§3 Flow of Control

ENGG1111

Computer Programming and Applications

Dirk Schnieders

Outline

- Branching
  - if-else
  - switch
- Looping
  - while
  - for

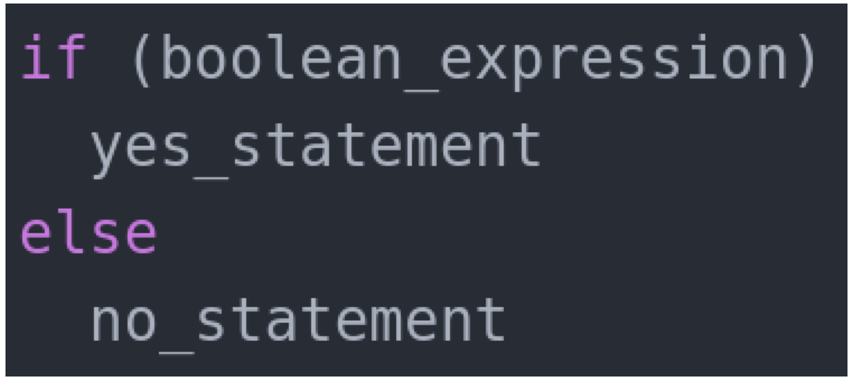
Flow of Control

- Recall that statements are executed sequentially
- In more complex programs, however, it is often necessary to alter the order in which statements are executed, e.g.,
  - Branching: Choose between alternative actions
  - Looping: Repeat an action a number of times
- The order in which statements are executed is often referred to as flow of control

Branching

if-else

- In C++, a simple branching action can be achieved using an if-else statement



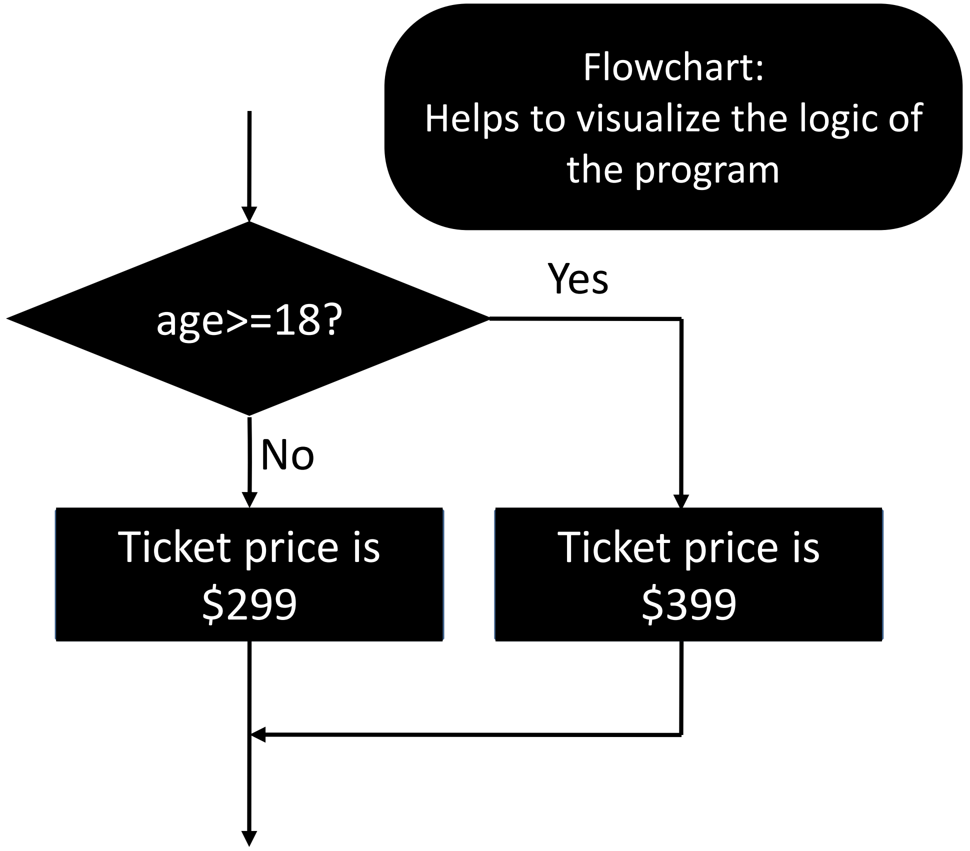
```
if (boolean_expression)
  yes_statement
else
  no_statement
```

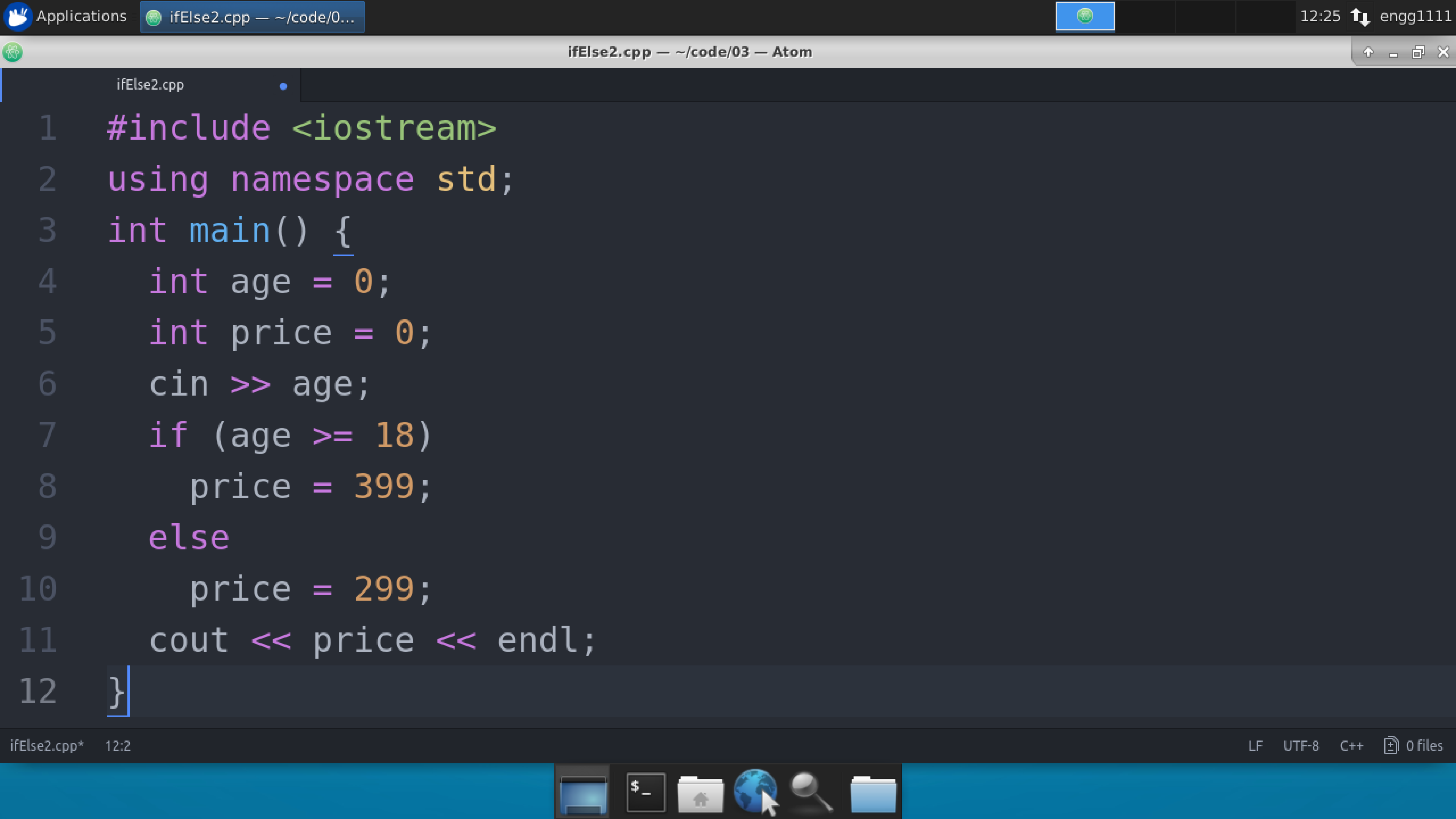
- When an if-else statement is executed, the
- boolean_expression will be evaluated
  - If it is true, yes_statement will be executed
  - If it is false, no_statement will be executed

Example

- Disneyland charges $399 for an adult ticket (age 18 or above), and $299 for a child ticket

HONG KONG DISNEYLAND
HONG KONG DISNEYLAND
FOR TERMS AND CONDITIONS, PLEASE REFER TO TICKET BACK 請參閱門票背後之條款及細則



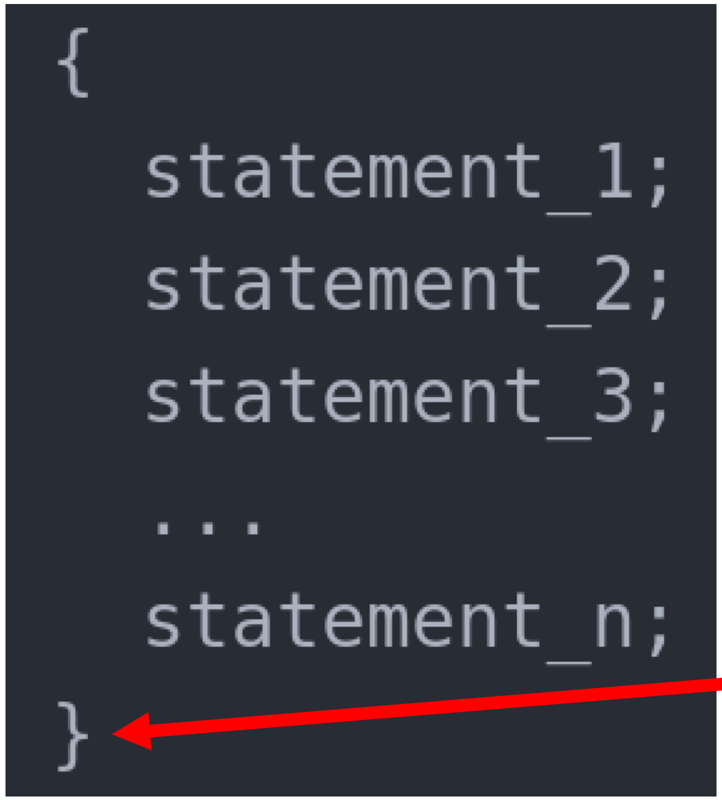


```cpp
#include <iostream>
using namespace std;
int main() {
  int age = 0;
  int price = 0;
  cin >> age;
  if (age >= 18)
    price = 399;
  else
    price = 299;
  cout << price << endl;
}
```

Compound Statements

- It is possible to execute more than one statement in each branch of an if-else statement by using compound statements
- A compound statement is simply a list of statements enclosed within a pair of braces {}
- A compound statement is treated as a single statement by C++, and may be used in any places where a single statement is expected

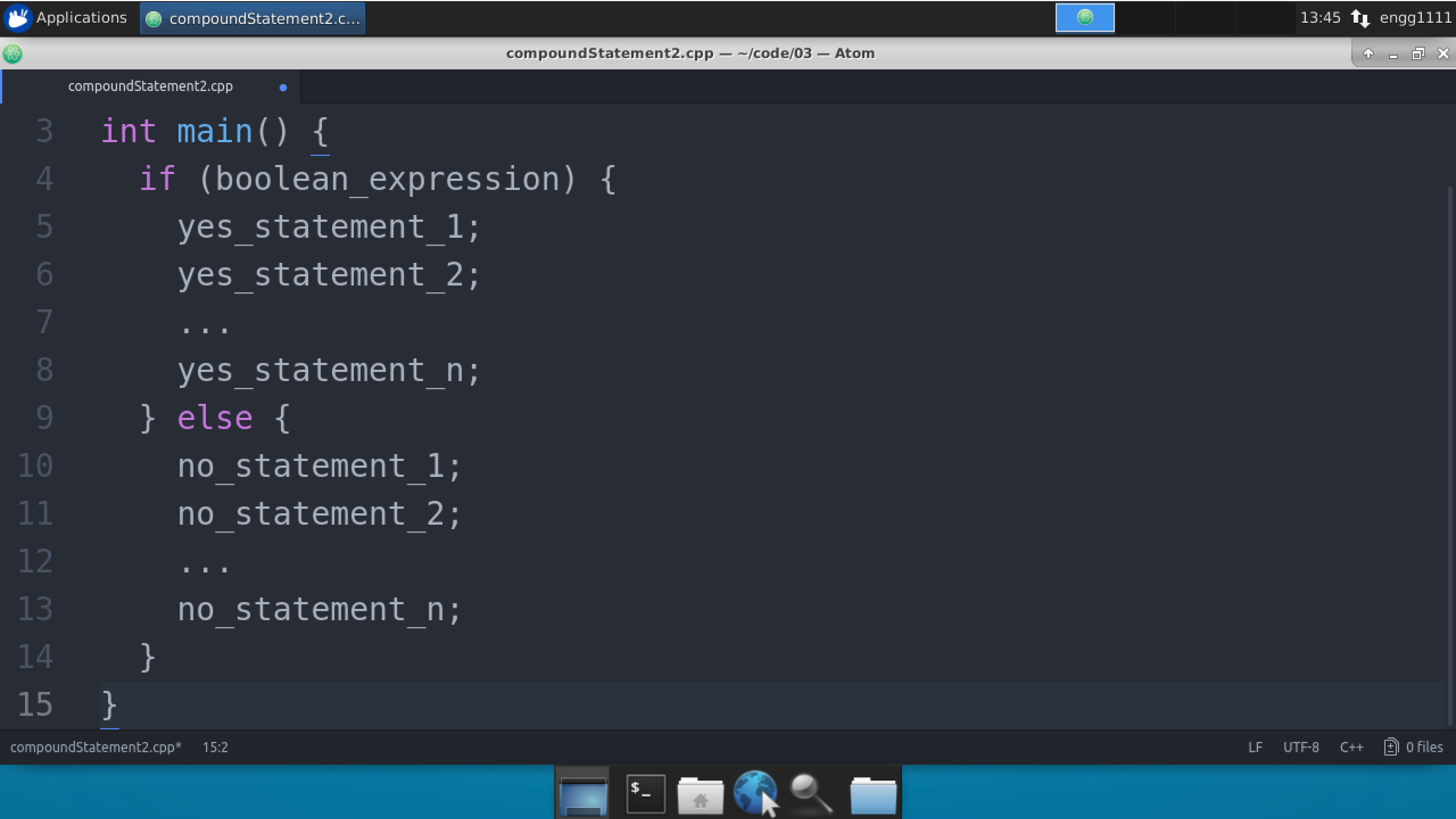
Compound Statements



```
{
    statement_1;
    statement_2;
    statement_3;
    ...
    statement_n;
}
```

Statements within a compound statement are executed sequentially

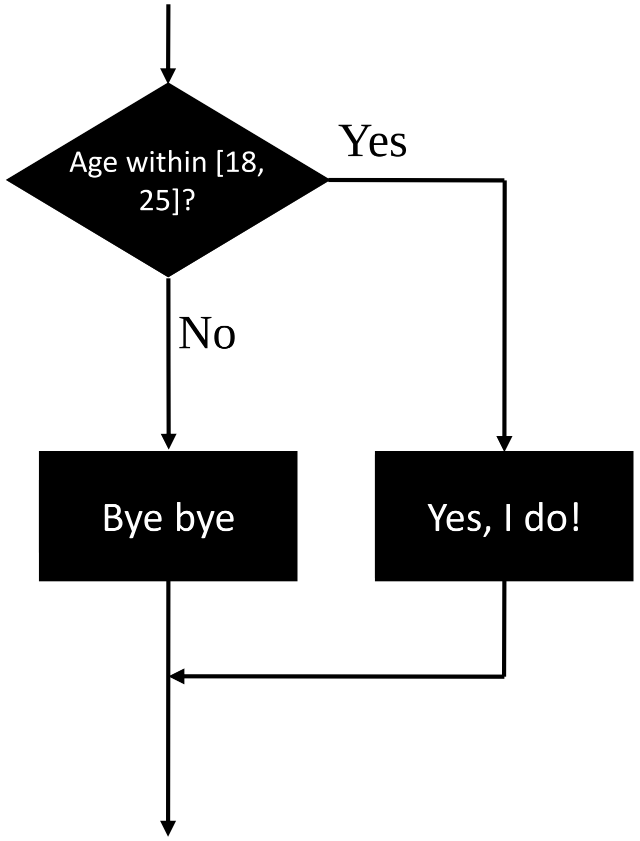
No
;
after compound statement !

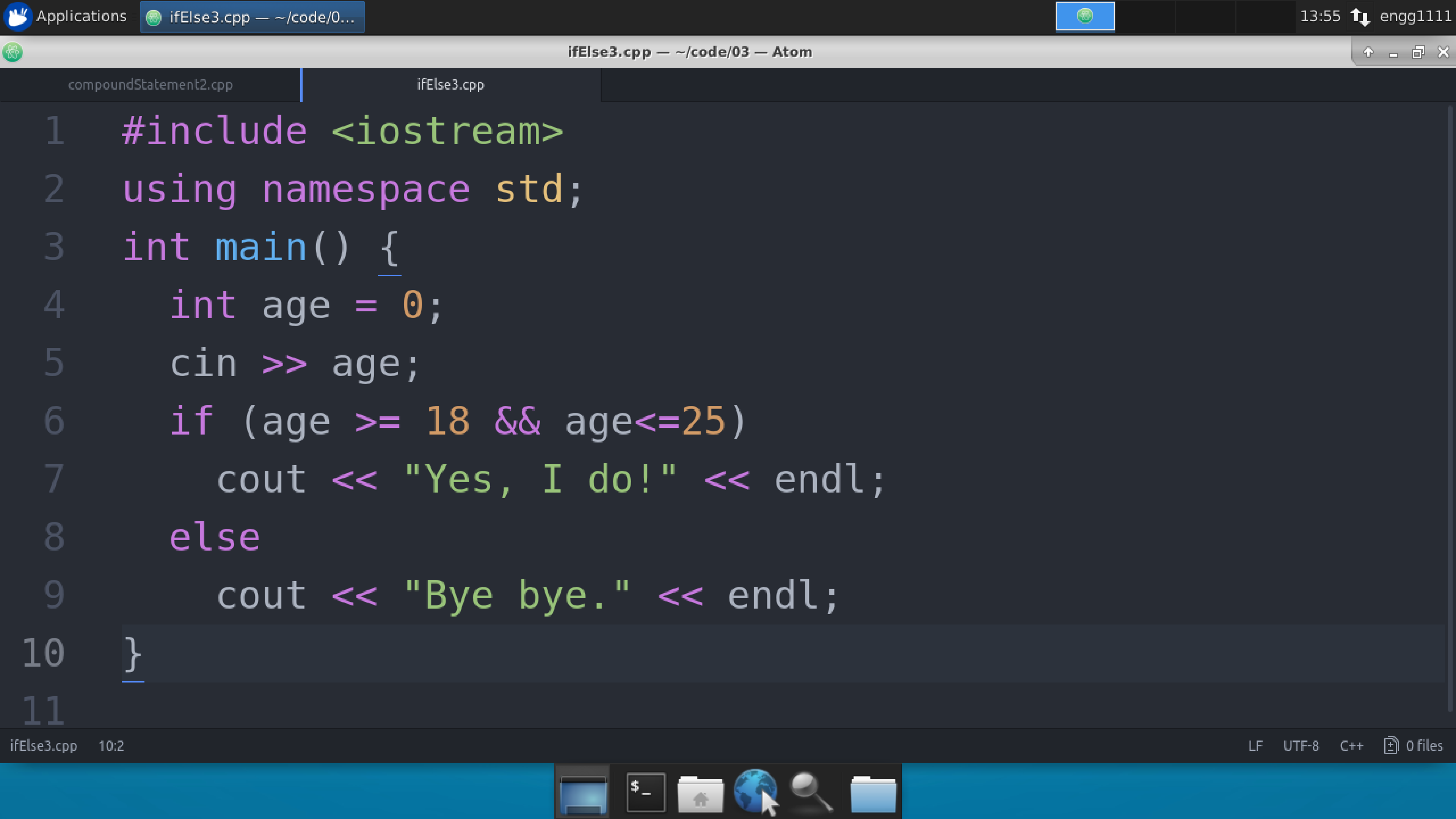


```cpp
int main() {
  if (boolean_expression) {
    yes_statement_1;
    yes_statement_2;
    ...
    yes_statement_n;
  } else {
    no_statement_1;
    no_statement_2;
    ...
    no_statement_n;
  }
}
```

Example

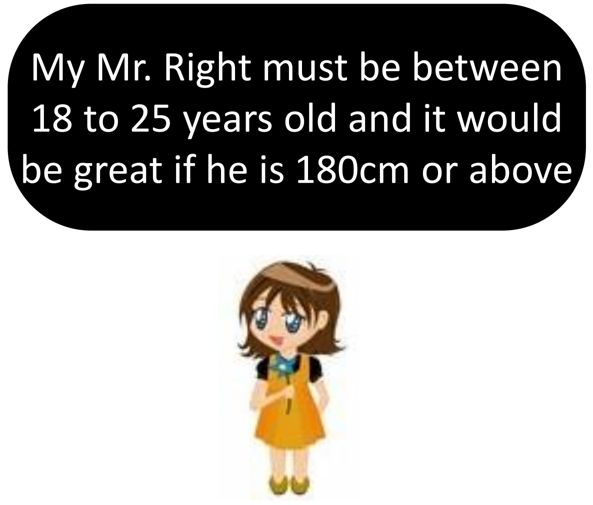
My Mr. Right must be between 18 to 25 years old

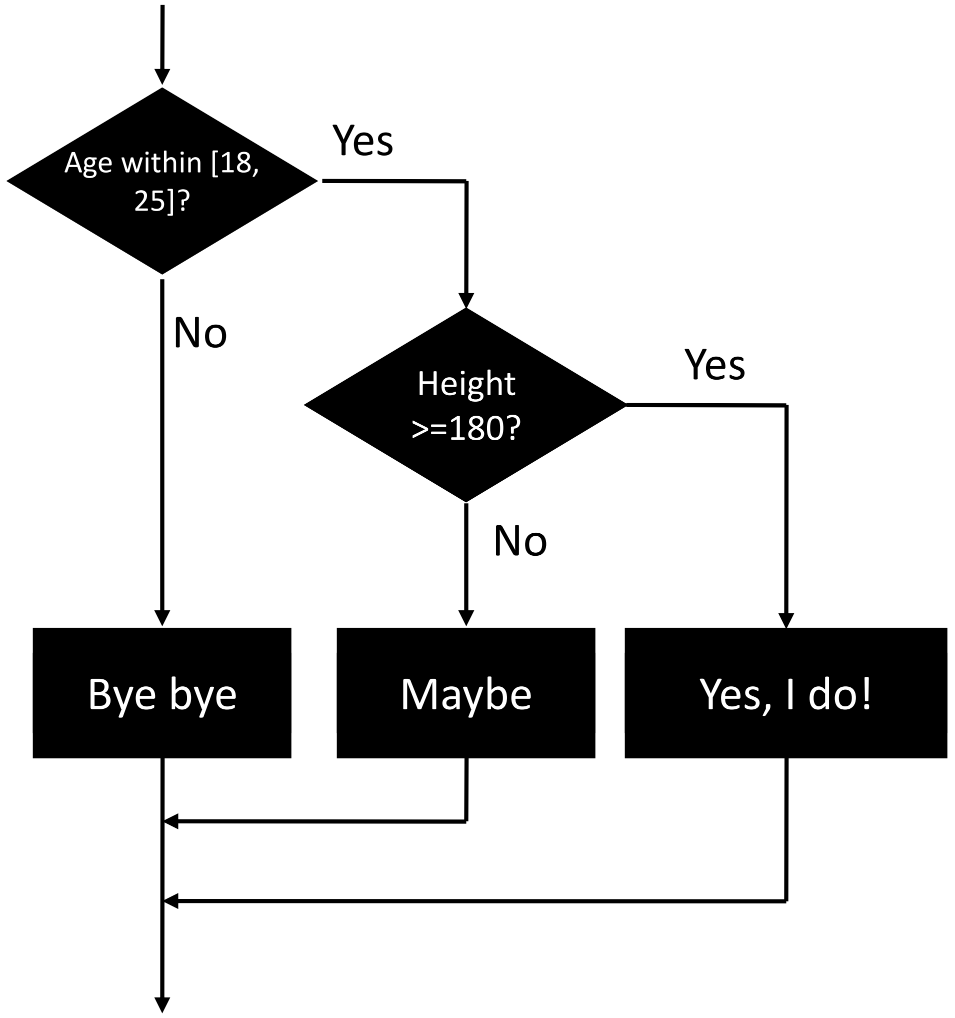


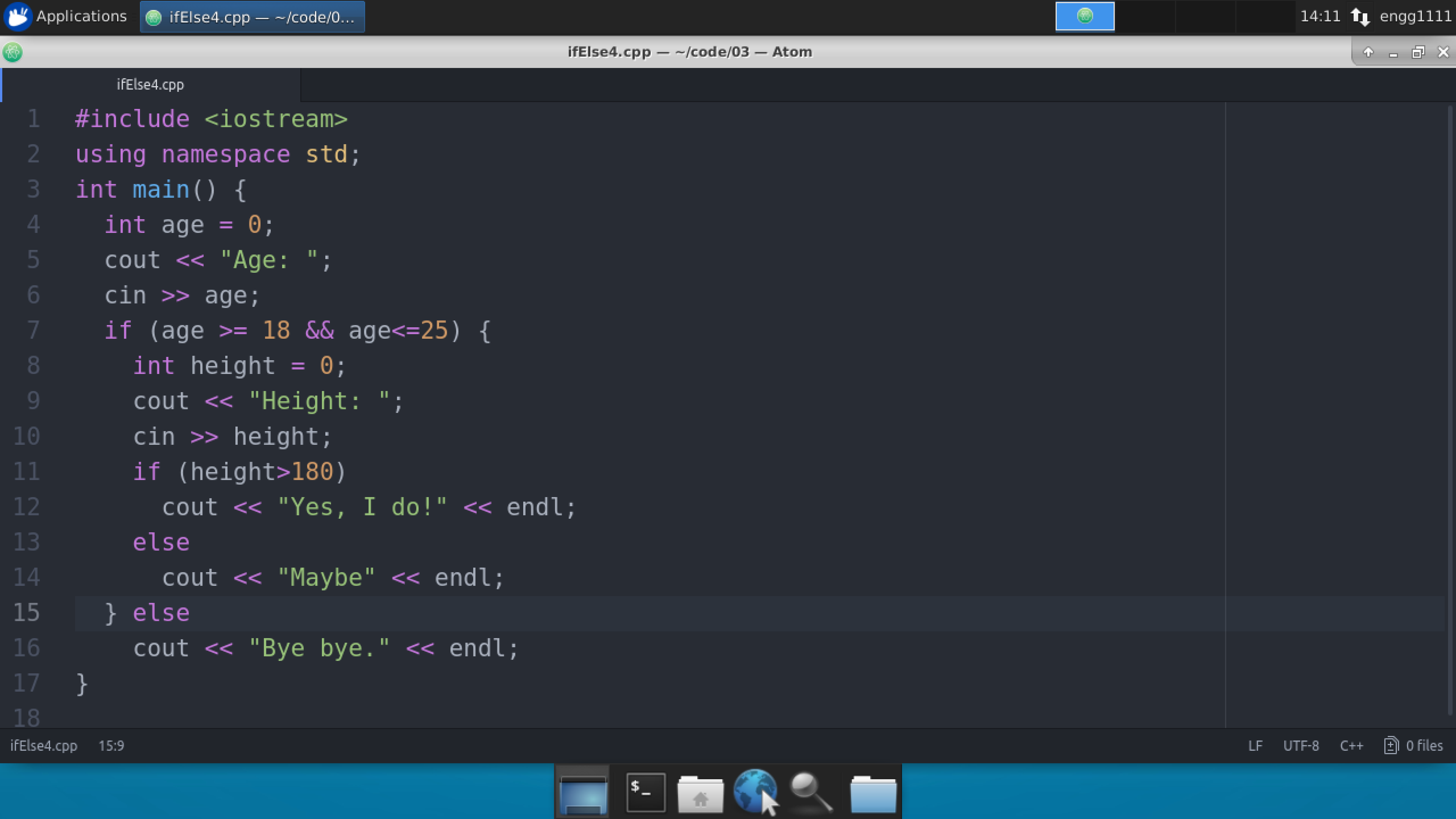


```cpp
#include <iostream>
using namespace std;
int main() {
  int age = 0;
  cin >> age;
  if (age >= 18 && age<=25)
    cout << "Yes, I do!" << endl;
  else
    cout << "Bye bye." << endl;
}
```

Example


My Mr. Right must be between 18 to 25 years old and it would be great if he is 180cm or above

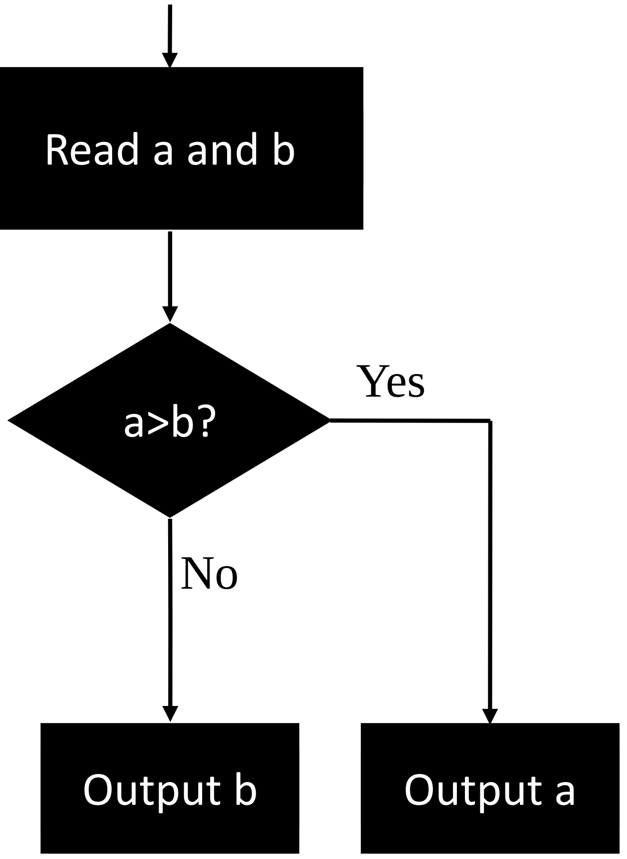

Age within [18, 25]?
Yes
No
Height >=180?
Yes
No
Bye bye
Maybe
Yes, I do!

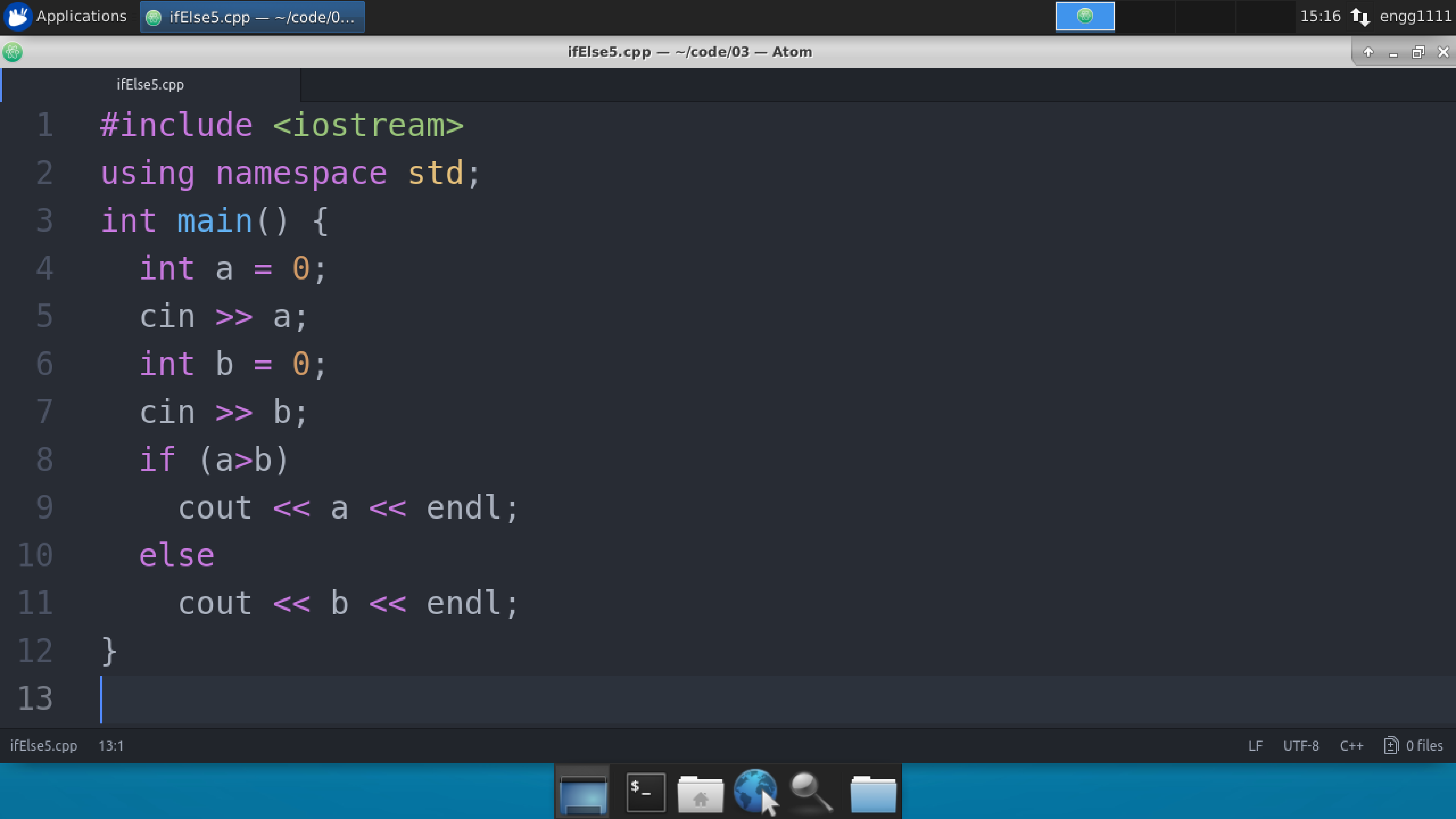


```cpp
#include <iostream>
using namespace std;
int main() {
  int age = 0;
  cout << "Age: ";
  cin >> age;
  if (age >= 18 && age<=25) {
    int height = 0;
    cout << "Height: ";
    cin >> height;
    if (height>180)
      cout << "Yes, I do!" << endl;
    else
      cout << "Maybe" << endl;
  } else
    cout << "Bye bye." << endl;
}
```

Example

Write a program that reads in two integers and outputs the maximum

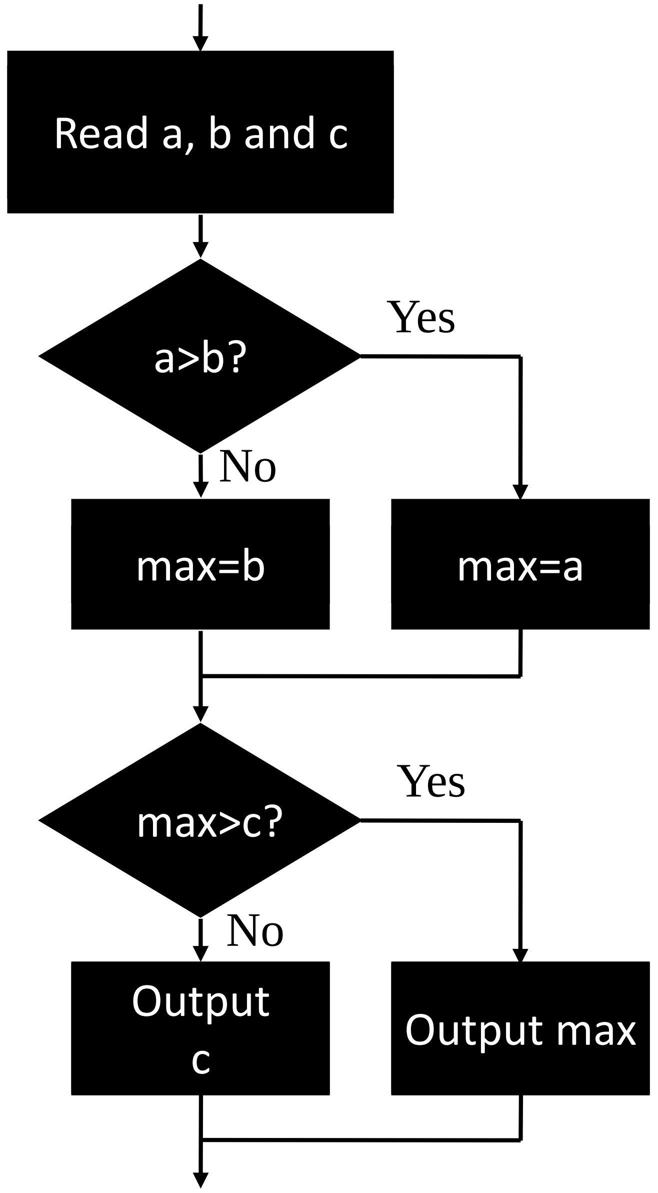


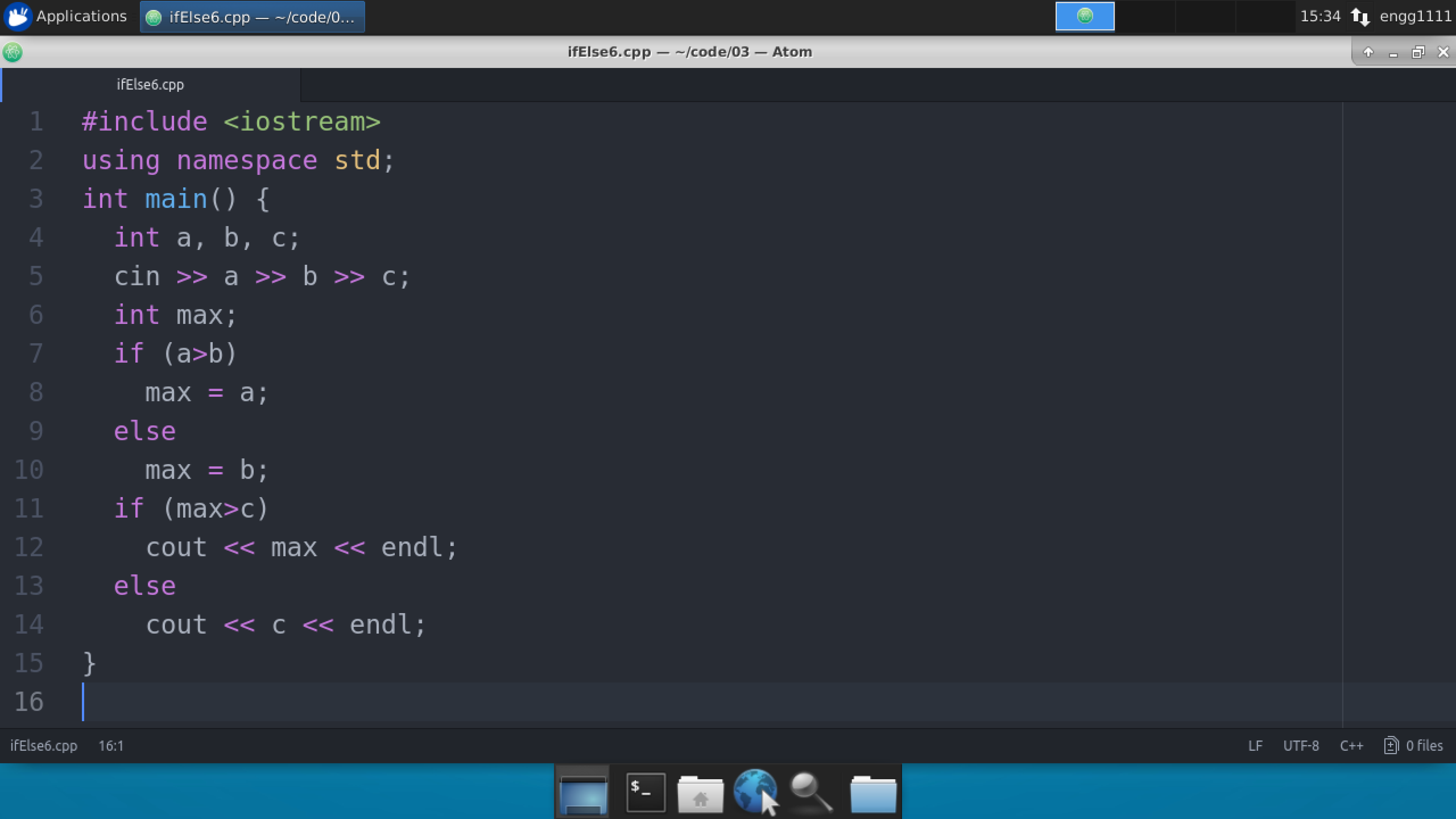


```cpp
#include <iostream>
using namespace std;
int main() {
  int a = 0;
  cin >> a;
  int b = 0;
  cin >> b;
  if (a>b)
    cout << a << endl;
  else
    cout << b << endl;
}
```

Example

Write a program that reads in three integers and outputs the maximum

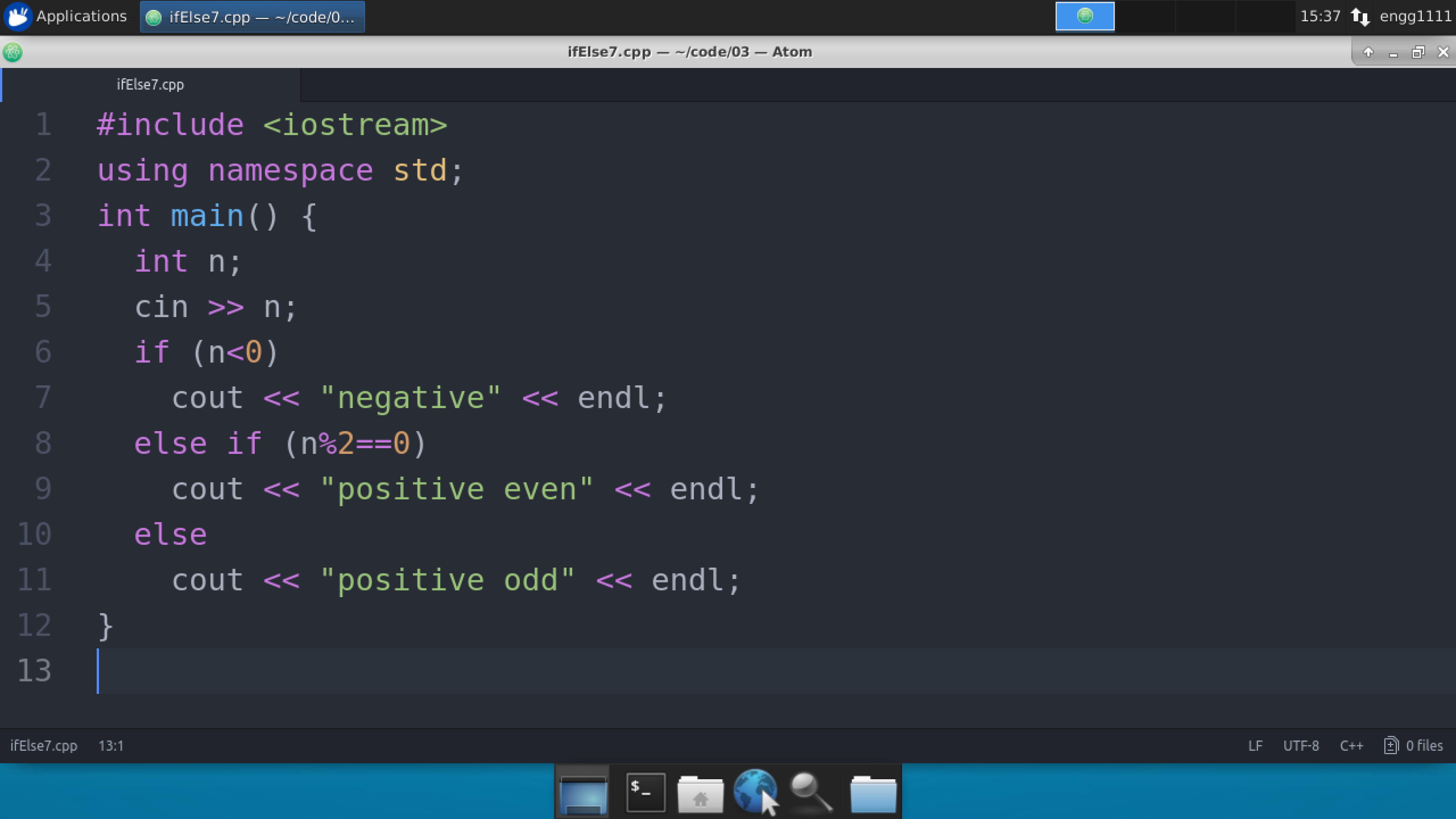




```cpp
#include <iostream>
using namespace std;
int main() {
  int a, b, c;
  cin >> a >> b >> c;
  int max;
  if (a>b)
    max = a;
  else
    max = b;
  if (max>c)
    cout << max << endl;
  else
    cout << c << endl;
}

```

Multi-way if-else Statement

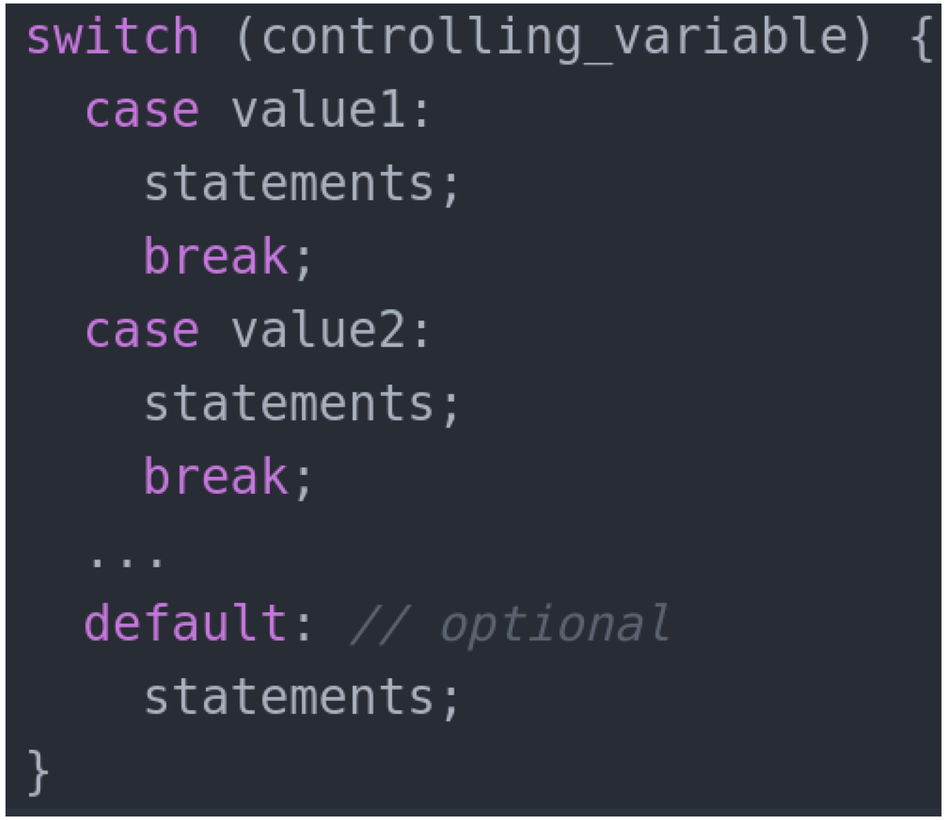
- In a multi-way if-else statement
- The expressions are checked in order until the first true expression is encountered, and then the corresponding statement is executed
- If none of the Boolean expressions is true, then the else statement is executed



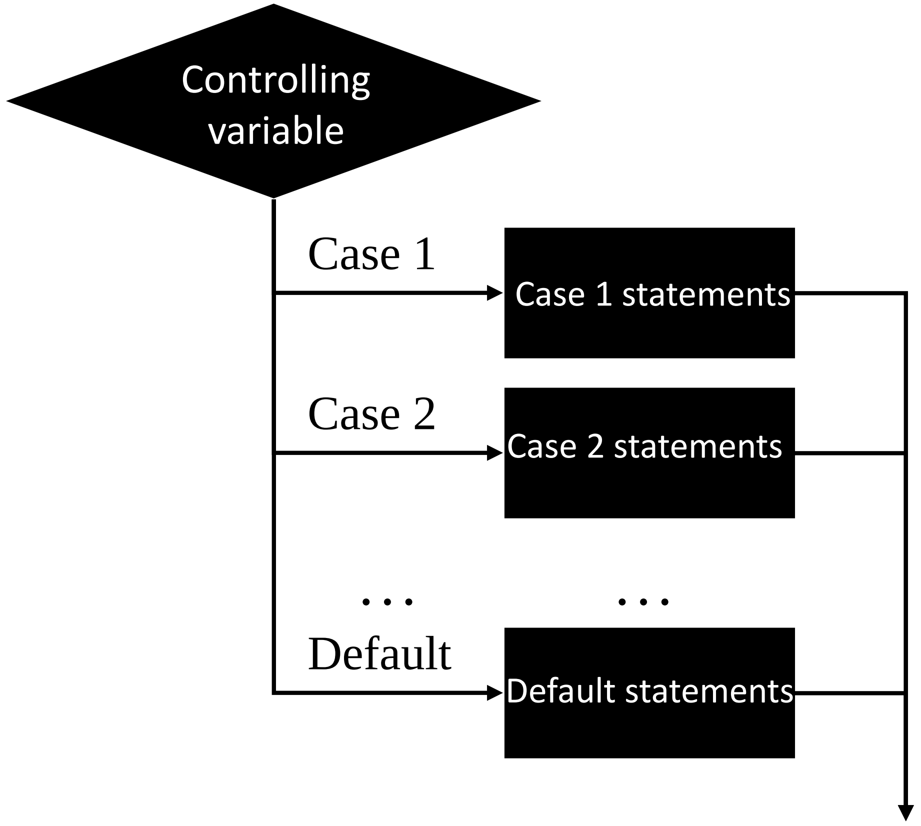
```cpp
#include <iostream>
using namespace std;
int main() {
  int n;
  cin >> n;
  if (n<0)
    cout << "negative" << endl;
  else if (n%2==0)
    cout << "positive even" << endl;
  else
    cout << "positive odd" << endl;
}

```

switch Statement

- A multi-way branching action can also be achieved using a switch statement

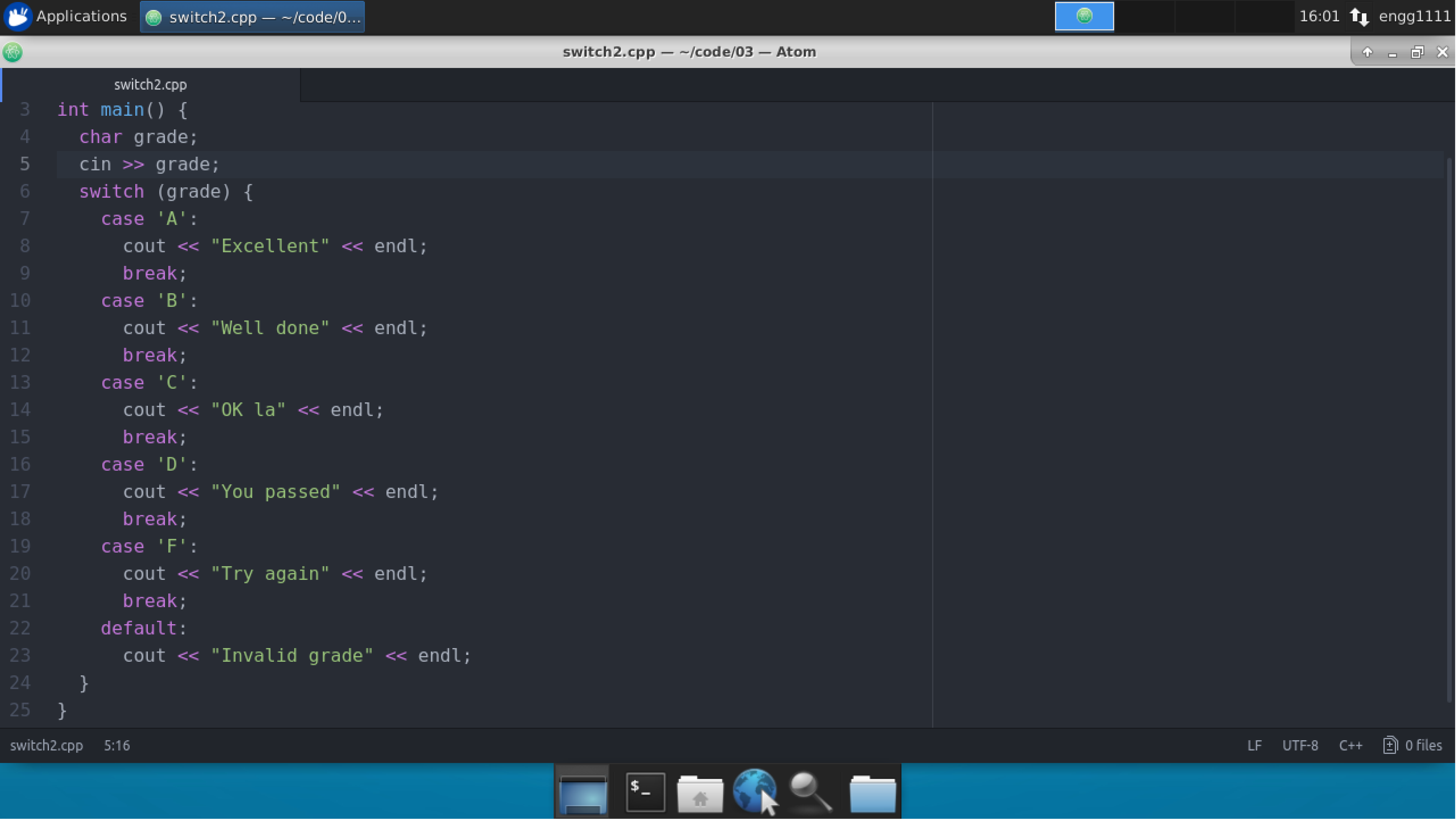


```
switch (controlling_variable) {
  case value1:
    statements;
    break;
  case value2:
    statements;
    break;
  ...
  default: // optional
    statements;
}
```



switch Statement

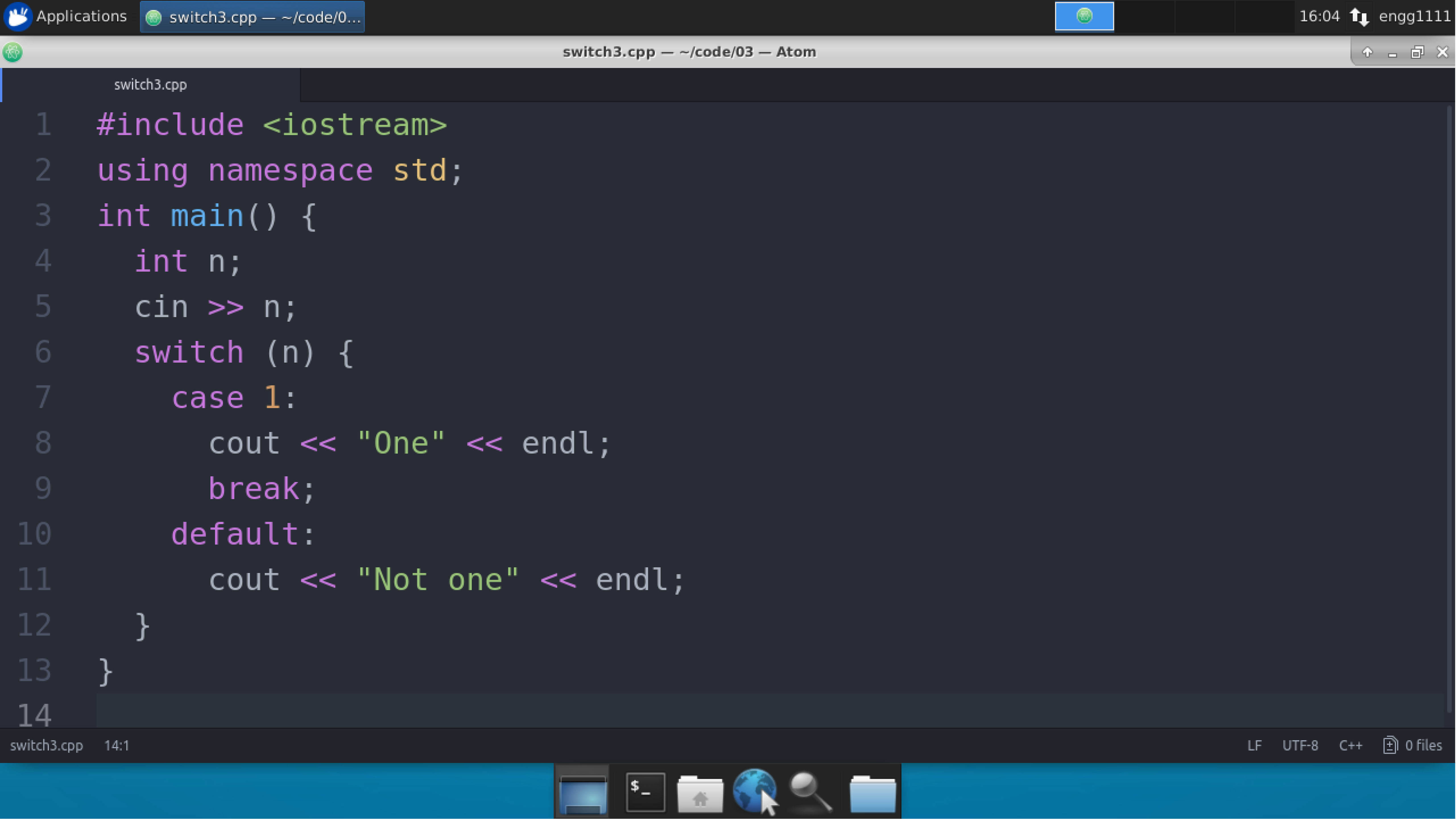
- The value given after the case keywords are checked in order until the first that equals the value of the controlling variable is found, and then the following statement(s) are executed
- If none of the constants matches the value of the controlling variable, then the default statement(s) are executed



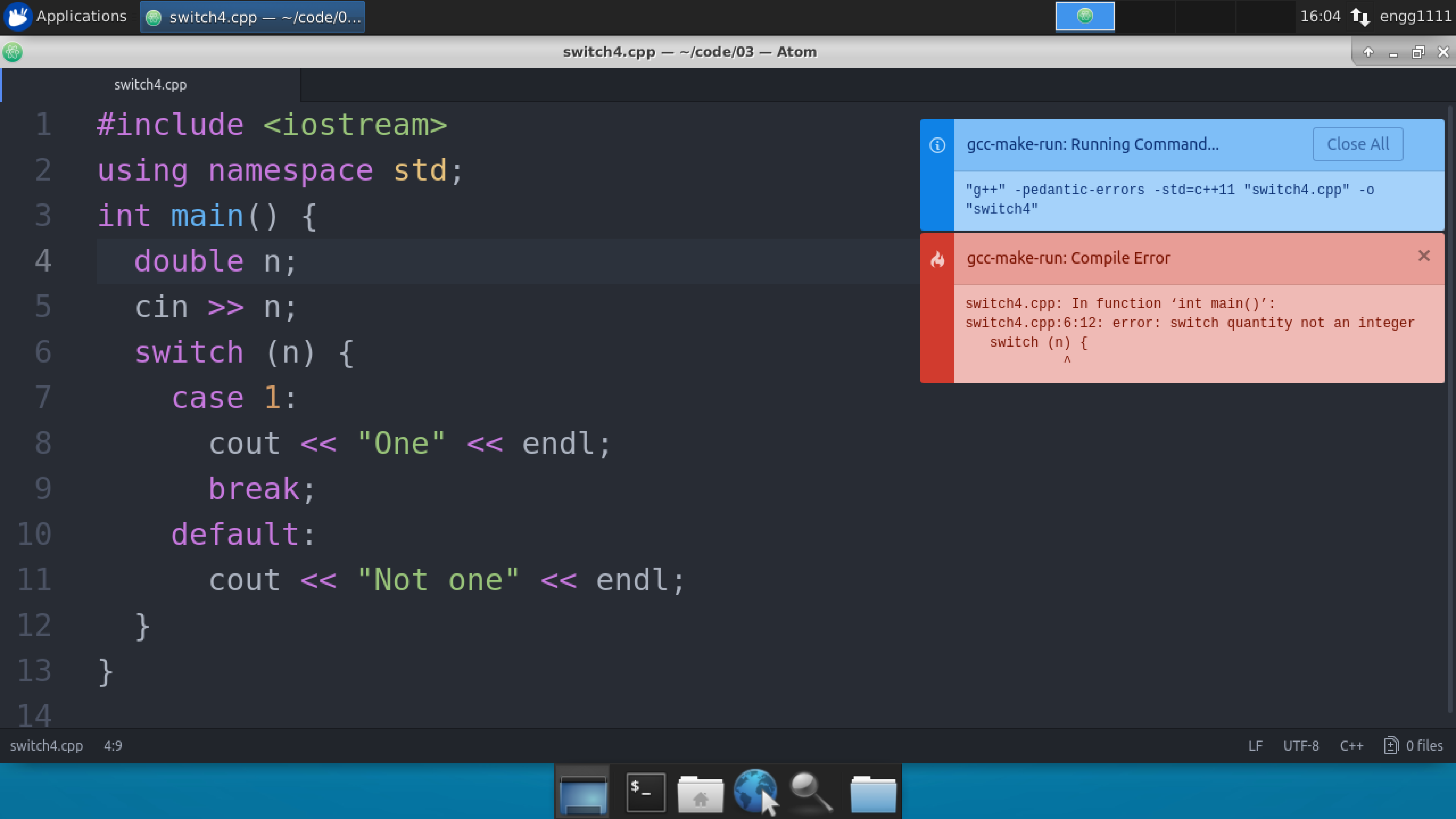
```cpp
int main() {
  char grade;
  cin >> grade;
  switch (grade) {
    case 'A':
      cout << "Excellent" << endl;
      break;
    case 'B':
      cout << "Well done" << endl;
      break;
    case 'C':
      cout << "OK la" << endl;
      break;
    case 'D':
      cout << "You passed" << endl;
      break;
    case 'F':
      cout << "Try again" << endl;
      break;
    default:
      cout << "Invalid grade" << endl;
  }
}
```

switch Statement

- The controlling variable in a switch statement must be an integer (int), boolean (bool), or a character (char)



```cpp
#include <iostream>
using namespace std;
int main() {
  int n;
  cin >> n;
  switch (n) {
    case 1:
      cout << "One" << endl;
      break;
    default:
      cout << "Not one" << endl;
  }
}

```



```cpp
#include <iostream>
using namespace std;
int main() {
  double n;
  cin >> n;
  switch (n) {
    case 1:
      cout << "One" << endl;
      break;
    default:
      cout << "Not one" << endl;
  }
}
```

gcc-make-run: Running Command...

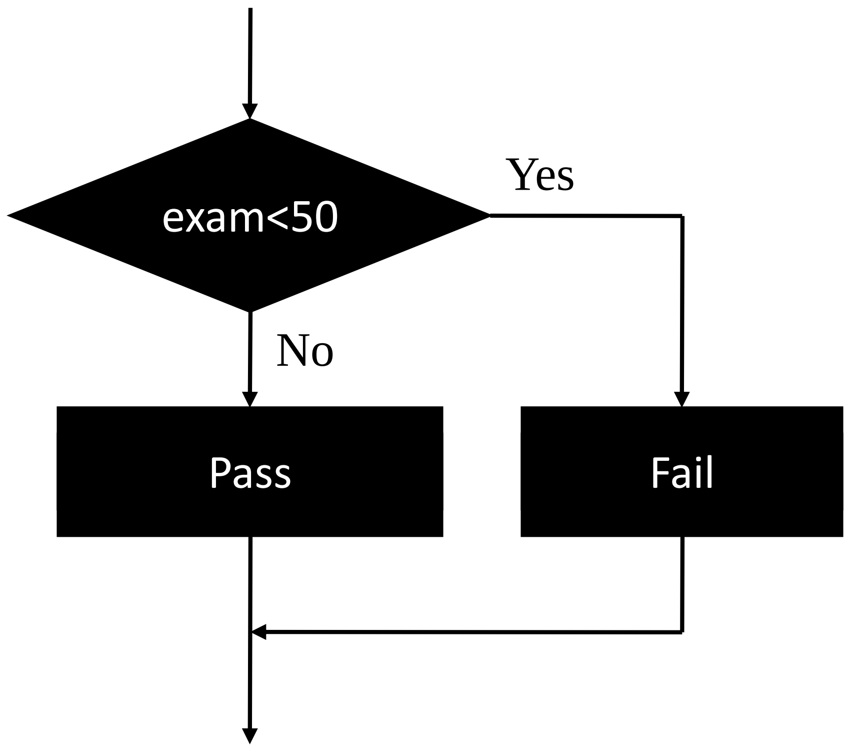
```
"g++" -pedantic-errors -std=c++11 "switch4.cpp" -o
"switch4"
```

gcc-make-run: Compile Error

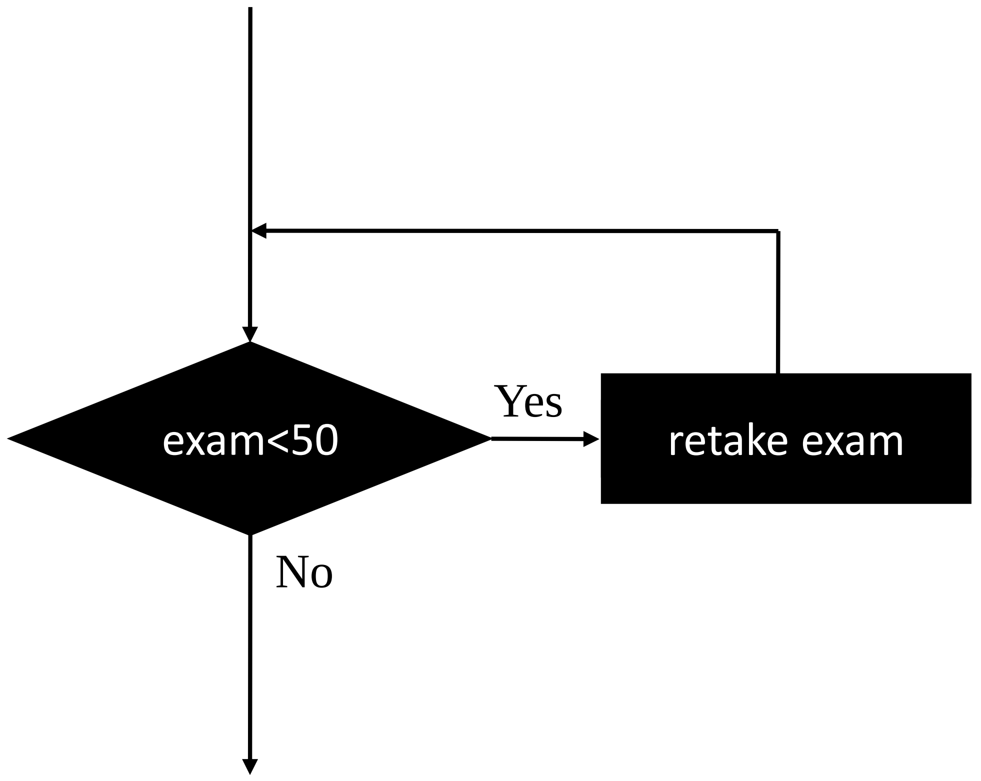
```
switch4.cpp: In function ‘int main()’:
switch4.cpp:6:12: error: switch quantity not an integer
   switch (n) {
            ^
```

Looping

Branching vs. Looping



Branching



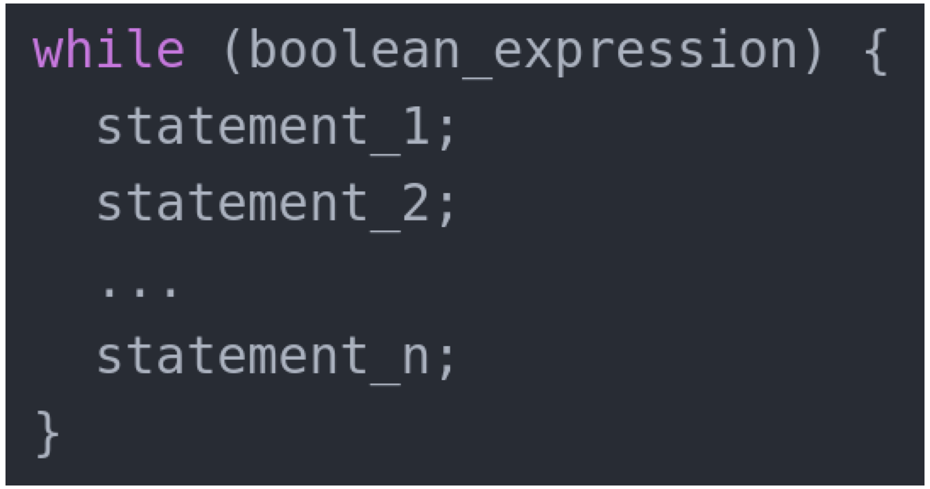
Looping

Loop

- A loop is any program construction that repeats a statement (or a compound statement) a number of times
- The statement to be repeated in a loop is called the body of the loop
- Each repetition of the loop body is called an iteration
- In C++, looping can be achieved using either a while statement or a for statement

while Statement

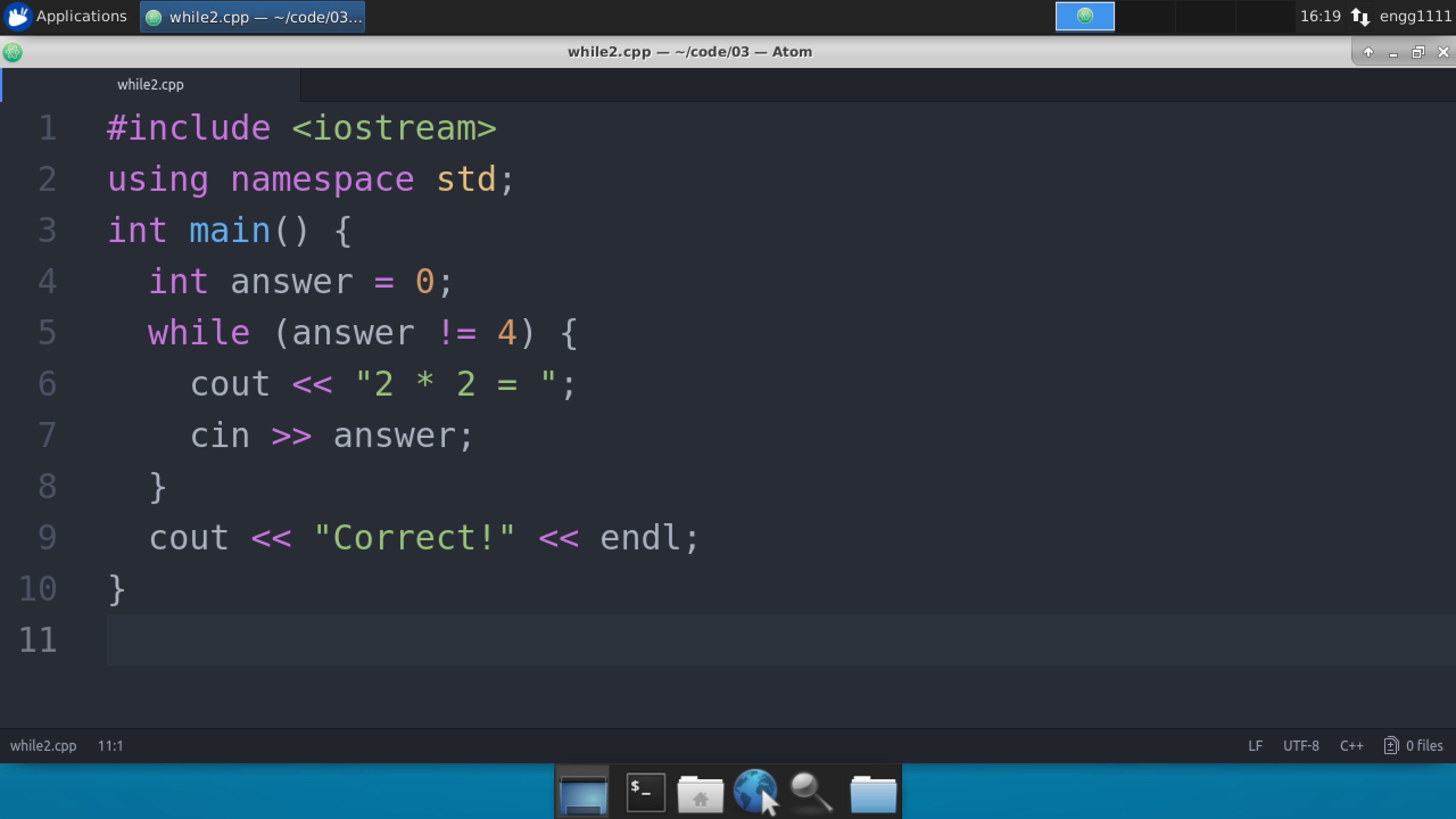
- When a while statement (aka while loop) is executed, the boolean_expression is evaluated
- If it returns true, the loop body is executed once (i.e., one iteration)
- If it returns false, the loop ends without executing its body
- After each iteration, the boolean_expression will be evaluated again and the process repeats



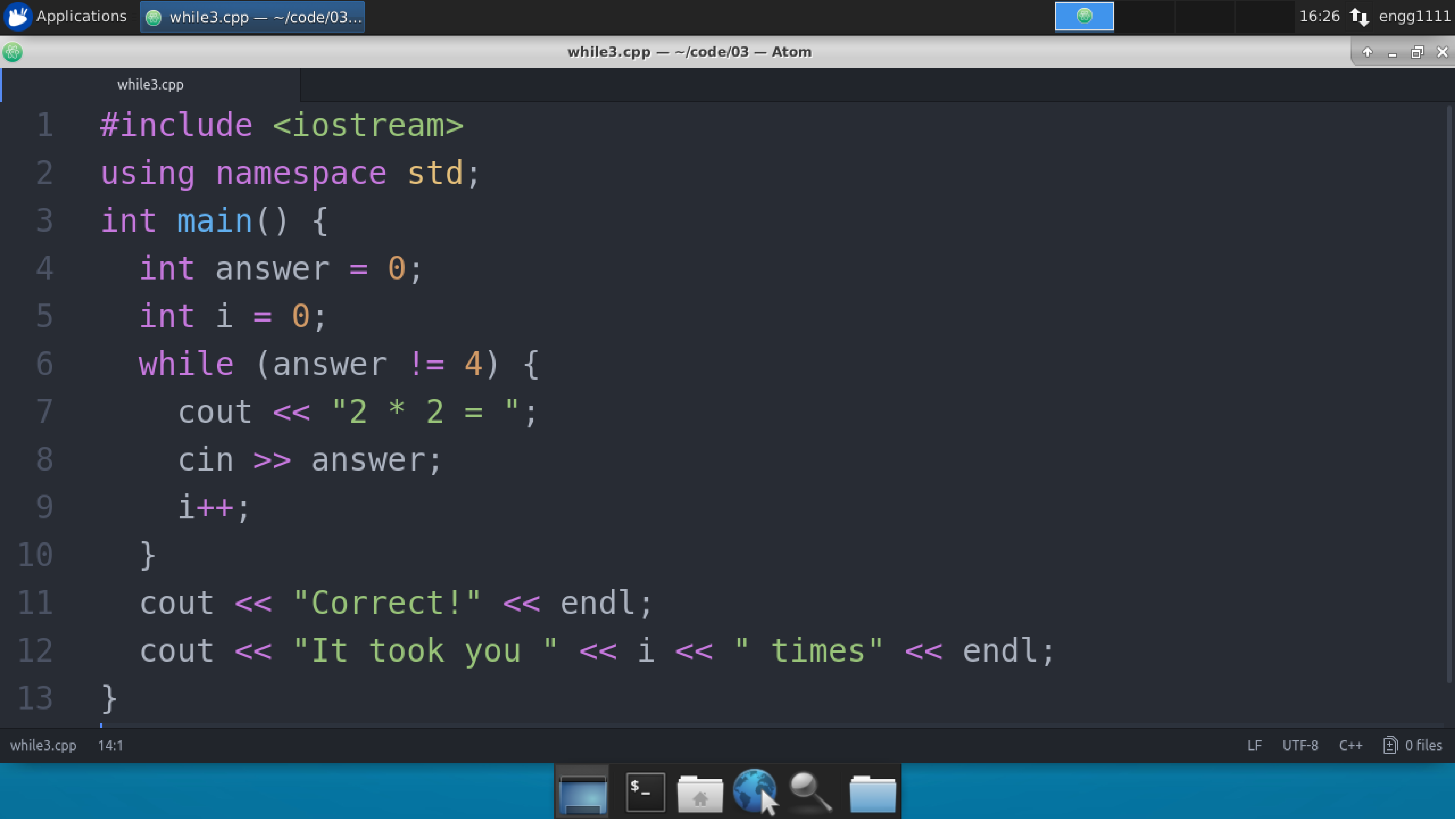
```
while (boolean_expression) {
  statement_1;
  statement_2;
  ...
  statement_n;
}
```

while Statement

- Usage
  - Something that has to repeat again and again as long as the specified condition is true



```cpp
#include <iostream>
using namespace std;
int main() {
  int answer = 0;
  while (answer != 4) {
    cout << "2 * 2 = ";
    cin >> answer;
  }
  cout << "Correct!" << endl;
}
```



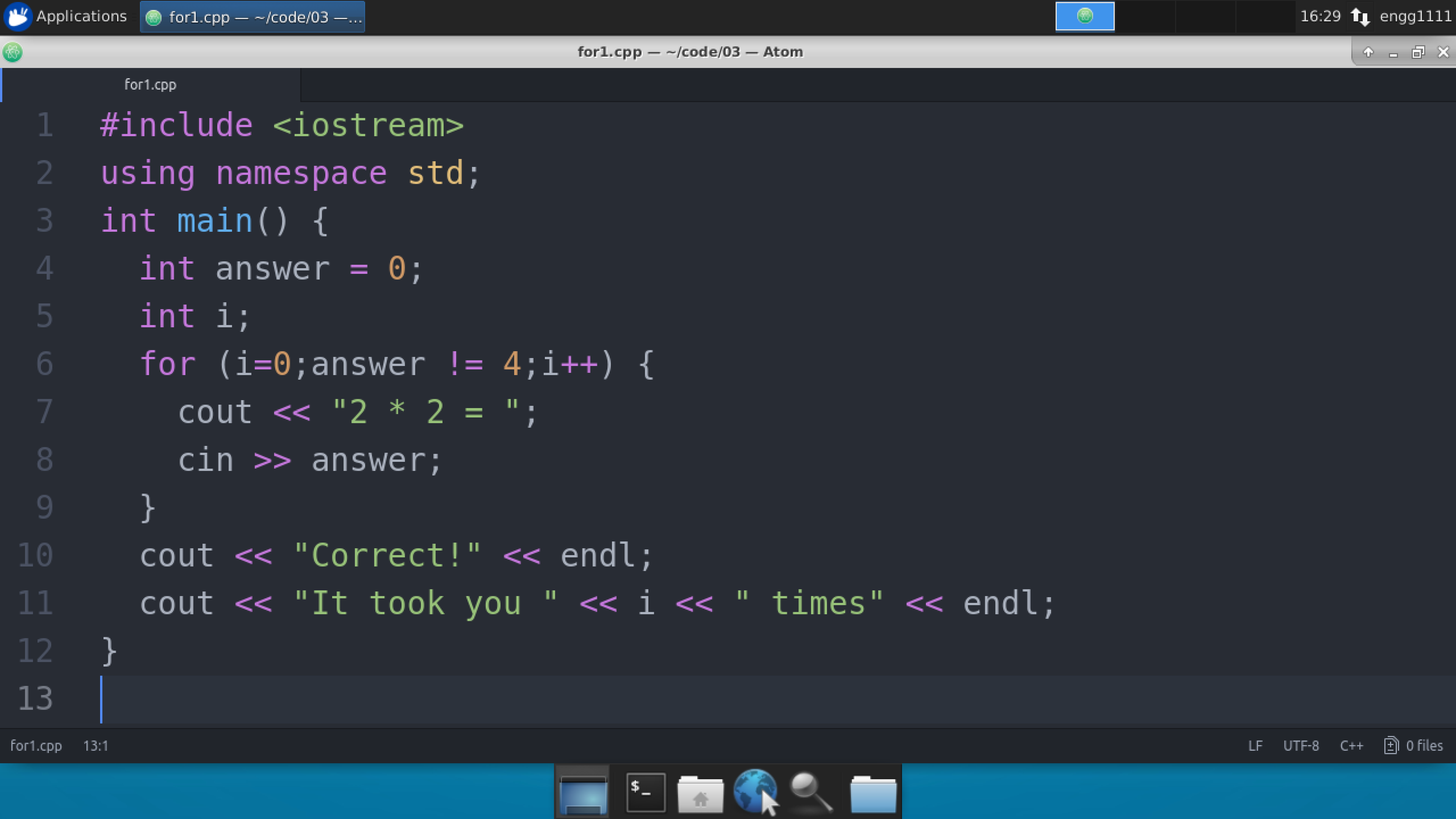
```cpp
#include <iostream>
using namespace std;
int main() {
  int answer = 0;
  int i = 0;
  while (answer != 4) {
    cout << "2 * 2 = ";
    cin >> answer;
    i++;
  }
  cout << "Correct!" << endl;
  cout << "It took you " << i << " times" << endl;
}
```

Quiz

- Write a complete C++ program that outputs the numbers 1 to 20, one per line

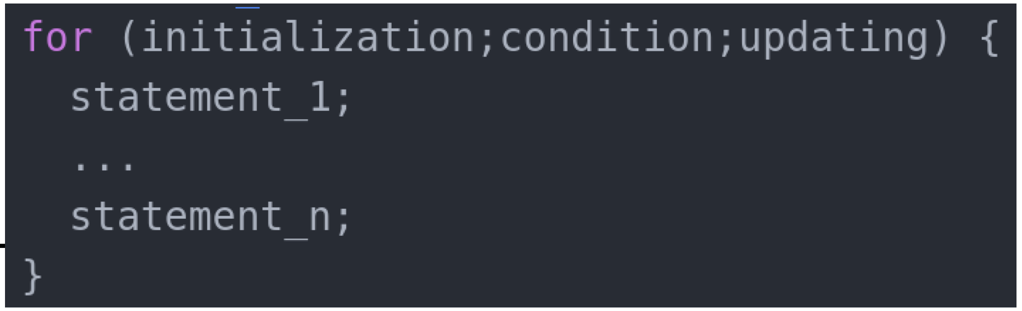
for Statement

- The for statement (aka for loop) in C++ provides a compact way of expressing the typical loop structure



```cpp
#include <iostream>
using namespace std;
int main() {
  int answer = 0;
  int i;
  for (i=0;answer != 4;i++) {
    cout << "2 * 2 = ";
    cin >> answer;
  }
  cout << "Correct!" << endl;
  cout << "It took you " << i << " times" << endl;
}

```

for Statement

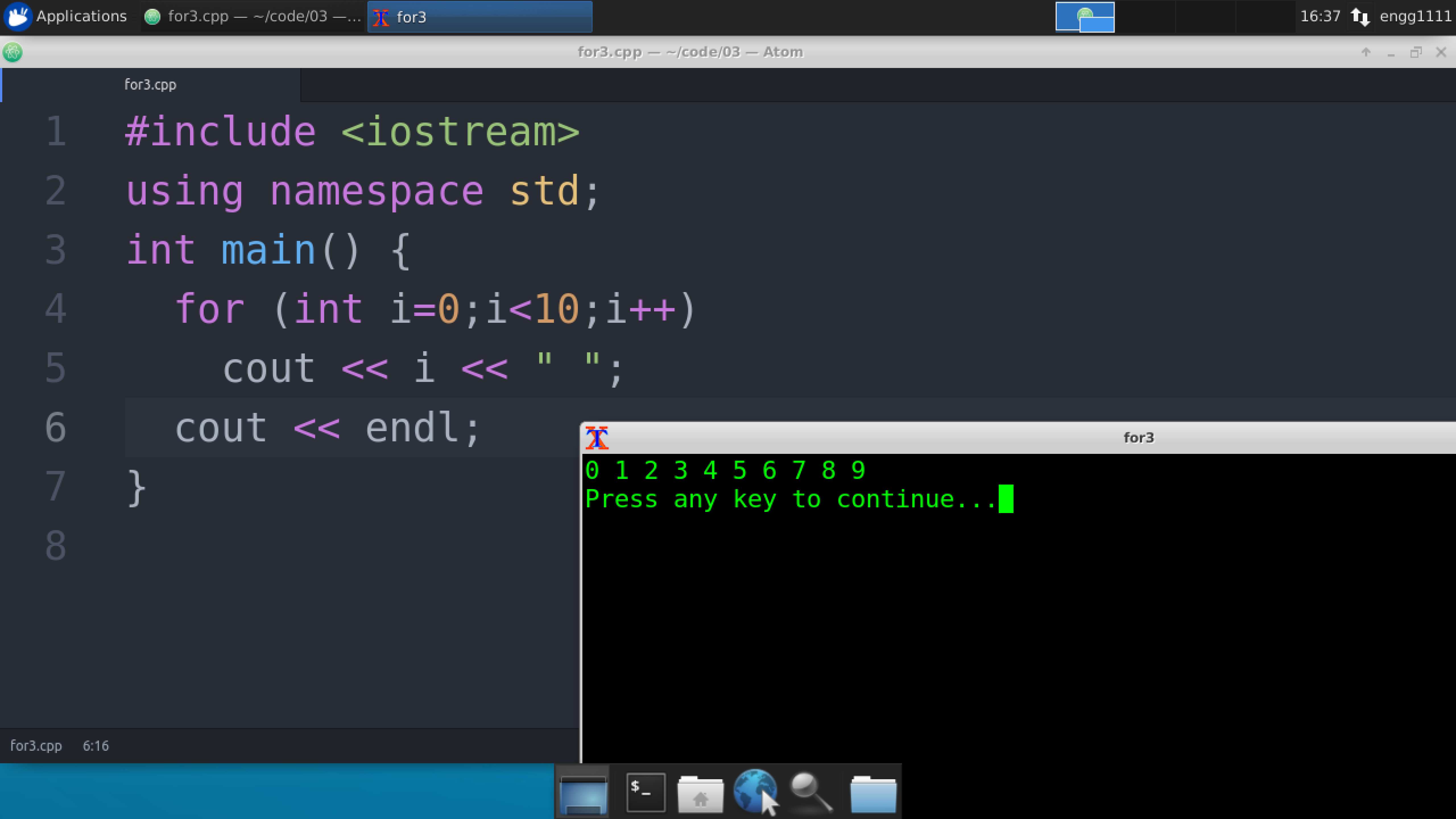


```
for (initialization;condition;updating) {
  statement_1;
  ...
  statement_n;
}
```

- When a for statement is executed
  1. The initialization is performed
     - Generally it sets the initial value of the loop variable
     - The initialization is executed only once
  2. The condition is checked
     - If it is true, the loop body is executed once (i.e., one iteration)
     - If it is false, the loop ends without executing its body
  3. After each iteration, the updating of loop variable is performed and the loop continues at Step 2

Example

- Write a program that outputs 0 1 2 3 4 5 6 7 8 9 using a for loop



```cpp
#include <iostream>
using namespace std;
int main() {
  for (int i=0;i<10;i++)
    cout << i << " ";
  cout << endl;
}
```

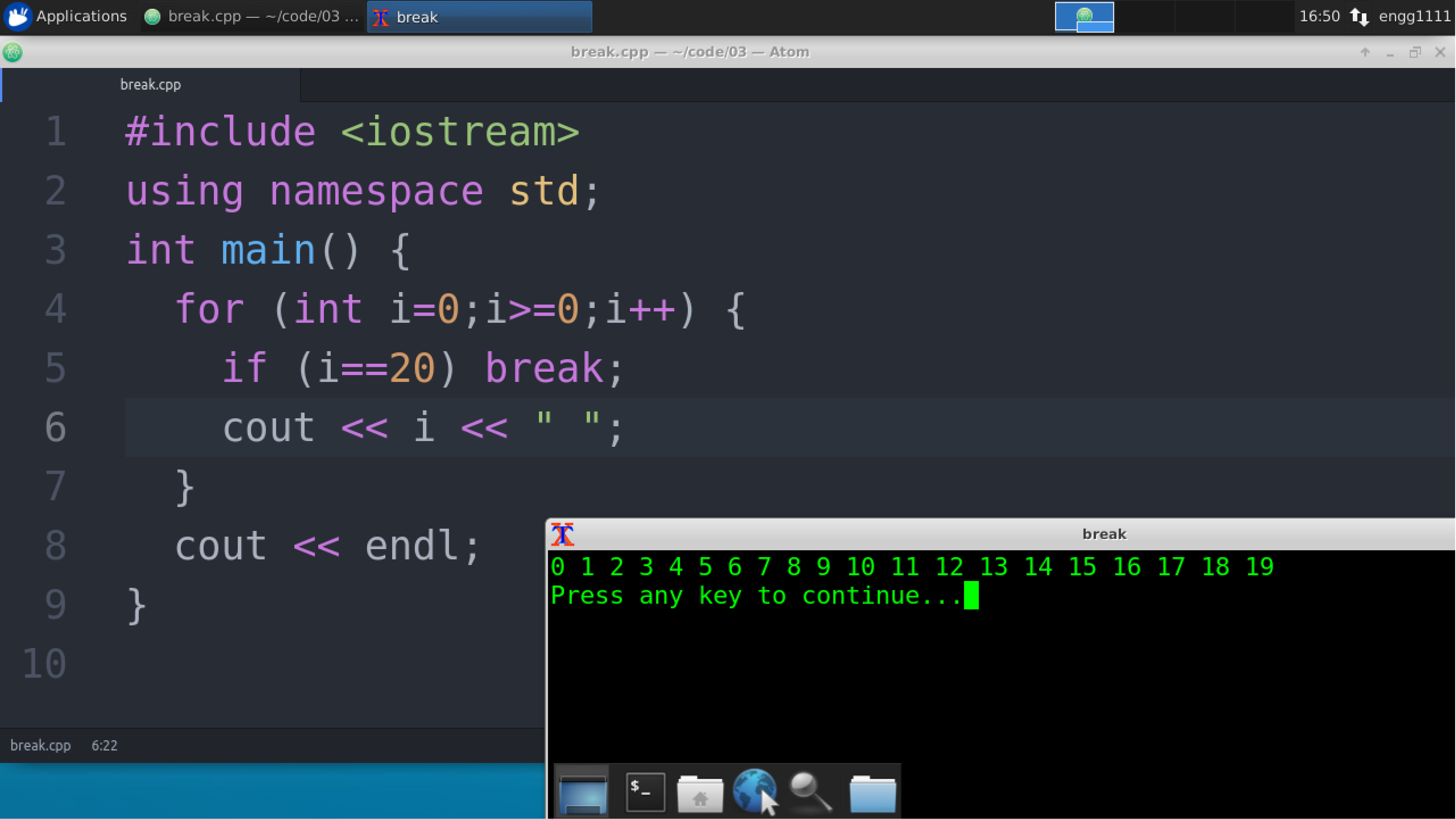
```
0 1 2 3 4 5 6 7 8 9
Press any key to continue...
```

Quiz

- Write a program that calculates the sum of odd numbers between 1 and 20

break Statement

- The break statement can be used to exit a loop
- When a break statement is executed
    - The loop ends immediately
    - The execution continues with the statement following the loop
- The break statement may be used in both while loop and for loop
- Avoid using a break statement to end a loop unless absolutely necessary
    - A proper way to end a loop is using the condition for continuation

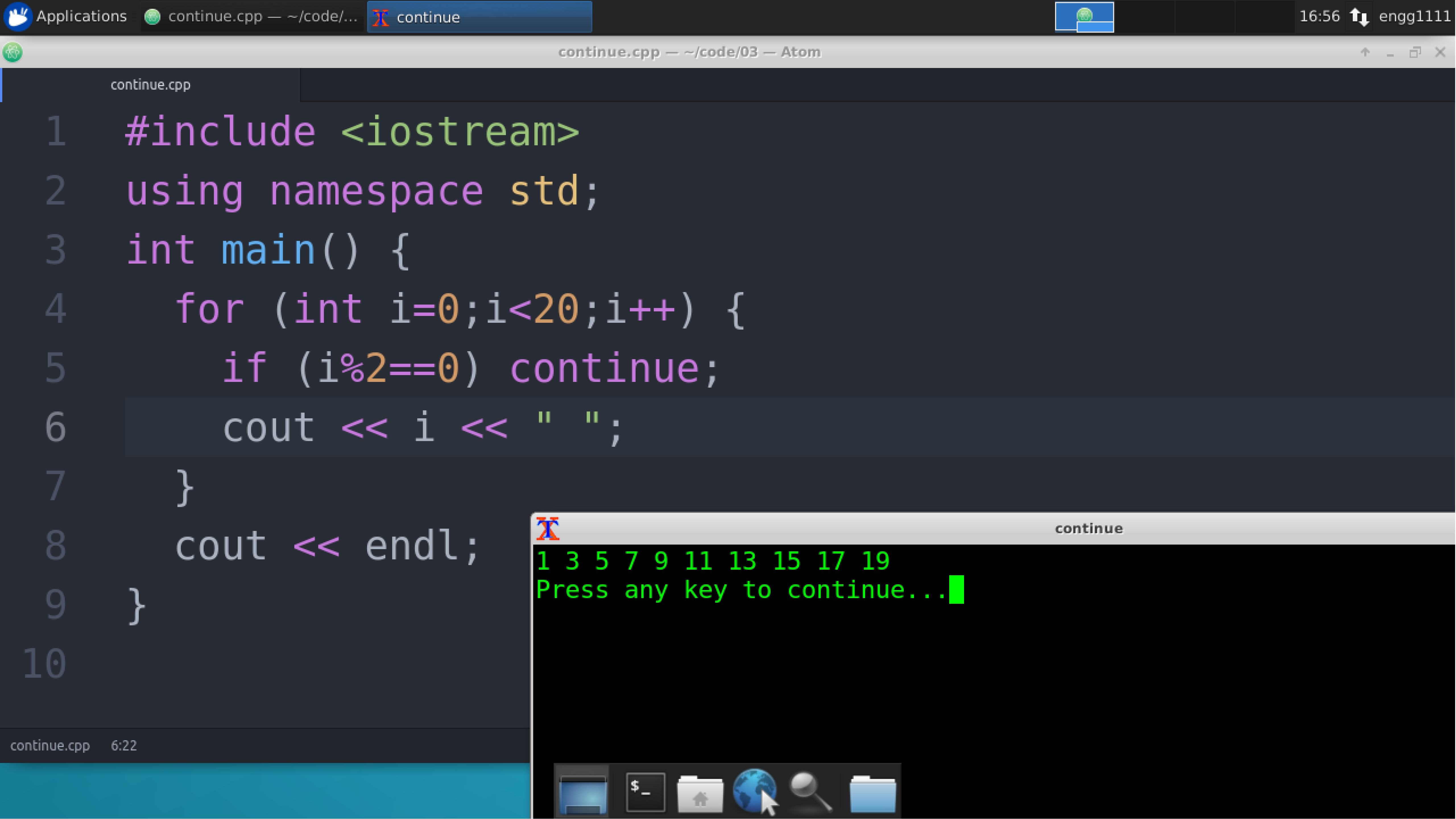


```cpp
#include <iostream>
using namespace std;
int main() {
  for (int i=0;i>=0;i++) {
    if (i==20) break;
    cout << i << " ";
  }
  cout << endl;
}
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Press any key to continue...
```

continue Statement

- The continue statement is used to terminate the current iteration of a loop
- When a continue statement is executed
  - Any loop body statements after it will be skipped
  - The loop continues by starting the next iteration
- Like the break statement, the continue statement may be used in both while loop and for loop



```cpp
#include <iostream>
using namespace std;
int main() {
  for (int i=0;i<20;i++) {
    if (i%2==0) continue;
    cout << i << " ";
  }
  cout << endl;
}
```

continue

```
1 3 5 7 9 11 13 15 17 19
Press any key to continue...
```

Reading

- Problem Solving with C++
  - Chapter 2
- C++ Language Tutorial: Control Structures
  - http://www.cplusplus.com/doc/tutorial/control