

1 Week 5/6

1.1 Functions

Functions are the most used tool across programming languages. An advantage of functions is that they allow us to think of parts of code as a black box. When you pass values or variables to a function it outputs new values we can use. They also provide a way to revise code so that we can avoid copying swaths of code. By placing functions in a header file (that you can include in your code), they also allow us to organize our code better. Finally, we might have some variables that we only need for a short part of our code. Functions allow the quarantining of those variables and calculations. This makes it easier to fix and maintain code. A disadvantage of functions is that it takes getting use to reading them. When you start programming, all the code you look at is sequential. Then you learn control flow and find that depending on the state of your variables, some code might be executed and some might not; or you might have loops which repeat until a condition is met. Functions don't do anything until they are called. You can have a function in your code that is never executed unless something is written inside `main`. Once a function is called, you then have to jump to the function definition (wherever it lives) to process the code. The function definition might be above `main`, below `main` (if you have a prior declaration), or even in another `.cpp` file. This can make spotting errors challenging. Pay particular attention to which file the error comes from and the suggested line number.

```
error on line 10
error in line 100 of function _
```

Here is a template of a typical function.

```
type0 functionName(type1 param1, type2 param2, ..., typeN paramN) {
    //code
    return outputVariable;
}
```

Begin by placing your function definitions below where you've included header files and above where you've define the `main` function. The components of a function definition are the `functionName`, which you can assign pretty much anything like we'd done with variable names. Then there's the input parameters and the corresponding indicated types. Each parameter is separated by commas. The last thing is the return type which you see as `type0`. A function returns one parameter or no parameters. That parameter should have type `type0` which could be an `int`, `string`, `vector<int>`, etc. If you don't want to return any parameter then the return type should be set to `void`. Below is an example of a function that takes in two ints and returns their sum as an int. The snippet of code in line 12: `f(idx,jdx)` is called executing function call or calling the function `f` on `idx` and `jdx`.

```
1  #include <iostream>
2  using namespace std;
3
4  int f(int a, int b){
5      int c = a + b;
6      return c;
```

```

7  }
8
9  int main() {
10     int idx = 2;
11     int jdx = 3;
12     cout << f(idx, jdx) << endl;
13     return 0;
14 }

```

You could imagine if you're writing a big code that includes lot of functions, your `main` block of code where you call the function will get moved further and further down. What programmers will do to make it more readable is to put only the function signature at the top, and maybe some function comments. So then they can put the full, possibly long, function definition at the bottom. That way the need to know details from the functions are at the top and we keep the part we're working on (inside `main`) towards the top. Here is an example of that reformatting.

```

#include <iostream>
using namespace std;

/**
 * Calculates the sum of two numbers.
 * @param a The first number.
 * @param b The second number.
 * @return The sum of a and b.
 */
int f(int a, int b);

int main() {
    int idx = 2;
    int jdx = 3;
    cout << f(idx, jdx) << endl;
    return 0;
}

int f(int a, int b){
    int c = a + b;
    // Imagine this is some very
    // long and complex code that
    // you want to hid away at the
    // bottom.
    return c;
}

```

There is a way to even further compartmentalize your code. Next week, we'll talk about separate `hpp` / `cpp` files and how to include them.

1.1.1 Pure Function vs. Procedures

This section covers the classification of functions as pure functions or procedures. It's a good practice to make functions either a pure function or a procedure, otherwise it's implementation can be confusing for users to follow as it will be doing too many things at once.

Pure functions:

- do not have side effects...
- in particular, they do not print things to the console;
- unless we assign to a variable, calling a pure function will not affect the value of any variables in `main`.
- Example:

```
double my_maximum(double x, double y) {  
    if (x>y) {  
        return x;  
    }  
    else {  
        return y;  
    }  
}
```

Procedures:

- do not return anything (the “return type” is `void`);
- they will have side effects...
- they might print things to the console;
- they might cause the values of variables to change in `main`.
- Example:

```
void happyBirthdayToConsole(string name) {  
    string hbty = "Happy Birthday to you\n";  
    string hbd  = "Happy Birthday dear ";  
  
    cout << hbty + hbty + hbd + name + "\n" + hbty << endl;  
}
```

1.2 Vectors

Vectors are a flexible container for holding stuff. They are a templated class which means you can specify what you want them to contain such as all `ints` or all `doubles`. This lets you store more interesting things than the string class which contains elements which are `const chars`. In the future, when you learn structs and classes you'll see more generally how to create containers.

For more information about the vector class, you can view the documentation here:

<https://cplusplus.com/reference/vector/vector/>

Go to the section that says “Member functions” and click on the link to the constructor. These are ways to create a vector. Scroll down and look at the examples.

1.3 Midterm topics

The midterm topics could be as much as this:

Variables and datatypes

- What is a data type?
- What is a variable?
- Can you change the data type of a variable?
- When is the appropriate time to use a certain data type? e.g., `size_t` vs `int` vs `double` vs `boolean`, and their various uses.
- Arithmetic operators (+, -, %, /, *) and logical operators (!, &&, ||)
- Boolean expressions
- The string class, member functions and indexing
- Common errors: build error, runtime error, and logic error.
- Precision and rounding (like `cout << fixed; cout << setprecision(3);`)
- Coercions/conversions (I won't ask you to convert a `char` to an `int` but you should know there's an organized ASCII table for doing that).

Console Input and Output

- Know the rules of `cin`, `getline`, `cin.get`, `cin.peek`, `cin.ignore` and how they interact with the input buffer.

Control Flow

- If, `else if` and `else` statements.
- Loops: `for`, `do while`, `while` loops and the situations in which each is most appropriate.
- Know how to process control flow and the extent of the scopes. (When is the iterating variable created and destroyed in a `for` loop? When do you evaluate the keep going condition in a `while` loop?)

Scope

- What is a block or local scope?
- Be able to analyze code and determine where variables defined relative to the scope and know what variables are referenced or copied.

Functions

- Function declaration and definition.
- The difference between using a data type vs. `void` in the function declaration.
- What happens when a function is called?
- Function scope & passing arguments to parameters.
- Pure functions vs procedures.

- References, passing by reference vs. passing by value vs. passing by reference to const (Monday).

Vectors

- The templated vector class, member functions and indexing.

Misc

- const correctness
- hpp / cpp arrangement

1.4 Exercises

1.4.1 Practice coding exercises

1. A very common use of functions in programming is to use them as math functions. Suppose you are interested in population growth. The logistic function used to model a population, P , is

$$P(t) = \frac{K P_0}{P_0 + (K - P_0) e^{-rt}}$$

where K is carrying capacity (maximum population density), P_0 is the initial population, r is the growth rate. Write a function which takes in a time t and returns a population P . The other parameters will be defined locally in the function. Try `double K = 1072764; double P0 = 900; double r = 0.2311;`.

2. Write a function `factorial` that computes the factorial for some positive integer `n`. The formula for the factorial of n is

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$$

$$0! = 1$$

After executing the following code,

```
int main() {
    cout << factorial(0) << endl;
    cout << factorial(1) << endl;
    cout << factorial(5) << endl;
    cout << factorial(10) << endl;

    return 0;
}
```

You should see in the console

```
1
1
120
3628800
```

3. The binomial coefficient is useful for computing probabilities. For example, calculating the chances of winning a lottery or the number of comparison in a merge sort algorithm (which you might learn in week 10).

The factorial formula is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Write a function `binomialCoeff` that takes in two integers `k` and `n` where $0 \leq k \leq n$. It should make use of the `factorial` function you wrote. Do not write the function definition for `factorial` inside `binomialCoeff` –make use of the function you created.

4. Star problem

- (a) Write a function `starProblemPrinter` that prints out a star pattern so that given an input n , you see n rows where the number of stars decreases from n to 1. Below is an example of the function call.

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

int main() {
    starProblemPrinter(5);
    return 0;
}
```

will result in the following pattern in the console.

```
*****
****
***
**
*
```

- (b) Next we will generalize this problem. Write a function that prints the character `myChar` n times in the console. Below are two function calls.

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

int main() {
    repeatPrinter(5, '*');
    cout << endl;
    repeatPrinter(3, '!');
    cout << endl;
    return 0;
}
```

will result in the following in the console.

```
*****  
!!!
```

(c) Create a second version of `starProblemPrinter` that uses `repeatPrinter`.

5. Consider the following function written to automate transactions.

```
#include <iostream>  
using namespace std;  
  
string Transaction(bool isStocked, bool isPaidFor, double price, double payment){  
    if (isStocked){  
        if (isPaidFor){  
            if (price == payment){  
                return "Your order has been placed.\n";  
            } else {  
                return "Your payment doesn't match the price.\n";  
            }  
        } else {  
            return "No payment was made.\n";  
        }  
    } else {  
        return "Item not in stock.\n";  
    }  
}  
  
int main() {  
    cout << Transaction(true, true, 100.0, 100.0);  
    cout << Transaction(true, true, 100.0, 10.0);  
    cout << Transaction(true, false, 100.0, 100.0);  
    cout << Transaction(false, false, 100.0, 100.0);  
  
    return 0;  
}
```

Console Output:

```
Your order has been placed.  
Your payment doesn't match the price.  
No payment was made.  
Item not in stock.
```

Write code that refactors the control flow so there's no nesting. Make use of multiple `return` statements.

6. Write functions that tell us various facts about a number, `n`. These function should not print anything themselves but should **return** a result.

- (a) Is it a prime number? Ex: `bool isPrime(int n)`
- (b) Is it a perfect square? There exists some integer `m` such that `n = m*m`.
- (c) Is it even or odd?
- (d) How many digits are in the number?

7. Determine whether the following functions are an example of a pure function, procedure, or neither.

- (a)

```
/**
    Returns the sum of the digits of a positive integer
    @param n : the input positive integer
    @return the sum of the digits of n
 */
unsigned int sumOfDigits(unsigned int n) {
    unsigned int sum = 0;

    while (n != 0) {
        sum += n % 10;
        n /= 10;
    }
    return sum;
}
```
- (b)

```
/**
    Prints the indices at which a word shows up in a phrase
    @param phrase : the phrase to look in
    @param word : the word to look for
 */
void printOccurences(string phrase, string word) {
    const size_t lengthOfPhrase = phrase.length();
    const size_t lengthOfWord = word.length();

    for (size_t i=0; i<lengthOfPhrase; ++i) {
        if (phrase.substr(i,lengthOfWord) == word) {
            cout << i << ' ';
        }
    }
    cout << endl;
}
```

- (c) */***
Returns a string containing the happy birthday song for a specified person
@param name : the name of the person
*@return the happy birthday song for a person with name *name**
**/*
 string happyBirthdayToString(string name) {
 string hbty = "Happy Birthday to you\n";
 string hbd = "Happy Birthday dear ";

 return hbty + hbty + hbd + name + "\n" + hbty;
 }
- (d) */***
Performs a swap on the values of the local variables,
but not on the values of the original variables.
@param i : first int
@param j : second int
**/*
 void mySwapAttempt(int i, int j) {
 int tmp = i;
 i = j;
 j = tmp;
 }
- (e) */***
Performs a swap on the values of the original variables
by using references
@param i : first int
@param j : second int
**/*
 void mySwapFix(int& i, int& j) {
 int tmp = i;
 i = j;
 j = tmp;
 }
- (f) */***
Prints and returns the maximum of two doubles.
@param x : first double
@param y : second double
@return the biggest double of the two.
**/*
 double my_maximum(double x, double y) {
 if (x>y) {
 cout << x << endl;
 }
 }

```

        return x;
    }
    else {
        cout << y << endl;
        return y;
    }
}

```

8. I declare and assign a vector of ints `vector<int> v = {0,1,2,3,4};`

As separate snippets, what do each of the actions separately do to `v`? They might cause an error. See if you can understand what's wrong from the error message.

- (a) `v.pop_back(4);`
- (b) `v.push_back(4);`
- (c) `v.push_back();`
- (d) `remove(v[4]);`
- (e) `v.pop_back();`
- (f) `delete(v);`

9. Create a vector of ints that has 3 rows, each row with 5 columns, all with the value 8. Do it one line.

10. What is the best data type to use where you see ??? in the following code.

```

#include <iostream>
using namespace std;

int main() {

    vector<int> v = {1,2,3};
    for (??? idx = 0, N = v.size(); idx < N; ++idx) {
        v.pop_back();
    }
    cout << v.size() << endl;

    return 0;
}

```

11. What is the best data type to use where you see ??? in the following code.

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

int main() {
    ??? sum = 0;
    for (??? idx = 0, N = 100; idx < N; ++idx) {
        sum += 2*idx - 7;
    }
    cout << sum << endl;

    return 0;
}
```

12. What is the best data type to use where you see ??? in the following code.

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

int main() {
    ??? sum = 0;
    ??? sign = 1;
    for (??? idx = 1, N = 100; idx < N; ++idx) {
        sum += sign*1/idx;
        sign *= -1;
    }
    cout << sum << endl;
    cout << log(2) << endl;

    return 0;
}
```

1.4.2 Predict the output exercises

Write the console outputs (not including the Program ended with exit code: 0 line). Assume the code runs error-free, includes the appropriate headers, and is using std namespace.

13. `#include <iostream>`
`#include <vector>`
`using namespace std;`

```

int main() {
    vector<vector<int>> v(3, vector<int>(5,7));
    v[0][1] = 1;
    v[2][4] = 8;

    for (size_t idx = 0; idx < v.size(); ++idx){
        for (size_t jdx = 0; jdx < v[0].size(); ++jdx){
            cout << v[idx][jdx] << ' ';
        }
        cout << endl;
    }

    return 0;
}

```

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

```

void f(int& a, int b, int c){
    a += 1;
    b += 2;
    c += b;
    cout << a << ' ' << b << ' ' << c << endl;
}

```

```

int main() {
    int a = 10;
    int b = 20;
    int c = 30;

    f(a, b, c);
    cout << a << ' ' << b << ' ' << c << endl;

    return 0;
}

```

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

```

int func(int& a, int b, int& c) {
    cout << a << ' ' << b << ' ' << c << endl;
    a += 1+c;
    b *= 2;
    cout << a << ' ' << b << ' ' << c << endl;
}

```

```

    return a;
}

int main() {
    int a = 20, b = 30, c = 40;
    cout << a << ' ' << b << ' ' << c << endl;

    func(...);

    cout << a << ' ' << b << ' ' << c << endl;
    cout << endl;

    return 0;
}

```

Predict the output when someone executes the following different lines of code in `main`.

- (a) `func(a, b, c);`
- (b) `func(c, b, a);`
- (c) `func(b, b, b);`

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

```

int f(int& x, int y, int z) {
    x--;
    return x+1;
}

int main() {
    int x = 2;
    int y = 2;
    int z = 0;
    int w = f(x, y, z);

    cout << x << " " << y << " " << z << " " << w << endl;
    return 0;
}

```

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

```

void foo(int& x, int& y, int z = 100) {
    x += 2;
    y = 0;
    z = z % 2;
}

int main() {
    int x = 10, y = 20, z = 30;
    cout << x << " " << y << " " << z << endl;
    foo(z, x, y);
    cout << x << " " << y << " " << z << endl;
    foo(y, z);
    cout << x << " " << y << " " << z << endl;
    return 0;
}

```

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

```

int f(int x) {
    return x * x;
}

int f(double x) {
    return x * x * x;
}

int main() {
    int a = 2;
    double b = 2.0;
    cout << f(a) << endl;
    cout << f(b) << endl;
    cout << f(2.5) << endl;

    return 0;
}

```

19. `#include <iostream>`
`#include <string>`
`using namespace std;`

```

string x_function(const string& word, const string& el){

```

```

size_t i = word.find(el);
if (el != ""){
    if (i != static_cast<size_t>(-1)){
        return word.substr(i, word.length()-i);
    }
} else {
    return word;
}
return "nope";
}

```

```

int main() {
    string word = "aliens exist";
    string ch = "o";
    cout << x_function(word, ch) << endl;

    ch = "x";
    cout << x_function(word, ch) << endl;

    ch = "";
    cout << x_function(word, ch) << endl;

    return 0;
}

```

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

```

int main() {
    int i, j = 0;

    i = 0;
    j = i++;
    cout << "(a) -> " << "i: " << i << ", j: " << j << endl;

    i = 0;
    j = ++i;
    cout << "(b) -> " << "i: " << i << ", j: " << j << endl;

    i = 0;
    j = i++ + i++;
    cout << "(c) -> " << "i: " << i << ", j: " << j << endl;
}

```

```

    i = 0;
    j = ++i + ++i;
    cout << "(d) -> " << "i: " << i << ", j: " << j << endl;

    i = 0;
    j = i++ + ++i;
    cout << "(e) -> " << "i: " << i << ", j: " << j << endl;

    i = 0;
    j = ++i + i++;
    cout << "(f) -> " << "i: " << i << ", j: " << j << endl;

    return 0;
}

```

1.4.3 Error spotting exercises

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.

```

21.  1  #include <iostream>
      2  using namespace std;
      3
      4  /* This code is meant to print out the min of 0 and 1 */
      5  int my_min(int a, int b){
      6      if (a>b){ return b;}
      7      if (a<b){ return a;}
      8  }
      9
     10  int main() {
     11      cout << my_min(0,1) << endl;
     12      return 0;
     13  }

```

```

22. 1  #include <iostream>
    2  using namespace std;
    3
    4  /* This code is meant to print "Hello, World!" to the console. */
    5  int main() {
    6      cout << "Hello, World!\n";
    7
    8  }

23. 1  #include <iostream>
    2  #include <vector>
    3  using namespace std;
    4
    5  /* This code is meant to print out all the elements in the vector. */
    6  int main() {
    7      vector<int> v = {1, 2, 3, 5, 8, 13};
    8      for (size_t i = 0; i <= v.size(); ++i) {
    9          cout << v.at(i) << ' ';
    10     }
    11     return 0;
    12 }

24. 1  #include <iostream>
    2  #include <vector>
    3  using namespace std;
    4
    5  /* This code is meant to add 1 to the 0th element in v. */
    6  void f(vector<int>& v){
    7      int i = 0;
    8      v[0] += 1;
    9  }
    10
    11 int main() {
    12     std::vector<int> v = {1, 1, 1, 1};
    13     v = f(v);
    14     return 0;
    15 }

25. 1  #include <iostream>
    2  #include <vector>

```

```

3  using namespace std;
4
5  /* This code is meant to print out the contents of vector v. */
6  int main() {
7      vector<vector<int>> v(2, vector<int>(2,8));
8
9      for (size_t idx = 0; idx < v.size(); ++idx){
10         for (size_t jdx = 0; idx < v[0].size(); ++jdx){
11             cout << v[idx][jdx] << ' ';
12         }
13         cout << endl;
14     }
15
16     return 0;
17 }

```

26.

```

1  #include <iostream>
2  using namespace std;
3
4  /* This code is meant to print out the value of x.
5     This is all the code. */
6  int main() {
7
8      int x = 10;
9
10     void func(int y);
11     func(x);
12
13     return 0;
14 }

```

27.

```

1  #include <iostream>
2  using namespace std;
3
4  /**
5   The absolute value function.
6   @param n The number to take the absolute value of.
7   @return The absolute value of n.
8   */
9  int my_abs(int n) {
10     if (n < 0) {
11         break -n;
12     } else {
13         break n;

```

```

14     }
15 }
16
17 /* This code is meant to print:
18 3
19 3
20 */
21 int main() {
22     cout << my_abs(-3) << endl;
23     cout << my_abs(3) << endl;
24     return 0;
25 }

```

28.

```

1  #include <iostream>
2  using namespace std;
3
4  /**
5   Adds two numbers.
6   @param a The first number.
7   @param b The second number.
8   @ return Their sum.
9   */
10 int my_add(int a, int b) {
11     return a + b;
12 }
13
14 /* This code is meant to print:
15 5.5
16 */
17 int main() {
18     cout << my_add(3,2.5) << endl;
19     return 0;
20 }

```

29. (The all_caps function is fine.)

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  /**
6   Capitalizes all letters in a string that are lowercase.
7   @param str string to capitalize.
8   @return All caps string.

```

```

9  */
10 string all_caps(string& str) {
11     for (int idx = 0; idx < str.length(); idx++) {
12         if (str[idx] >= 'a' && str[idx] <= 'z') {
13             str[idx] -= 'a' - 'A';
14         }
15     }
16     return str;
17 }
18
19 /**
20  Yells a name.
21  @param first The first name.
22  @param last The last name.
23  */
24 void yell_name(string& first, string& last) {
25     cout << all_caps(first) << '!' << ' ' << all_caps(last) << '!' << endl;
26 }
27
28 /* This code is meant to print:
29    BATMAN!
30    */
31 int main() {
32     string name = "Batman";
33     yell_name(name);
34     return 0;
35 }

```