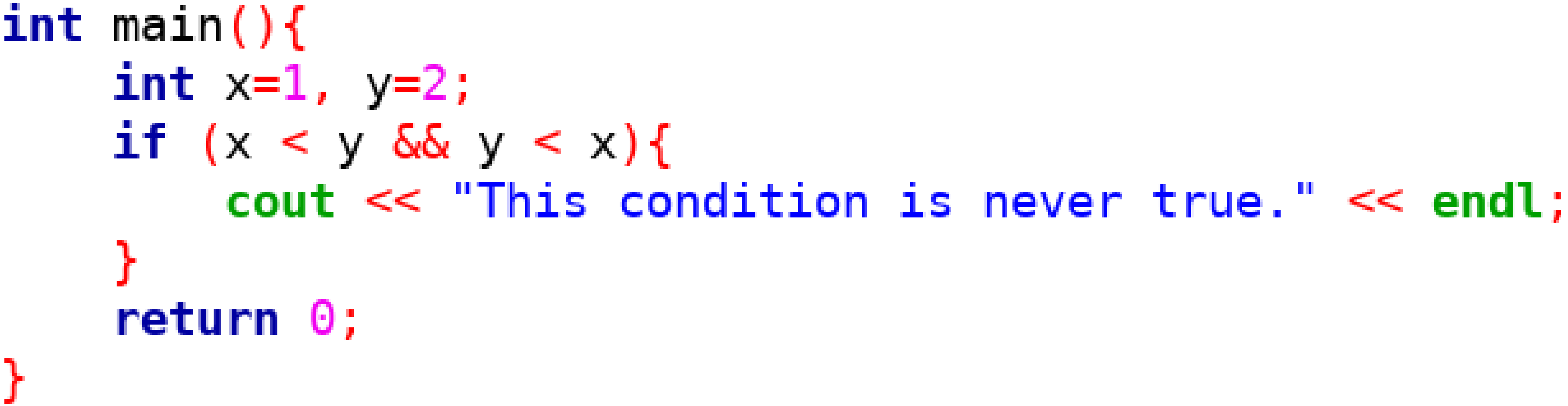
```
1 < 3 && 3 > 10
(1 < 3) && (3 > 10)
true && false
false
```

OR examples:

```
1 < 3 || 3 > 10
(1 < 3) || (3 > 10)
true || false
true
```

```
10 < 3 || 3 > 10
(10 < 3) || (3 > 10)
false || false
false
```

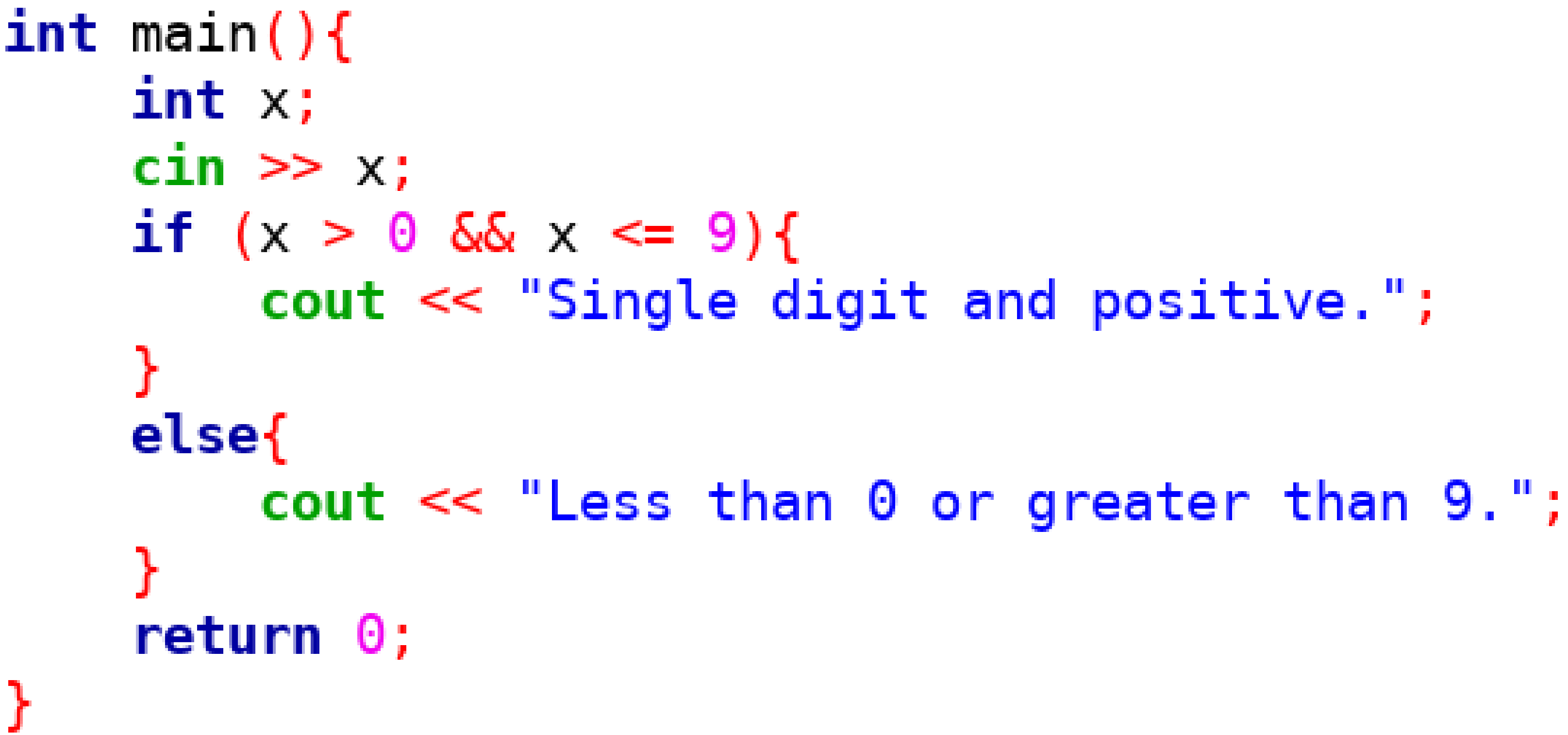
Code example of compound conditions:



```
int main(){
    int x=1, y=2;
    if (x < y && y < x){
        cout << "This condition is never true." << endl;
    }
    return 0;
}
```

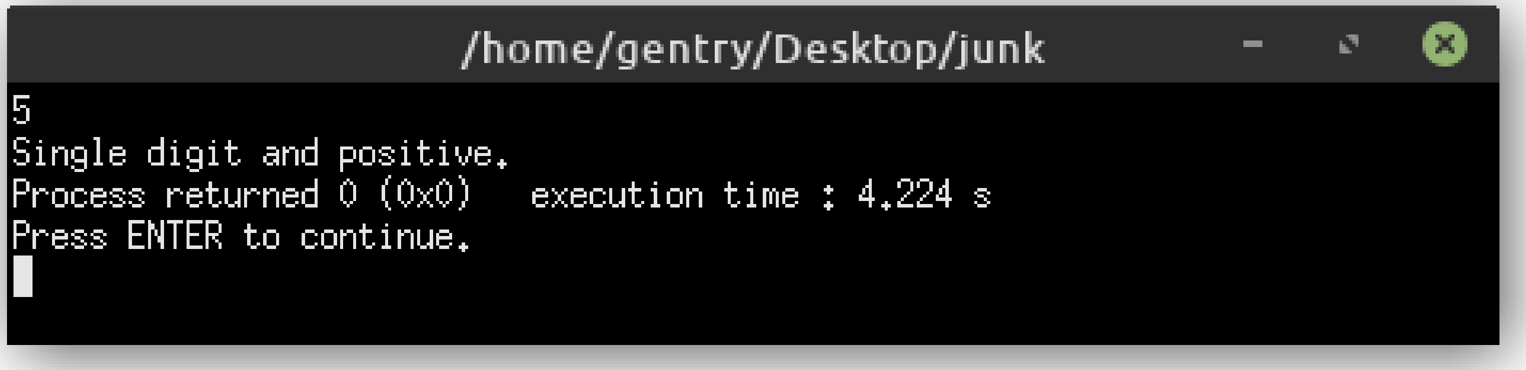
Why is the condition never true?

Code example of compound conditions:

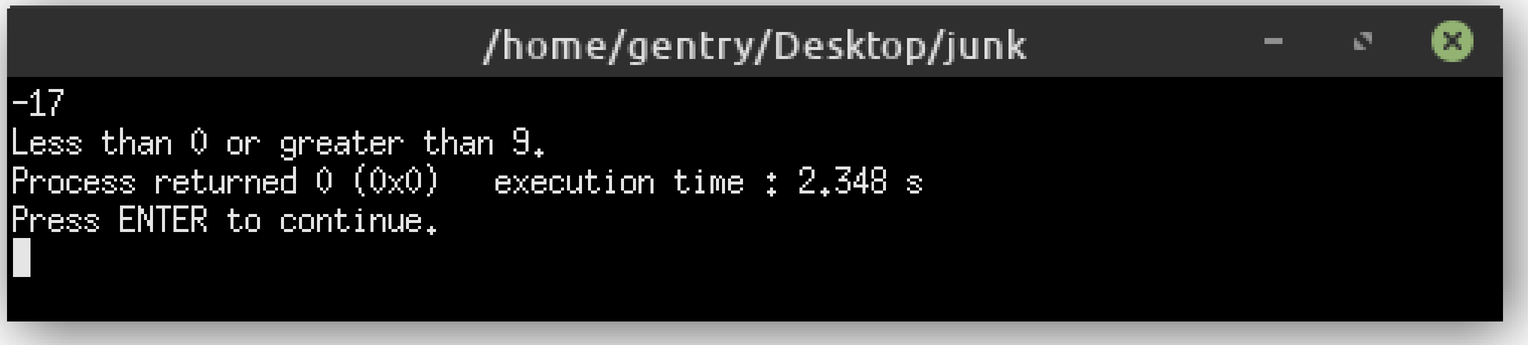


```cpp
int main(){
    int x;
    cin >> x;
    if (x > 0 && x <= 9){
        cout << "Single digit and positive.";
    }
    else{
        cout << "Less than 0 or greater than 9.";
    }
    return 0;
}
```

Output from User Input:



```
5
Single digit and positive.
Process returned 0 (0x0)   execution time : 4.224 s
Press ENTER to continue.
```

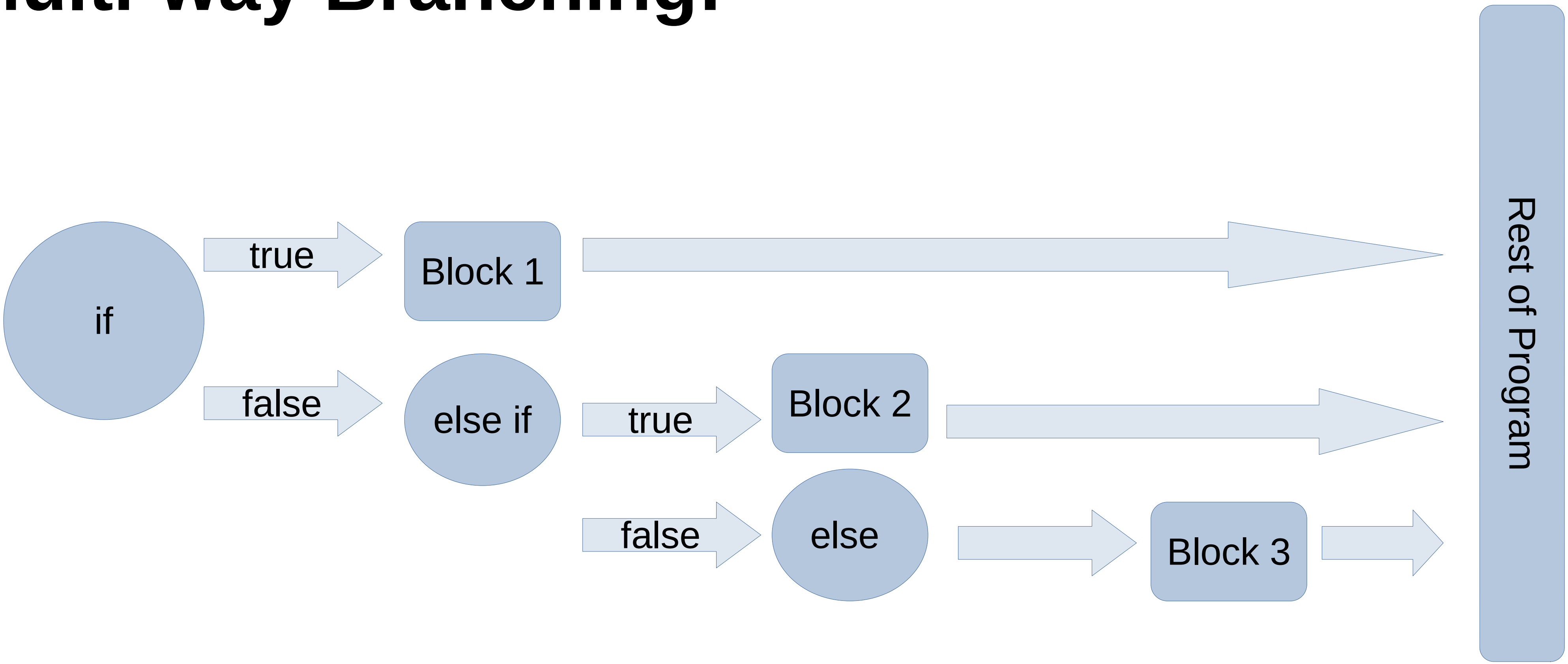


```
-17
Less than 0 or greater than 9.
Process returned 0 (0x0)   execution time : 2.348 s
Press ENTER to continue.
```

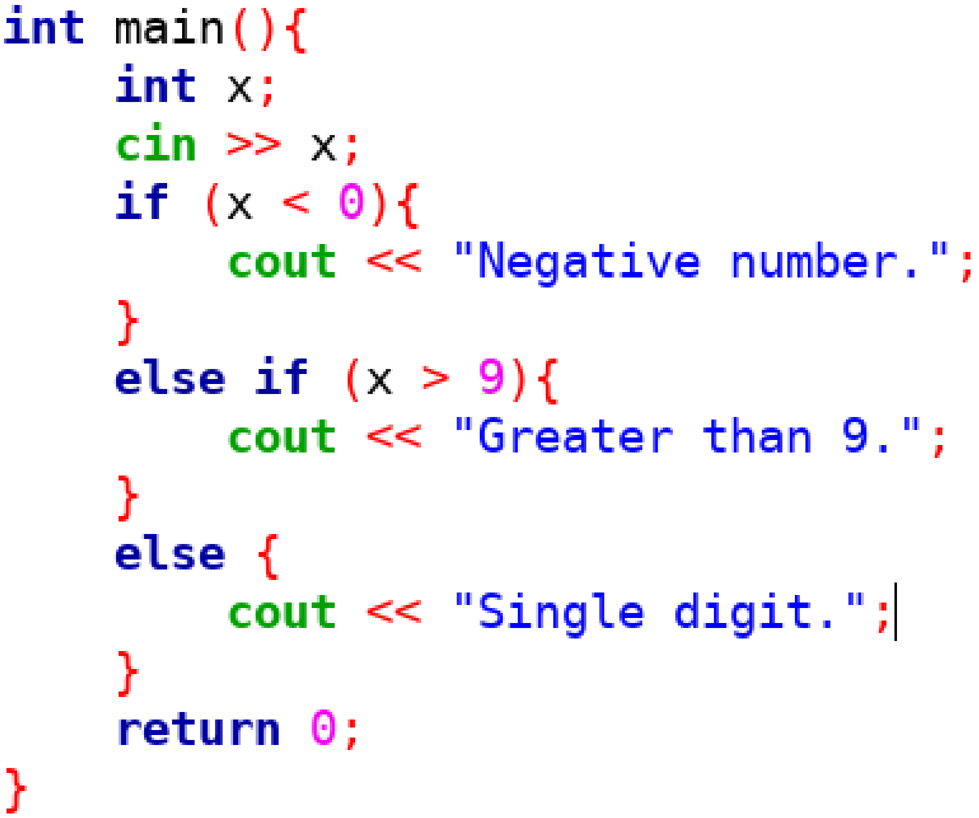
Multi-way Branching:

- **if...else** let us implement 2-way branching, but often we want to write programs that follow three or more paths.
- We can add as many **else if** block between the **if** and the **else** as we want.
- Every **else if** needs its own condition. Only the **else** does not need a condition.

Multi-way Branching:

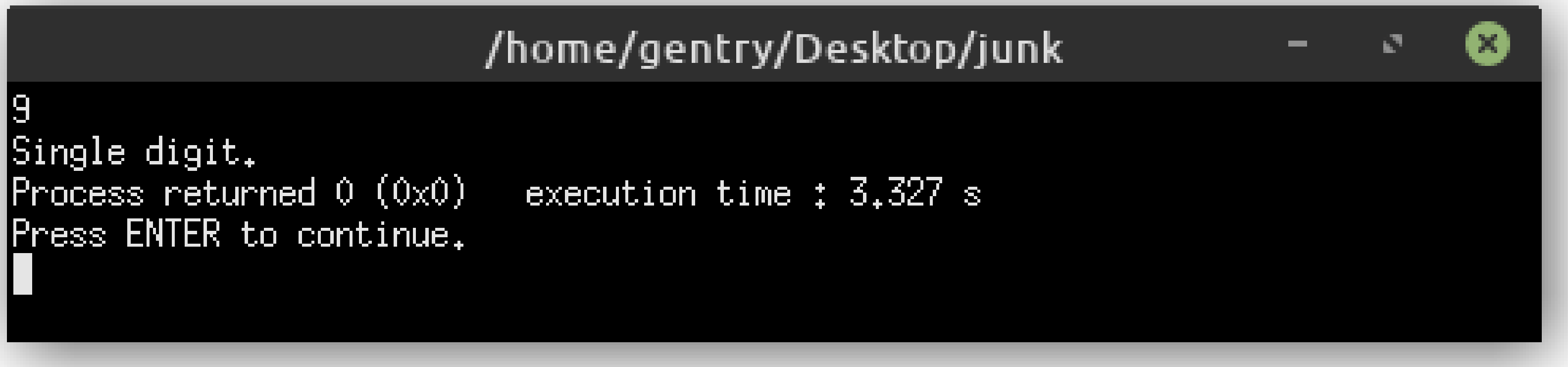


3 Way Branching:

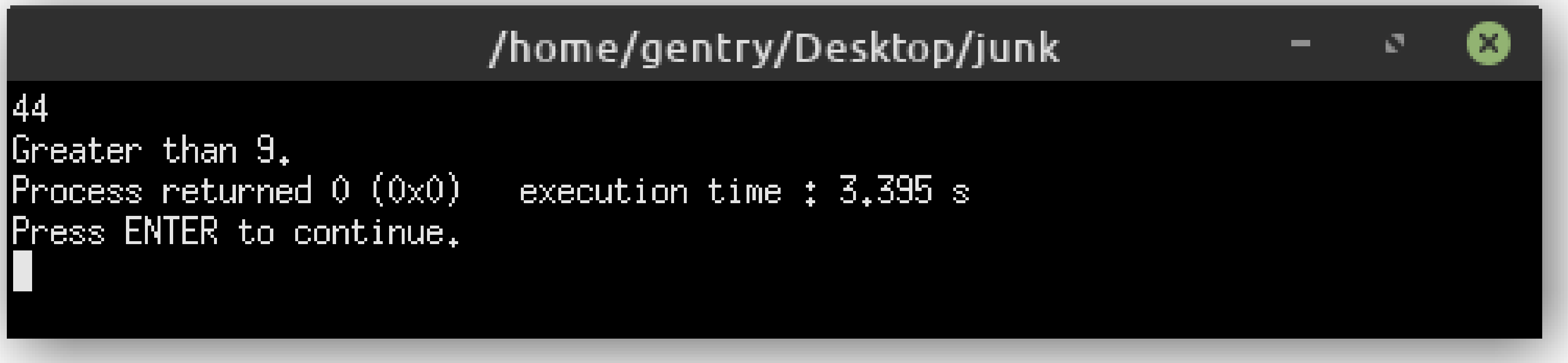


```
int main(){
    int x;
    cin >> x;
    if (x < 0){
        cout << "Negative number.";
    }
    else if (x > 9){
        cout << "Greater than 9.";
    }
    else {
        cout << "Single digit.";
    }
    return 0;
}
```

3 Way Output:

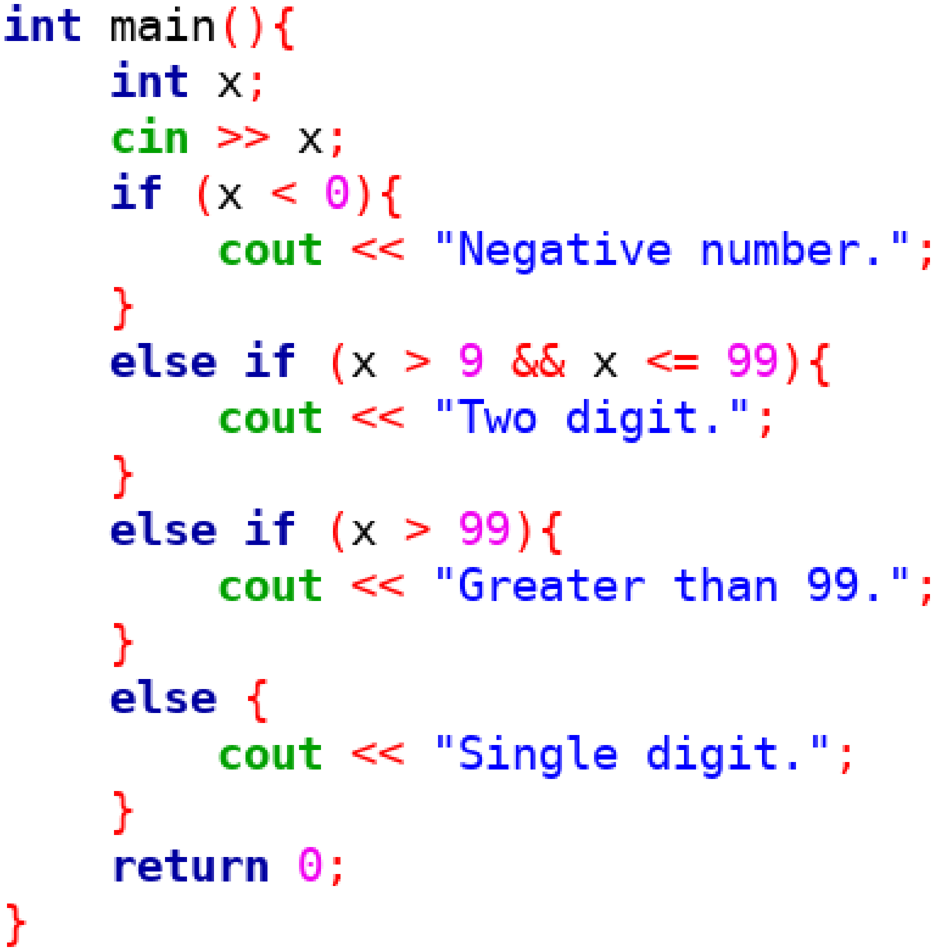


```
9
Single digit.
Process returned 0 (0x0)   execution time : 3.327 s
Press ENTER to continue.
```



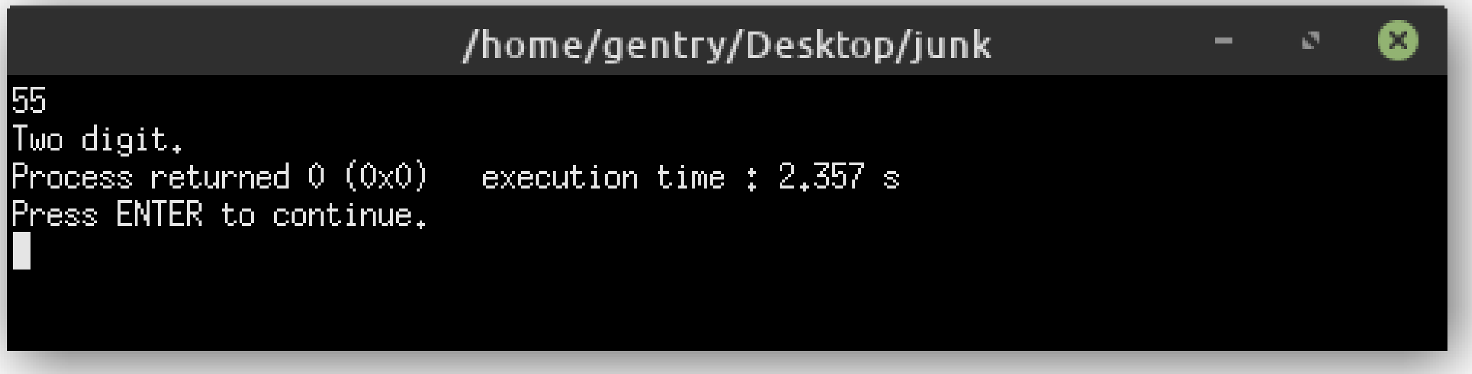
```
44
Greater than 9.
Process returned 0 (0x0)   execution time : 3.395 s
Press ENTER to continue.
```

4 Way Branching:

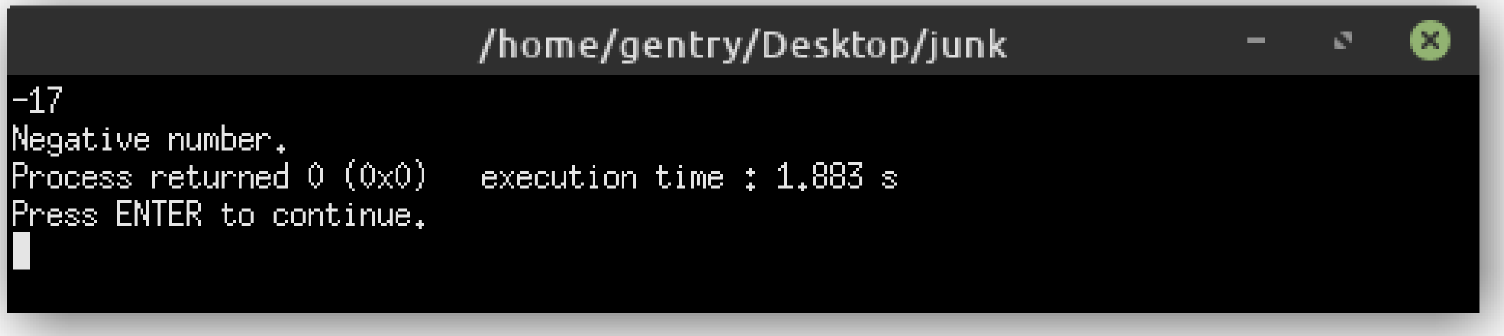


```
int main(){
    int x;
    cin >> x;
    if (x < 0){
        cout << "Negative number.";
    }
    else if (x > 9 && x <= 99){
        cout << "Two digit.";
    }
    else if (x > 99){
        cout << "Greater than 99.";
    }
    else {
        cout << "Single digit.";
    }
    return 0;
}
```

4 Way Output:



```
55
Two digit.
Process returned 0 (0x0)   execution time : 2.357 s
Press ENTER to continue.
```

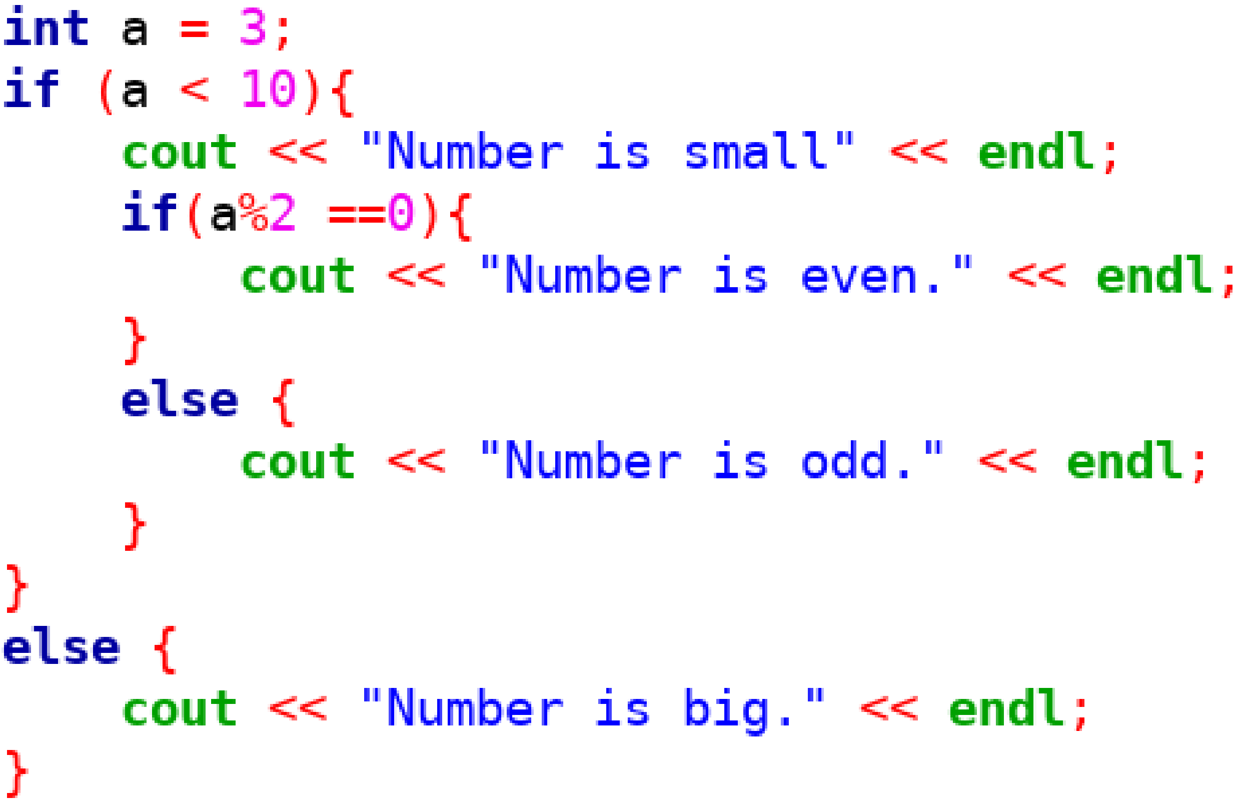


```
-17
Negative number.
Process returned 0 (0x0)   execution time : 1.883 s
Press ENTER to continue.
```

Nesting ifs:

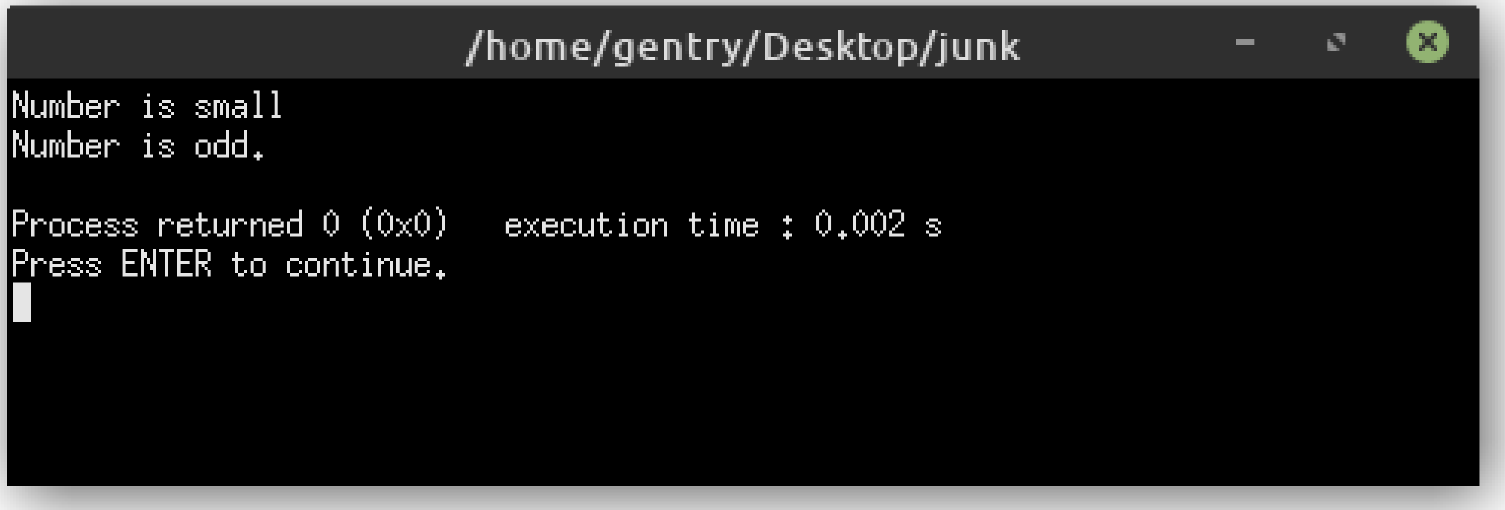
- **if** statements can be placed inside of other **if** statements.
- The inside **if** only has its condition checked if the outside **if** is **true**.
- Branches can be placed inside of **else if** blocks and **else** blocks just like **if**s
- Compound conditions (with **&&** or **||**) can often be replaced with nested **if**s

Nested ifs:



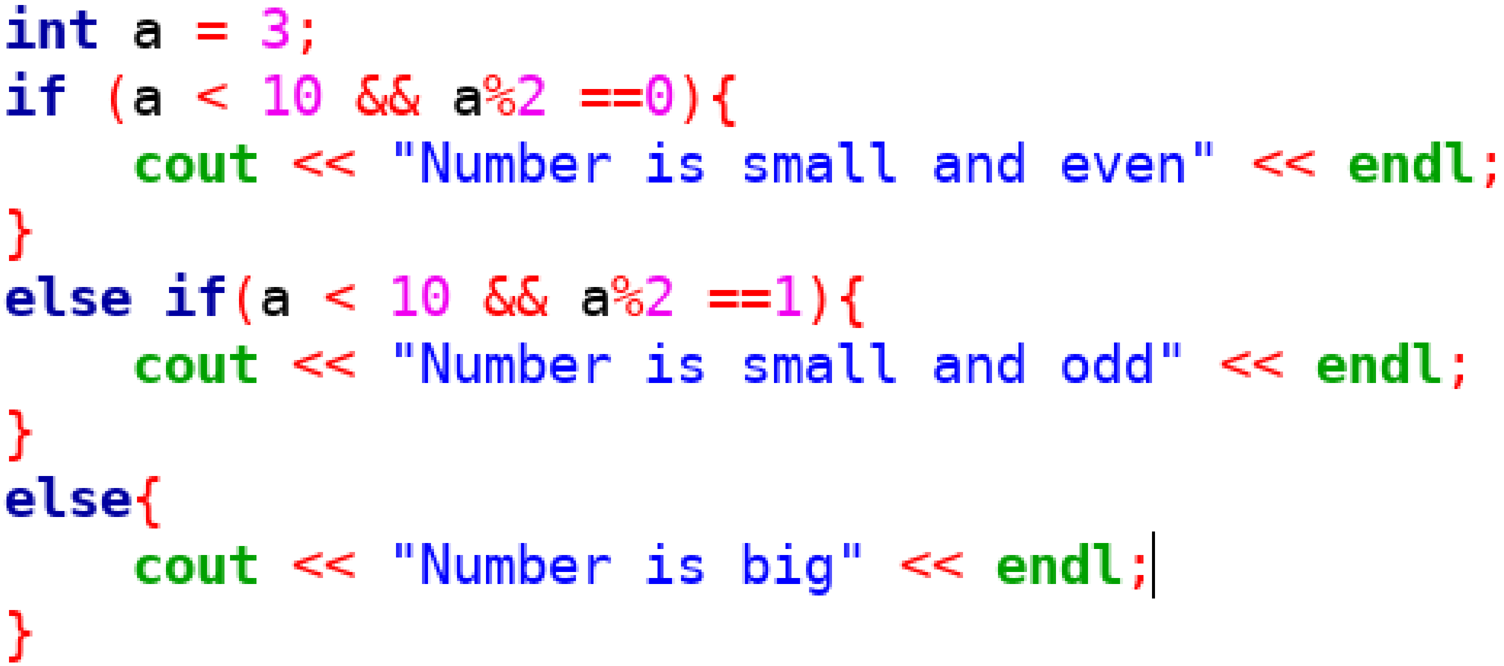
```
int a = 3;
if (a < 10){
    cout << "Number is small" << endl;
    if(a%2 ==0){
        cout << "Number is even." << endl;
    }
    else {
        cout << "Number is odd." << endl;
    }
}
else {
    cout << "Number is big." << endl;
}
```

Nested if output:



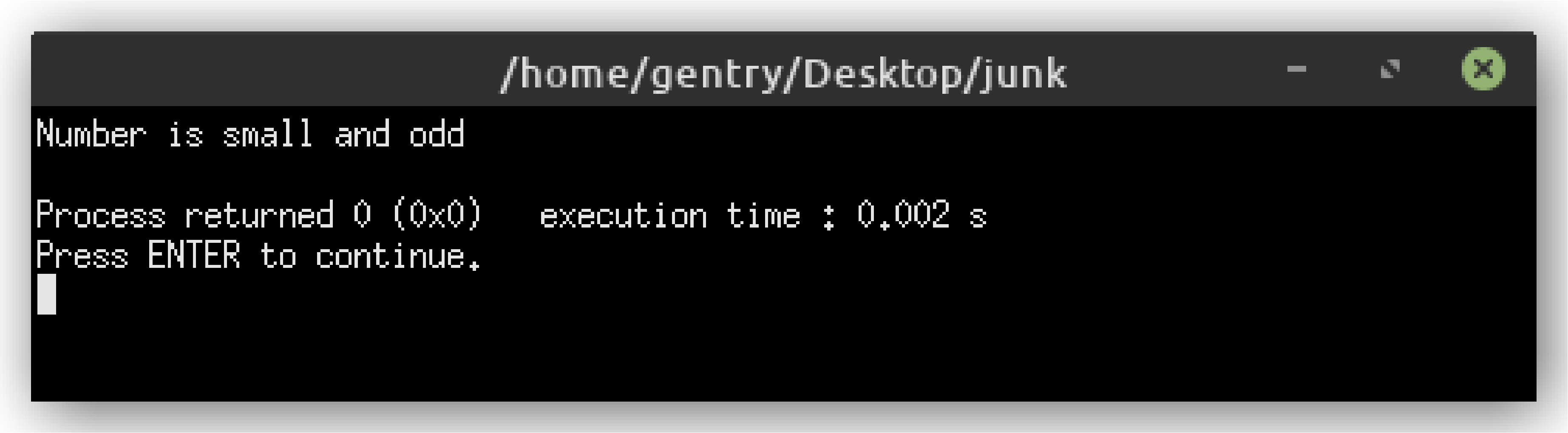
```
Number is small
Number is odd.

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Compound condition or nesting:



```
int a = 3;
if (a < 10 && a%2 ==0){
    cout << "Number is small and even" << endl;
}
else if(a < 10 && a%2 ==1){
    cout << "Number is small and odd" << endl;
}
else{
    cout << "Number is big" << endl;
}
```

Compound output:

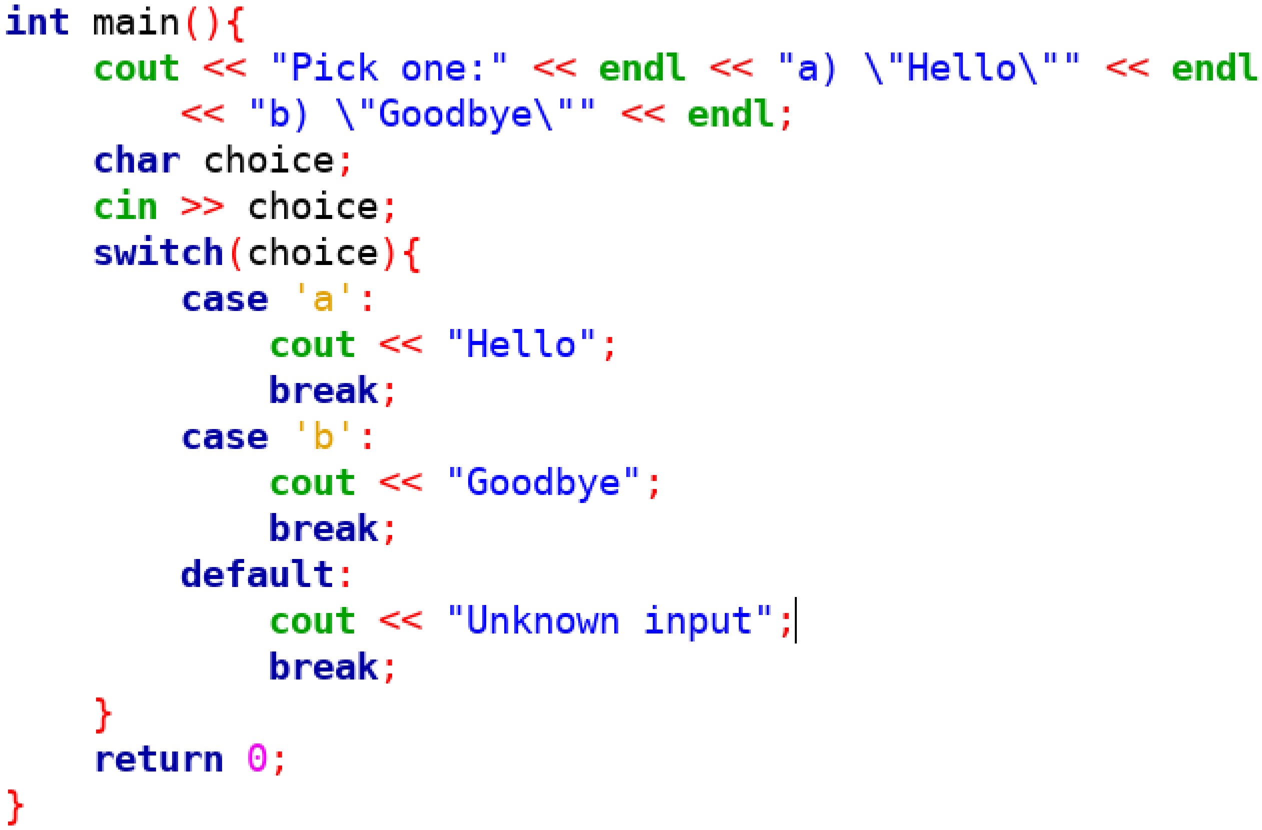


```
Number is small and odd

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Multi-way Branching with **switch**:

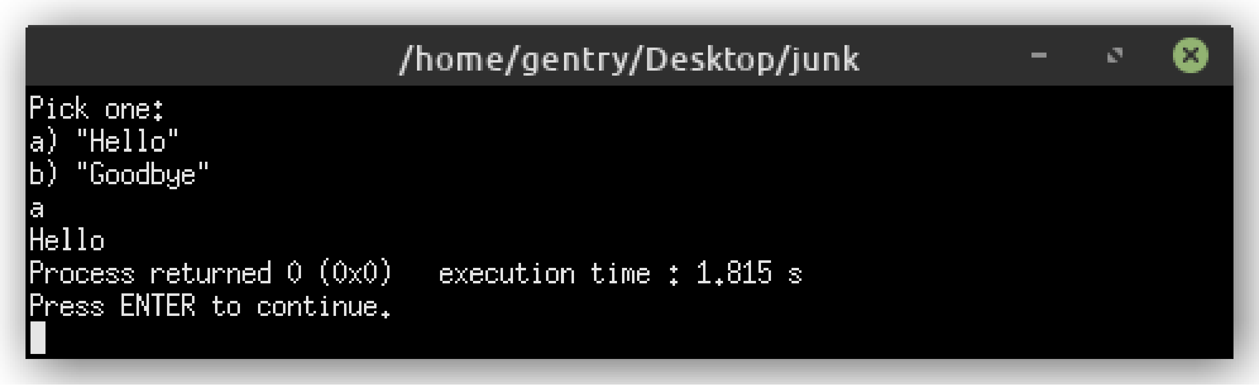
- When there are many cases that all depend on the same variable, **switch** statements can be used instead of **if...else if...else**
- Programs written to use **switch** sometimes has a faster run-time than the same program written using **if...else if...else**
- Every **switch** can be rewritten as an **if...else if...else**, but not the other way around.
- The keyword **break** marks the end of a case.
- The **else** becomes the **default** case of a switch statement.

Switch statements:

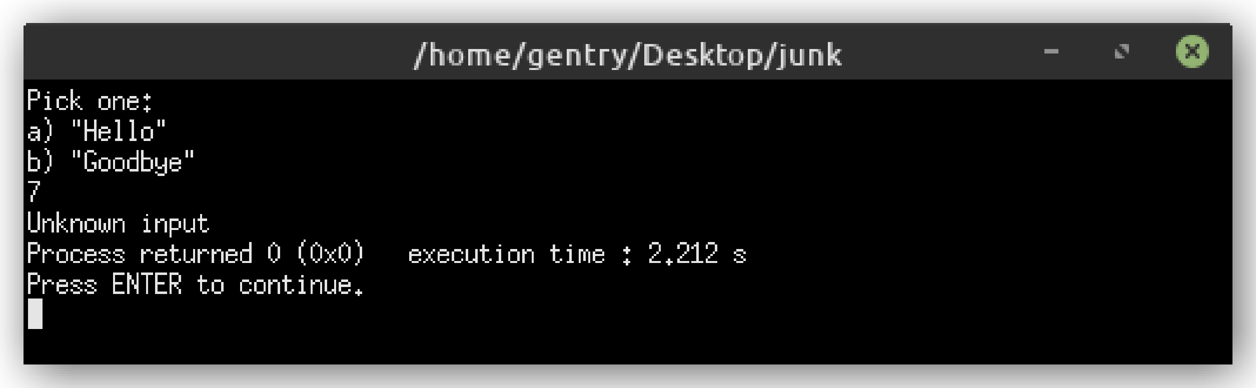


```
int main(){
    cout << "Pick one:" << endl << "a) \"Hello\"" << endl
        << "b) \"Goodbye\"" << endl;
    char choice;
    cin >> choice;
    switch(choice){
        case 'a':
            cout << "Hello";
            break;
        case 'b':
            cout << "Goodbye";
            break;
        default:
            cout << "Unknown input";
            break;
    }
    return 0;
}
```

Switch output:



```
Pick one:
a) "Hello"
b) "Goodbye"
a
Hello
Process returned 0 (0x0)   execution time : 1.815 s
Press ENTER to continue.
```



```
Pick one:
a) "Hello"
b) "Goodbye"
7
Unknown input
Process returned 0 (0x0)   execution time : 2.212 s
Press ENTER to continue.
```

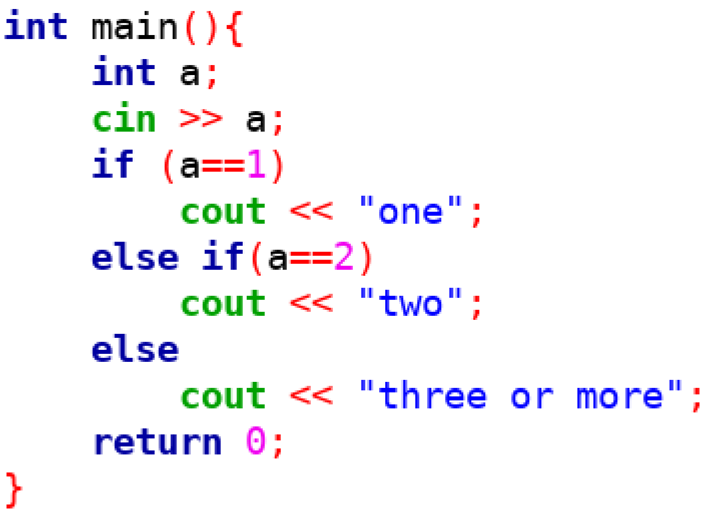
“Trapping” characters:

- “Quotation marks”, ‘single quotes’, and new lines are all special characters but we sometimes need to include them in string literals.
- A single backslash \ tells the compiler that the next character is part of a literal.
- Common trap characters:
  - \” a literal double quote
  - \’ a literal single quote
  - \n a new line
  - \t a tab
  - \\ a literal backslash

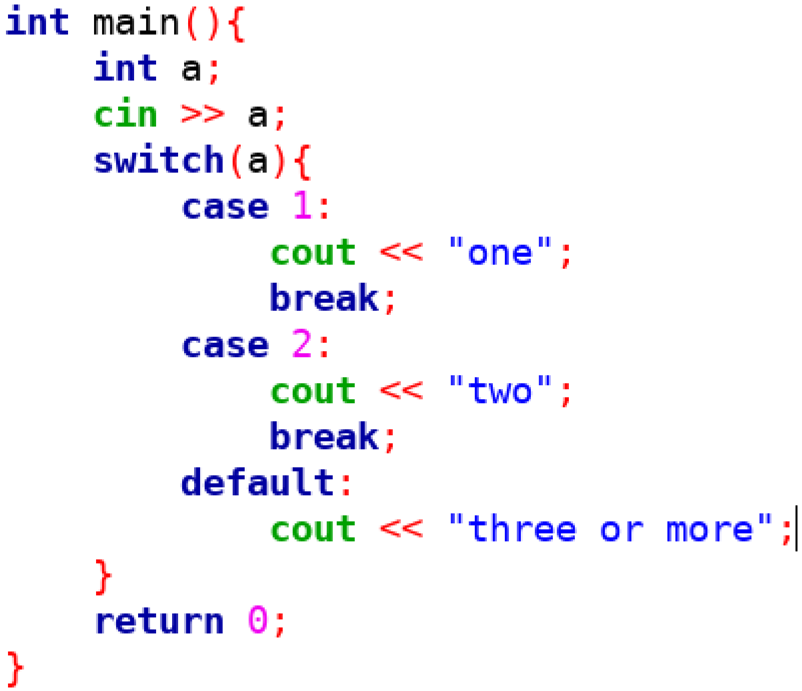
When to use switch:

- All conditions must depend on one variable
- All comparisons are equality (not greater than or less than)
- There is a “big” number of cases

if...else to switch:



```
int main(){
    int a;
    cin >> a;
    if (a==1)
        cout << "one";
    else if(a==2)
        cout << "two";
    else
        cout << "three or more";
    return 0;
}
```

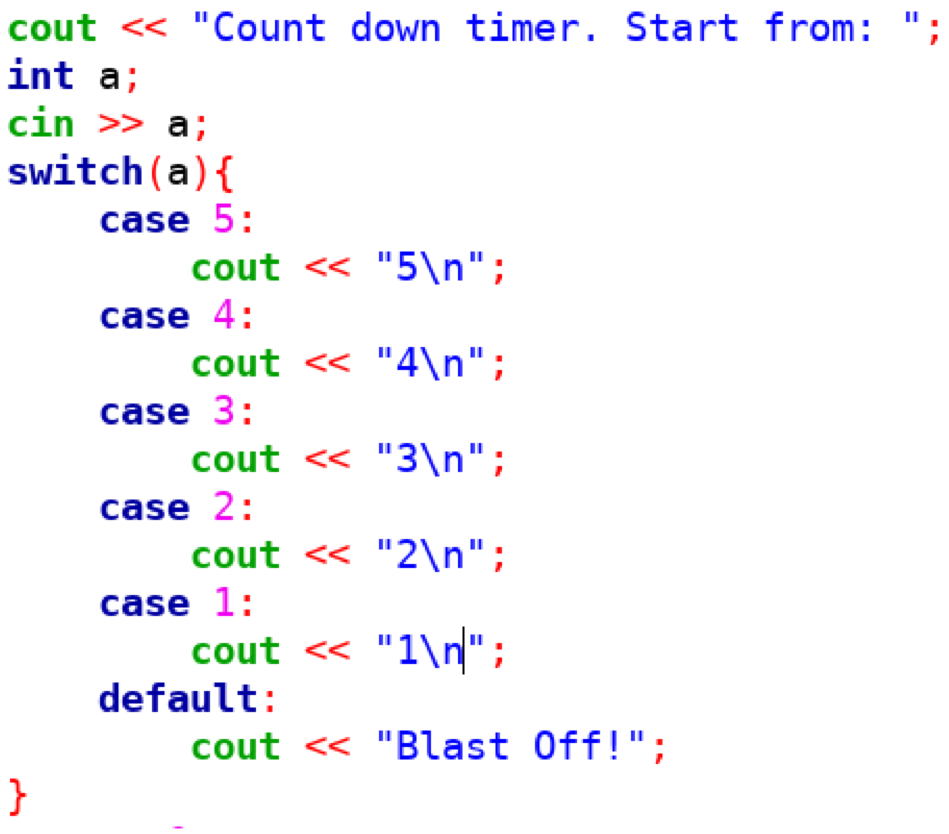


```
int main(){
    int a;
    cin >> a;
    switch(a){
        case 1:
            cout << "one";
            break;
        case 2:
            cout << "two";
            break;
        default:
            cout << "three or more";
    }
    return 0;
}
```

Switch statements can “fall through”:

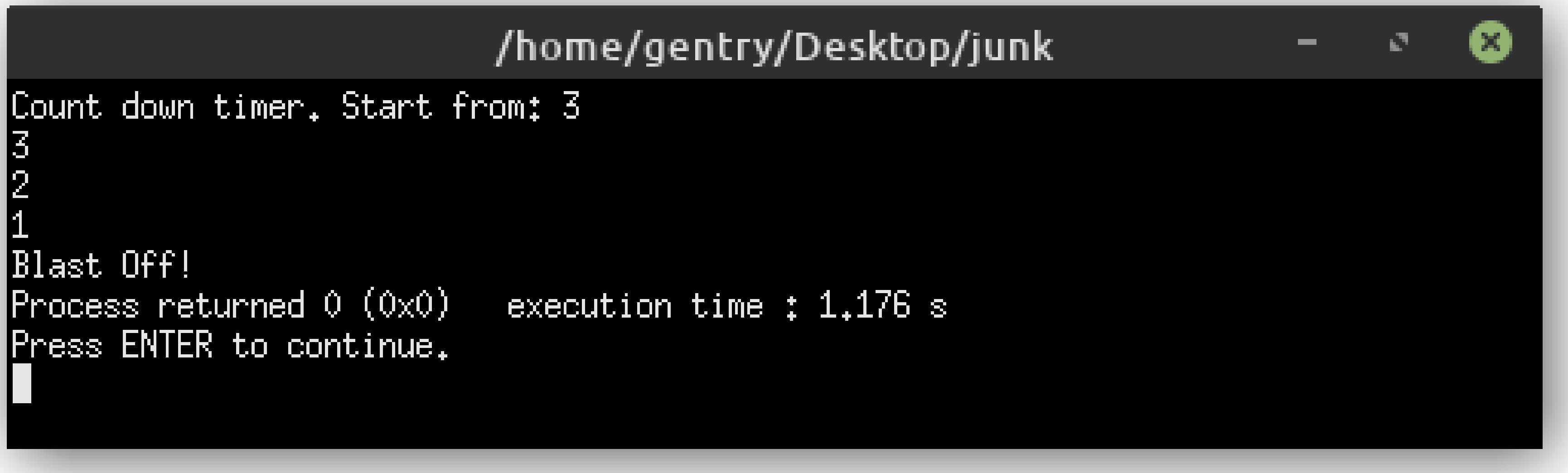
- Every **case** in a **switch** can be terminated with a **break**, but that’s not necessary.
- Without a **break** the program will “fall through” and continue to execute the next case.
- This behavior can be good or bad depending on whether it is done on purpose.

Fall through example:



```
cout << "Count down timer. Start from: ";
int a;
cin >> a;
switch(a){
    case 5:
        cout << "5\n";
    case 4:
        cout << "4\n";
    case 3:
        cout << "3\n";
    case 2:
        cout << "2\n";
    case 1:
        cout << "1\n";
    default:
        cout << "Blast Off!";
}
```

Fall through output:

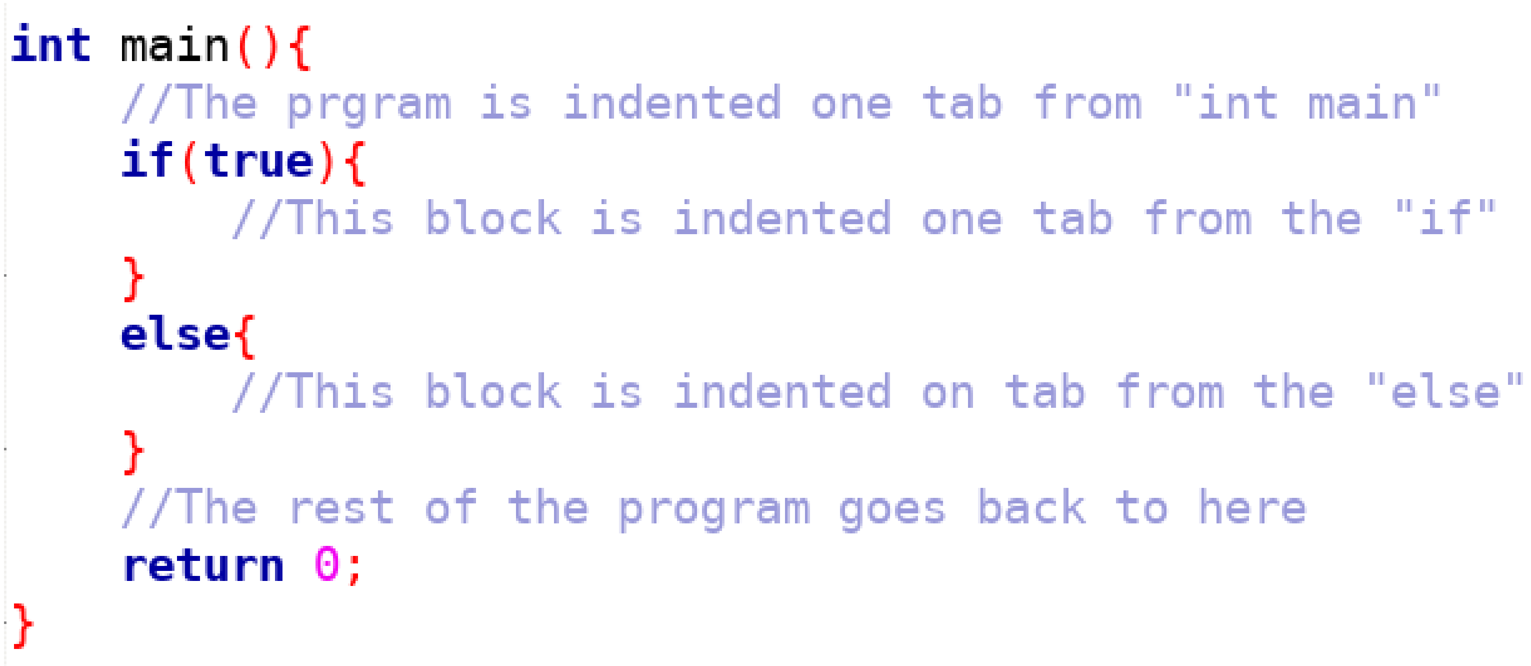


```
Count down timer. Start from: 3
3
2
1
Blast Off!
Process returned 0 (0x0)   execution time : 1.176 s
Press ENTER to continue.
```

Style Guide, Indentation:

- Code in C++ is written in “blocks”, which are often enclosed by {curly brackets}.
- Indenting each block by one tab or 4 spaces from its parent statement makes code that is easier for humans to read.
- Extra whitespace does not affect the execution of C++ programs, so indenting will not affect the compiler or program.

Indentation:



```
int main(){
    //The prgram is indented one tab from "int main"
    if(true){
        //This block is indented one tab from the "if"
    }
    else{
        //This block is indented on tab from the "else"
    }
    //The rest of the program goes back to here
    return 0;
}
```