Discussion 3

Project 1, Functions (overloading, value vs reference), Style

Project 1

Overview

Your program will be calling accrued interest in a bank account!

- Given an initial deposit, you will calculate how much interest is accrued over a specified # of months
- Interest is compounded **monthly**
- Ex. $1000 over 5 months at 10% interest
  - After 5 months, your balance will be 1000 * 1.1 * 1.1 * 1.1 * 1.1 * 1.1
- See the spec and sample output files for details!

Preparing For Submission

Follow these steps for submitting:

1. Move your files to CAEN and log into CAEN
2. Type “script” and press enter
3. Run “g++ -Wall -std=c++98 project1.cpp -o project1.exe”
4. Run “valgrind --leak-check=full ./project1.exe”
5. Type “exit” and press enter

Submitting Your Project

If there are no issues, email “project1.cpp” and your generated “typescript” file to eecs402@eecs.umich.edu

- **Make sure your subject line follows the correct form**
    - SUBMIT <#> <uniqname>
        - Replace “#” with the current project number
        - Replace “uniqname” with your uniqname
        - Example: SUBMIT 1 yankevn

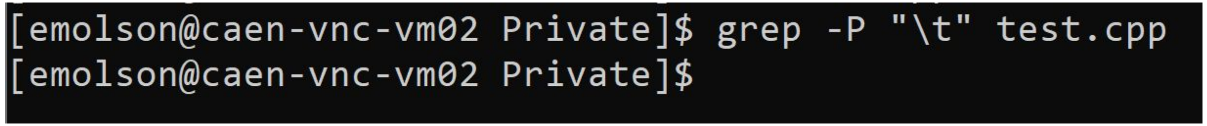
Tips

- Start early! Read the spec and the sample output
    - This project is due on September 21. That's **1 week from yesterday!!!**
- Start building good style habits
    - Remember, you can't get style points back, so get it right the first time
- Come to office hours!!!
    - We can answer any questions about code, structure, style (hint hint), etc.

Style

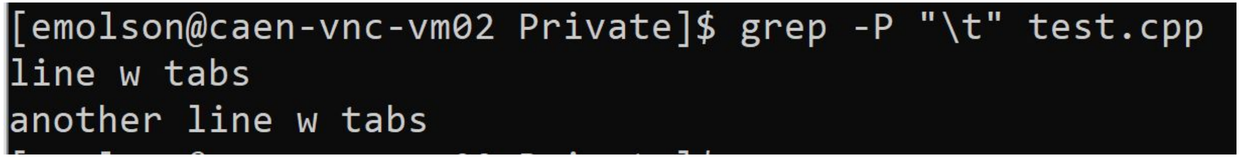
No tabs

- Tabs are disallowed for any coding assignment
- How to check?
    - grep -P “\t” <filename>



```
[emolson@caen-vnc-vm02 Private]$ grep -P "\t" test.cpp
[emolson@caen-vnc-vm02 Private]$
```

Good!



```
[emolson@caen-vnc-vm02 Private]$ grep -P "\t" test.cpp
line w tabs
another line w tabs
```

Not good:/
Prints out the lines that have tabs in them

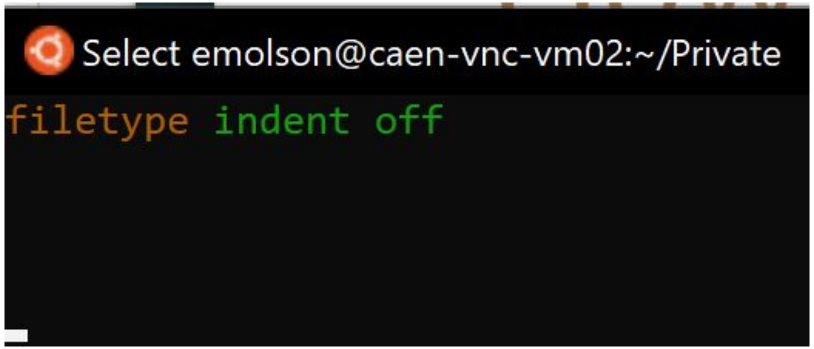
How to turn off auto tabs in VIM

$ vi ~./vimrc   (this creates a new file called vimrc or opens an existing one)

In your new file type:

filetype indent off

Save and quite file



```
filetype indent off
```

How to set auto tabs to spaces (recommended)

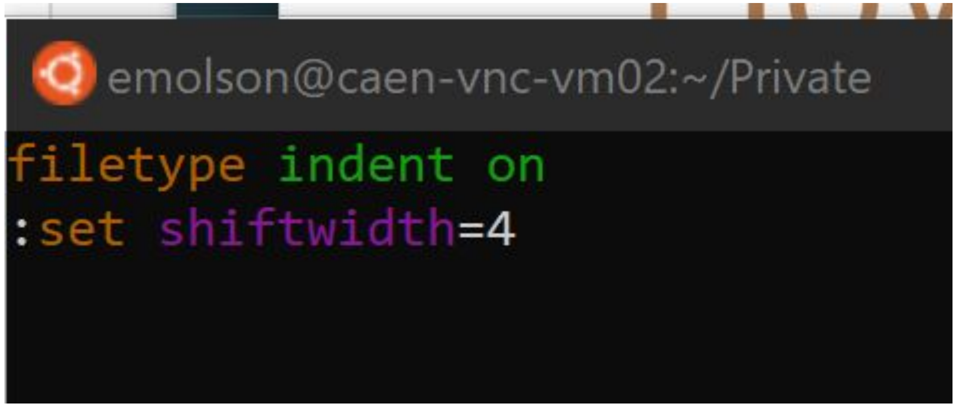
$ vi ~./vimrc   (this creates a new file called vimrc or opens an existing one)

In your new file type:

filetype indent on

:set shiftwidth=4

Save and quite file



```
filetype indent on
:set shiftwidth=4
```

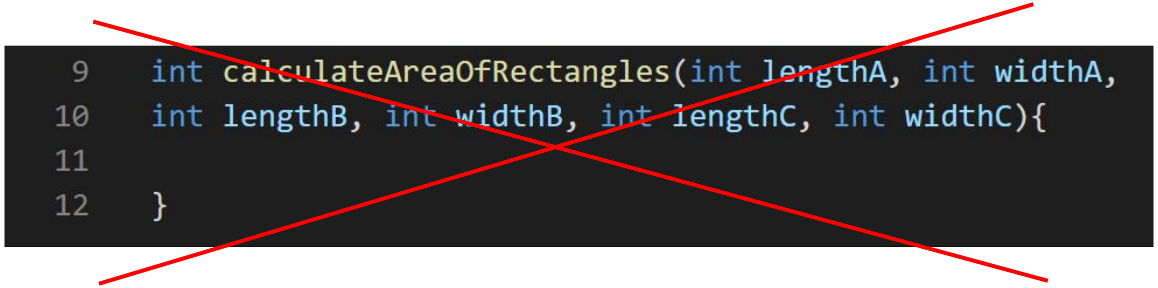
Line width limit

- Lines cannot be longer than 80 characters including leading whitespace
- If a line goes over, start a new line
- How to check?
    - Terminal-based editor: Make screen 80 characters wide (code is always monospaced)

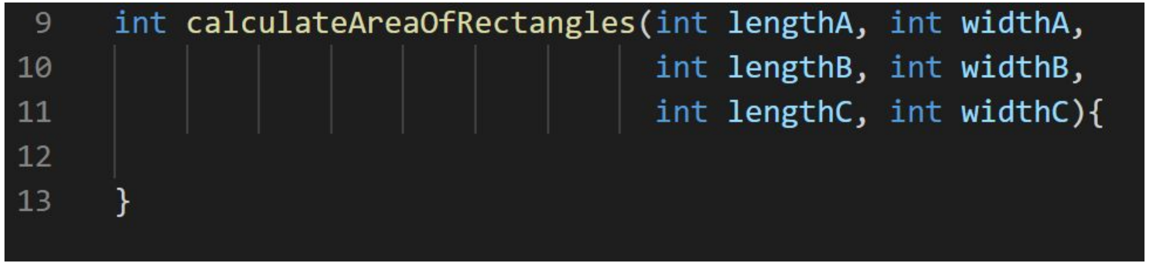
How to avoid long lines

```
:set colorcolumn=80
```

Line width limit



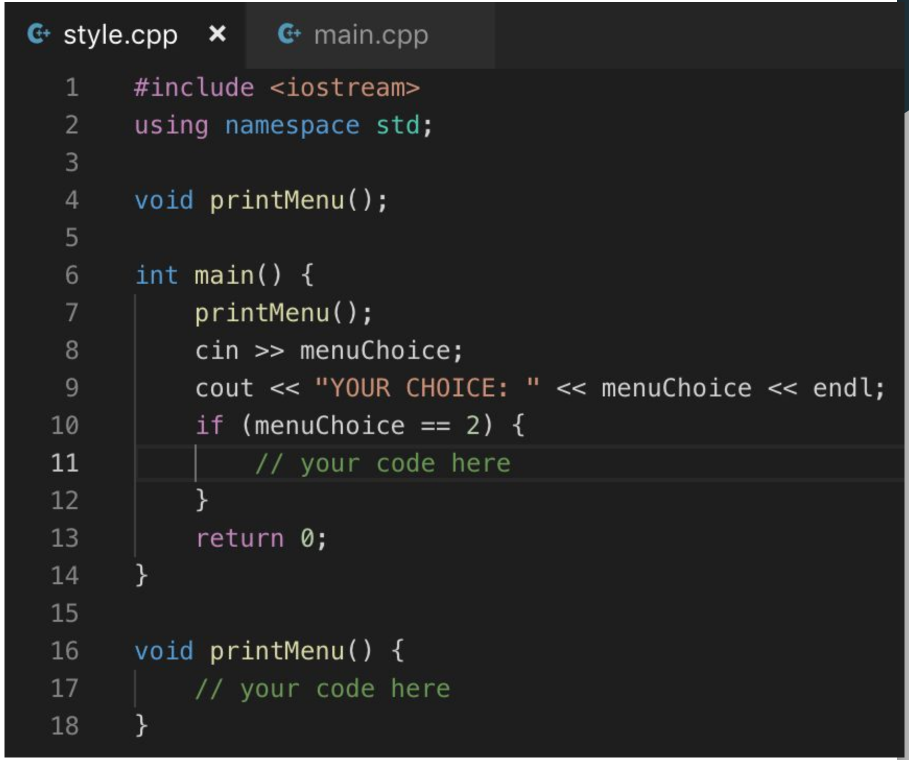
```
int calculateAreaOfRectangles(int lengthA, int widthA,
int lengthB, int widthB, int lengthC, int widthC){

}
```



```
int calculateAreaOfRectangles(int lengthA, int widthA,
                              int lengthB, int widthB,
                              int lengthC, int widthC){

}
```

No “Magic Numbers”

- Number literals should not appear in your code unless it’s clear what they do



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

void printMenu();

int main() {
    printMenu();
    cin >> menuChoice;
    cout << "YOUR CHOICE: " << menuChoice << endl;
    if (menuChoice == 2) {
        // your code here
    }
    return 0;
}

void printMenu() {
    // your code here
}
```

No duplicate code

- Identical (or nearly identical) blocks of code should not exist in multiple parts of your code
- Instead, use functions!

Proper naming conventions

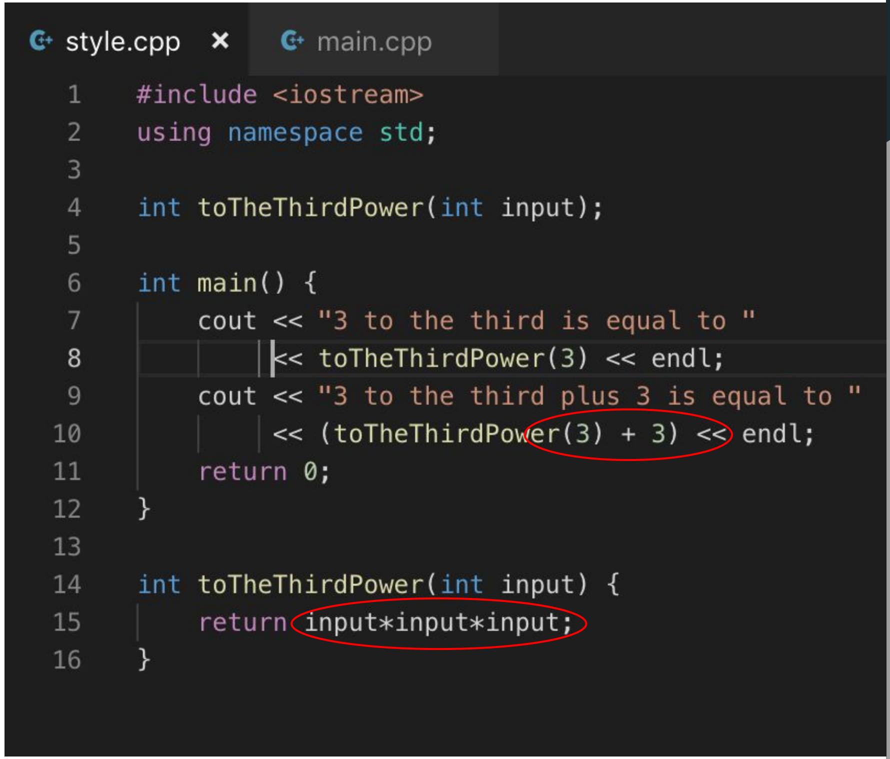
1. camelCase should be used for all non-constant variables and functions
2. UPPERCASE_SNAKE_CASE should be used for all constant variables
3. Functions should be verbs (not relevant for this project)
4. Variables names should be descriptive of what they represent

No Global Variables!

Just don’t

Consistent spacing

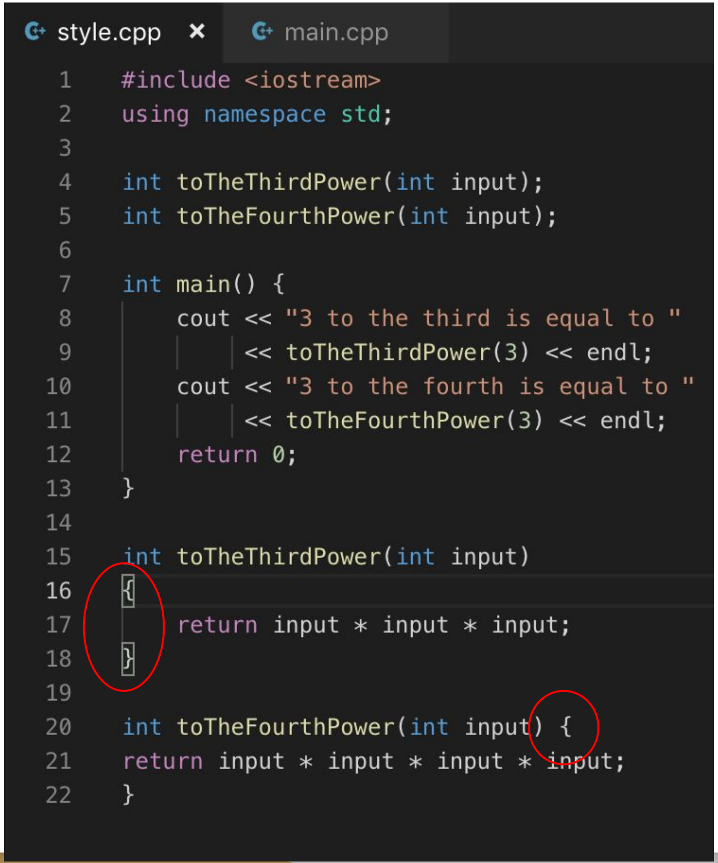
- Operators should have spaces on either side to make it more clear to read



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

int toTheThirdPower(int input);

int main() {
    cout << "3 to the third is equal to "
         << toTheThirdPower(3) << endl;
    cout << "3 to the third plus 3 is equal to "
         << (toTheThirdPower(3) + 3) << endl;
    return 0;
}

int toTheThirdPower(int input) {
    return input*input*input;
}
```

Consistent indentation / { }

- Indentation is required in:
    - Loops
    - Switch statements
    - Functions
    - Line overflow
- Curly braces need to be organized like only one of these two functions, every time they are used



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

int toTheThirdPower(int input);
int toTheFourthPower(int input);

int main() {
    cout << "3 to the third is equal to "
        << toTheThirdPower(3) << endl;
    cout << "3 to the fourth is equal to "
        << toTheFourthPower(3) << endl;
    return 0;
}

int toTheThirdPower(int input)
{
    return input * input * input;
}

int toTheFourthPower(int input) {
return input * input * input * input;
}
```

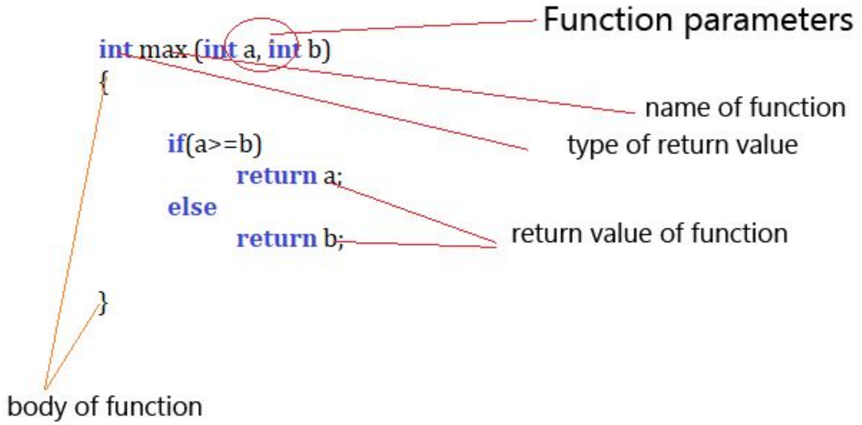
Hint!! Match styles throughout project

Whatever code you add should match your own style

Functions

Functions

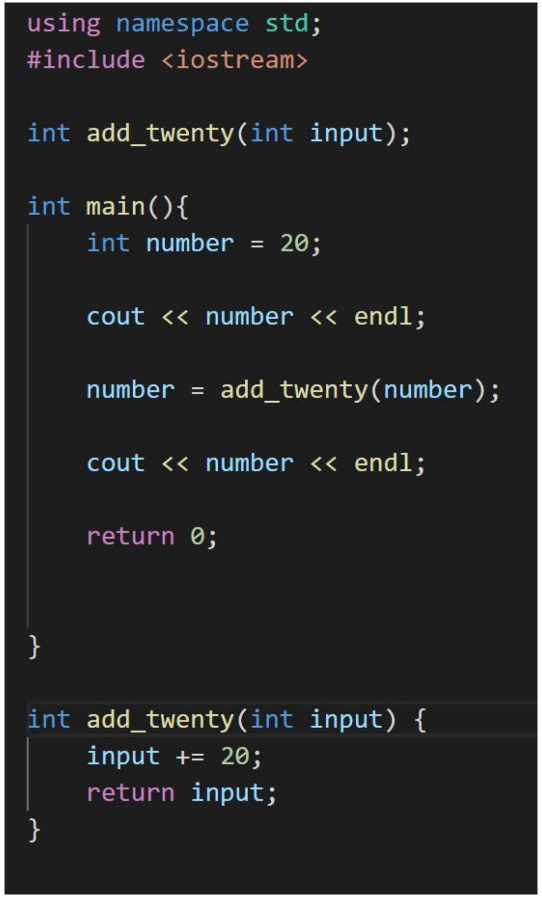
- Functions must be declared before use
- Can take any number of inputs, but can only return up to 1 value



```
int max (int a, int b)
{

        if(a>=b)
                return a;
        else
                return b;

}
```

Functions

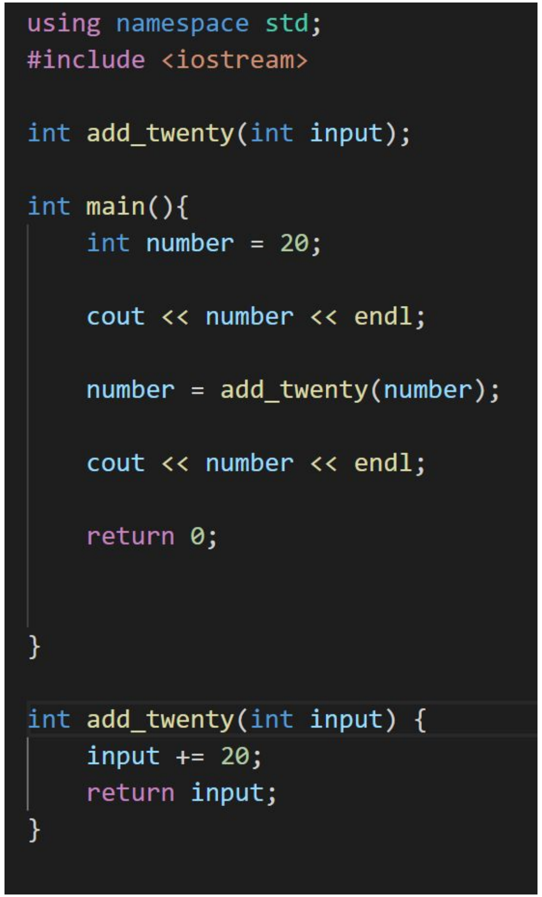
- Functions must be declared before use
- Can take any number of inputs, but can only return up to 1 value



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

int add_twenty(int input);

int main(){
    int number = 20;

    cout << number << endl;

    number = add_twenty(number);

    cout << number << endl;

    return 0;


}

int add_twenty(int input) {
    input += 20;
    return input;
}
```

A note on scope

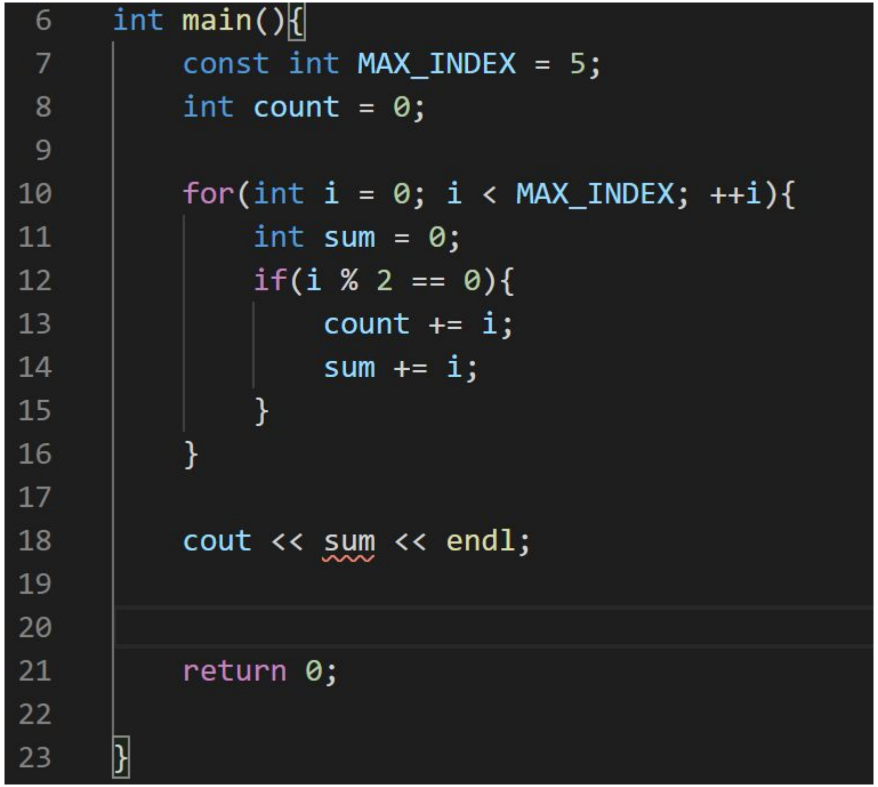
- The variable “input” does not exist in main and therefore cannot be used
- It’s scope is in the “add_twenty” function



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

int add_twenty(int input);

int main(){
    int number = 20;

    cout << number << endl;

    number = add_twenty(number);

    cout << number << endl;

    return 0;


}

int add_twenty(int input) {
    input += 20;
    return input;
}
```

A note on scope

- sum is declared inside the for loop
- sum cannot be used outside of its scope



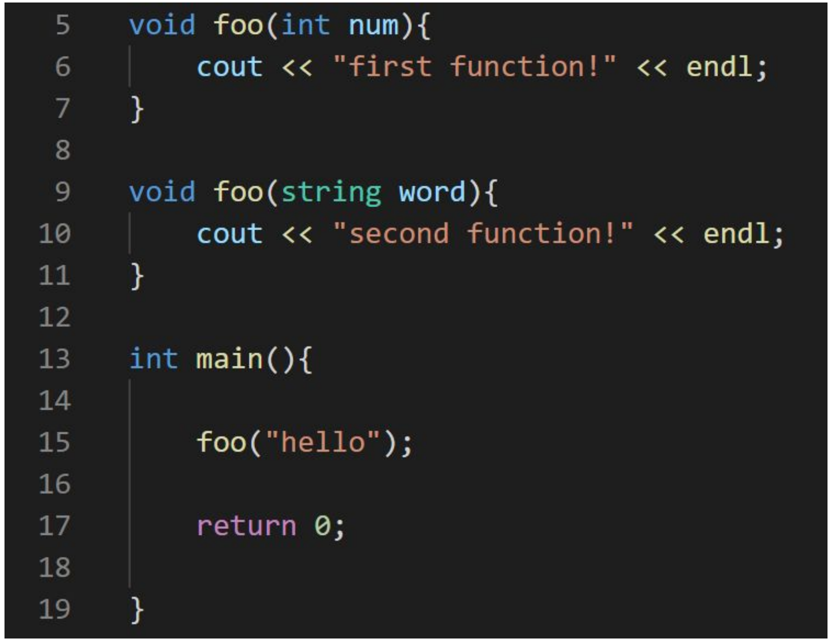
```
int main(){
    const int MAX_INDEX = 5;
    int count = 0;

    for(int i = 0; i < MAX_INDEX; ++i){
        int sum = 0;
        if(i % 2 == 0){
            count += i;
            sum += i;
        }
    }

    cout << sum << endl;


    return 0;

}
```

Function Practice!

Write a function that returns two integers added together!

Function Overloading

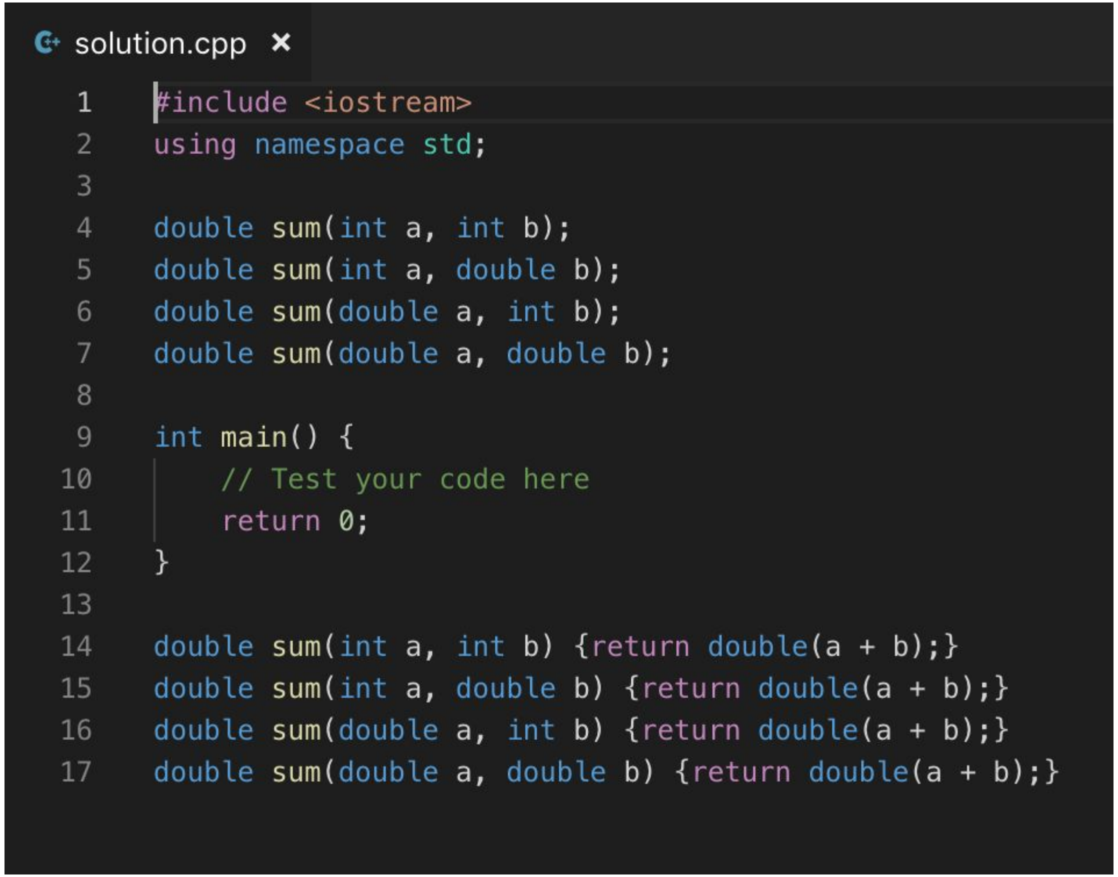
- Using the same name for multiple functions
- How is this allowed?
    - The variables being passed in are different
    - The program chooses which function to use based on what is passed in



```cpp
void foo(int num){
    cout << "first function!" << endl;
}

void foo(string word){
    cout << "second function!" << endl;
}

int main(){

    foo("hello");

    return 0;

}
```

Function Overloading Practice!

- Write a program that can take either ints or doubles and add them together


solution.cpp

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

double sum(int a, int b);
double sum(int a, double b);
double sum(double a, int b);
double sum(double a, double b);

int main() {
    // Test your code here
    return 0;
}

double sum(int a, int b) {return double(a + b);}
double sum(int a, double b) {return double(a + b);}
double sum(double a, int b) {return double(a + b);}
double sum(double a, double b) {return double(a + b);}
```

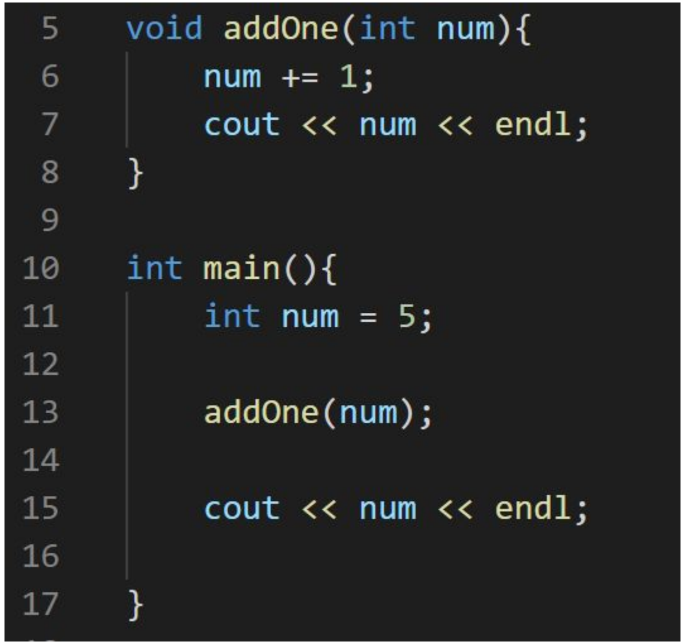
Pass by Reference / Pass by Value

Pass by Value vs Pass by Reference

Pass by value

- What you are used to
- Makes a copy of the variable
- If modified in the called function, it is not changed in the original function

What would the following code output?



```
void addOne(int num){
    num += 1;
    cout << num << endl;
}

int main(){
    int num = 5;

    addOne(num);

    cout << num << endl;

}
```

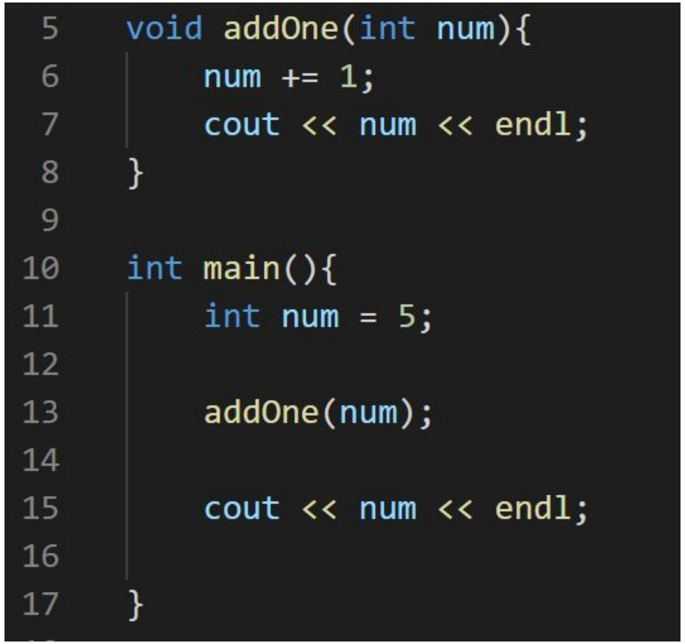
Pass by Value vs Pass by Reference

Pass by value

- What you are used to
- Makes a copy of the variable
- If modified in the called function, it is not changed in the original function

What would the following code output?

6
5



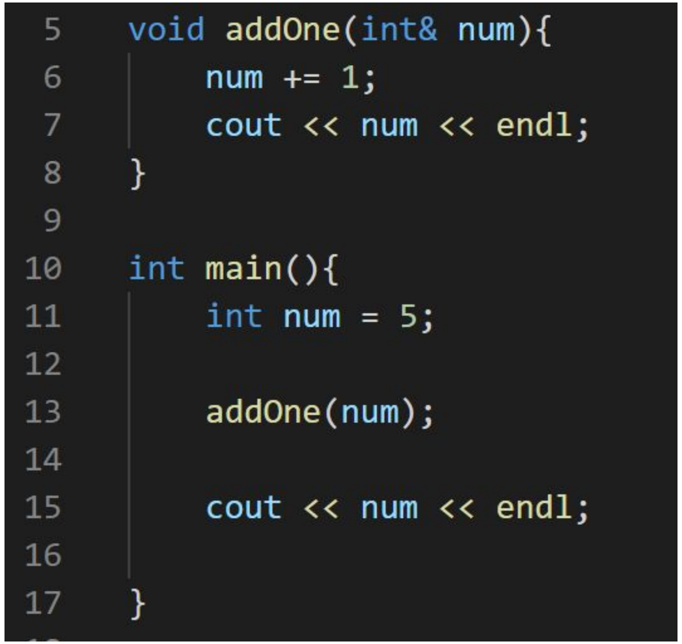
```
void addOne(int num){
    num += 1;
    cout << num << endl;
}

int main(){
    int num = 5;

    addOne(num);

    cout << num << endl;

}
```

Pass by Value vs Pass by Reference

Pass by reference

- Does not make a copy
- Can be modified by the called function

What would the following code output?



```
void addOne(int& num){
    num += 1;
    cout << num << endl;
}

int main(){
    int num = 5;

    addOne(num);

    cout << num << endl;

}
```

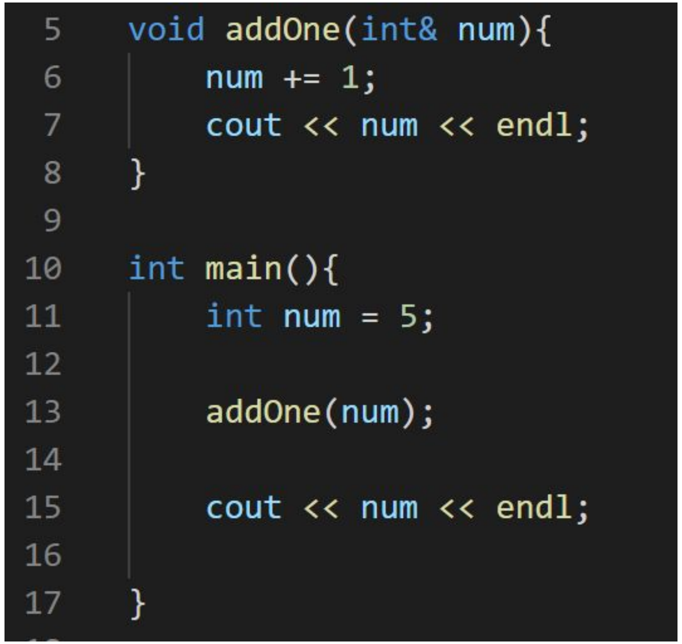
Pass by Value vs Pass by Reference

Pass by reference

- Does not make a copy
- Can be modified by the called function

What would the following code output?

6
6

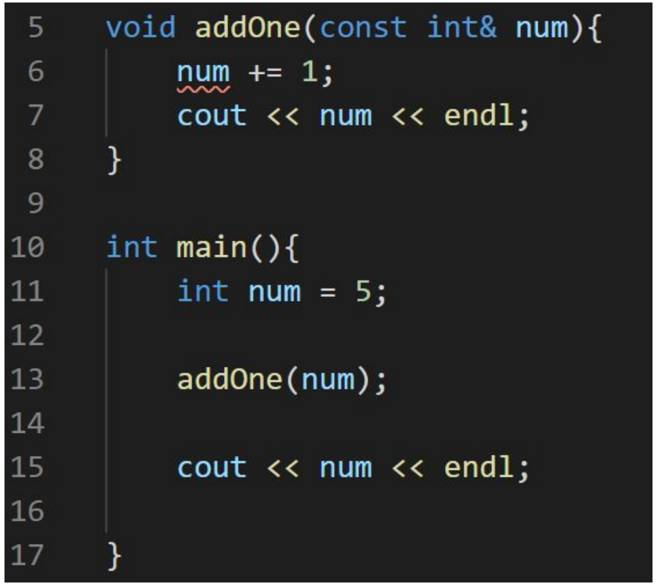


```
void addOne(int& num){
    num += 1;
    cout << num << endl;
}

int main(){
    int num = 5;

    addOne(num);

    cout << num << endl;

}
```

Pass by Value vs Pass by Reference

Pass by const reference

- Does not make a copy
- Can not be modified by the called function
- Useful for large data types



```cpp
void addOne(const int& num){
    num += 1;
    cout << num << endl;
}

int main(){
    int num = 5;

    addOne(num);

    cout << num << endl;

}
```

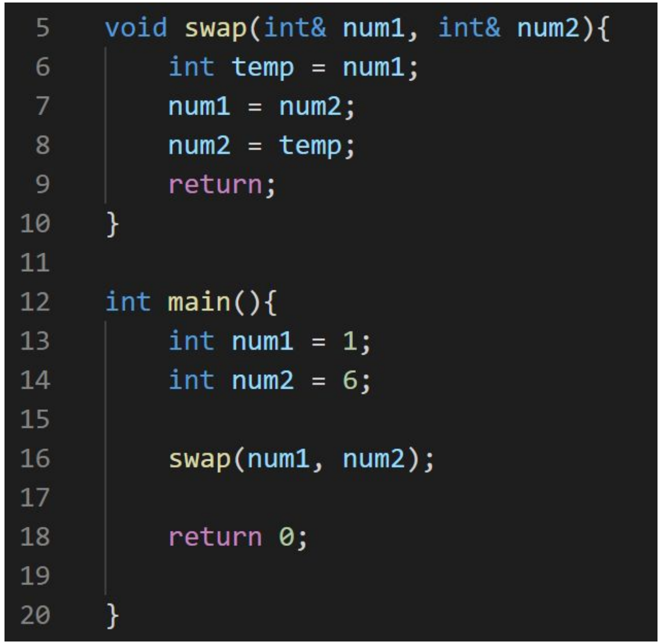
error!

Pass by Reference Example!

Write a function that swaps two integers

Pass by Reference Example!

Write a function that swaps two integers



```
void swap(int& num1, int& num2){
    int temp = num1;
    num1 = num2;
    num2 = temp;
    return;
}

int main(){
    int num1 = 1;
    int num2 = 6;

    swap(num1, num2);

    return 0;

}
```