Functions & Scopes

DISCUSSION 4

SAKIB

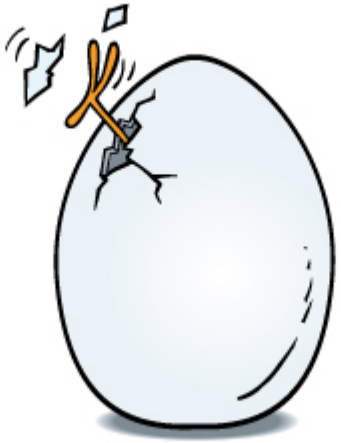
break me out

The break statement can be used to “break out” of the loop before all intended iterations

- Immediate exit

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

int main()
{
    // the following loop only iterates 5 times!!
    for (int i = 0; i < 10; i++) {
        if (i == 5)
        break;
    }
}
```



continue

Causes a “jump” directly to the end of the loop body

Then we move on to the next iteration

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

int main()
{
    int i = 0;
    // Nothing gets printed in the loop below
    while (i < 3) {
        i++;
        continue;
        cout << i << endl;
    }
}
```

continue

Be careful how you use it! You might cause infinite loops

```
int i=0;
while( i<3 ) {
    continue;
    i++;
}
```

FUNCTIONS!

Your code is just a function call away

Anywhere

Anytime

FUNCTIONS!

In math functions are written like

- f(x) = x + x^2
- f(x,y,z) = x + y*z

They

- take in a list of values (Different parameter types!)
- do some computation
- Return an output

C++ functions take these properties and expand on them!

- Can work with different types (strings, Booleans, chars, etc)

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

int power(int a, int b) {
    int pow = 1;
    for (int i = 0; i < b; i++) {
        pow *= a;
    }

    return pow;
}

int main()
{
    int res = power(4, 4);
    cout << res << endl;
}
```

Functions

If you find yourself repeating code or copy pasting it

It's a better way to go

Remember, functions take a list of parameters and you MUST indicate the types of the parameters (if you have any

You also need to specify the return values

- `int myfn(int nn, double dub, string str)`
- `double superfun(char c_u, bool cool)`
- `string no_such_thing_as_too_much_FUN()`
- `void boringfun()`

Can you Functionmatize this code??

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

int main()
{
	string mystr1 = "functions everywhere man!!";
	string mystr2 = "The Avengers is an awesome movie";
	string mystr3 = "Programmers are super kewl";

	for (int i = 0; i < 9; i++)
	cout << mystr1[i];
	cout << " ";

	for (int i = 12; i < 15; i++)
	cout << mystr3[i];
	cout << " ";

	for (int i = 19; i < 27; i++)
	cout << mystr2[i];
	cout << " ";

	cout << endl;
}
```

Example

Notice that the for loops get a substring from each of the different strings

We can write a function to take a string, a starting position, and an ending position

It would return the substring between the starting and ending positions!

You can store the results of functions into variables for later use

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

string gimmeDatStr(int i) {
     if (i == 1)
          return "DatStr";
     else
          return ">:P";
}

int main()
{
     string theStr = gimmeDatStr(1);
     string mysteryStr = gimmeDatStr(999);

     string bigStr = theStr + mysteryStr;
     cout << bigStr << endl;
}
```

Or you can start using it directly!

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

int power(int a, int b) {
    int pow = 1;
    for (int i = 0; i < b; i++) {
        pow *= a;
    }

    return pow;
}

int main()
{
    cout << power(2, 3) + power(6, 7) << endl;
}
```

Does this compile?

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

int main() {
    double x = f(5.0);

    cout << x << endl;
}

double f(double x) {
    return x + x * x;
}
```

Code is run from the first line to the last.

In the previous example, the function was not compiled until after the main.

When f is called the compiler will say something like “identifier not found”

- Function was not compiled yet

But I want to put my main function first because it makes sense for the first thing I run to be up towards the top

THERE IS A WAY

Function Prototypes are hints to the compiler that say, "Hey, here's a function with a name, a return type, and some parameters... I'm not going to define it now, but if you use it, I promise I'll have it defined later!

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

// Function prototype for f
double f(double x);

int main() {
    double x = f(5.0);

    cout << x << endl;
}

double f(double x) {
    return x + x * x;
}
```

Does this compile?

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

// Function prototype for f
double f(double x);

int main() {
    double x = f(5.0);

    cout << x << endl;
}
```

Is this legit?

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

// Function prototype for f
double f(double x);

int main() {
    string derp = "you wanted a string?";
    double x = f(derp);

    cout << x << endl;
}

double f(double x) {
    return x + x * x;
}
```

Is this legit?

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

// Function prototype for g
int g(int x);

int main() {
    cout << g(5) << endl;
    cout << g(-3.2) << endl;
    cout << g('c') << endl;
}

int g(int x) {
    return x + x * x;
}
```

Why were we able to pass a char to an int parameter?

Why could we not do the same for the string and pass it as a double??

The char is a basic primitive type. It is represented by a number

The string is a custom class. It has a much deeper implementation and functionality

Will it compile

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

string deliverGoods(string s);

int main() {
    cout << deliverGoods("My goods, please.")
    << endl;
}

string deliverGoods(string s) {
    if (s == "Got the goods?") {
        return "Yeah I got the goods.";
    }
    // Uhh... err...
}
```

Functions can call other function.

A function called by the main can call other functions!!!

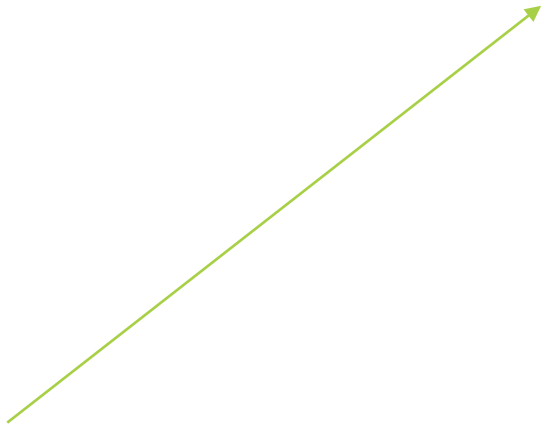
Its like a function within a function within a function

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

void happy();
void sad();
void mood(int m);

int main() {
    int in;
    cin >> in;

    mood(in);
}
```



```cpp
void mood(int m) {
    if (m == 1)
        happy();
    else
        sad();
}

void happy() {
    cout << ":D" << endl;
}

void sad() {
    cout << ":(" << endl;
}
```

Multiple return values?

By syntax functions only return one value

- (or use void if doesn't return anything)

But there are tricks to get multiple outputs from functions

Such as passing by **reference**

Think of a reference as a nickname

By default, a parameter is copied when the function is called but a reference is another name for the original variable

If you edit the parameter by reference, you edit the original variable!

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

// void type functions dont return anything
void print(int x) {
    cout << x << endl;
}

void fun(int &x) {
    x = 20;
}

int main() {
    int a = 10;

    print(a);
    fun(a);
    print(a);
}
```

Const

Parameters can be passed in as const too!

The function code will not be able to overwrite the parameters

One more thing on functions

Can I have two functions with the same name??

YES!

BUT

They must have **different input argument** types.

Better to use different names.

Will it compile?

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

void print() {
    cout << "in print that returns void" <<
    endl;
}

bool print() {
    cout << "in print that returns bool" <<
    endl;
    return true;
}

int main() {
    // your code goes here
    bool b = print();
    print();

    return 0;
}
```

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

void swap(int &a, int &b) {
    // code to swap ints
    cout << "in swap ints" << endl;
}

void swap(char &a, char &b) {
    // code to swap chars
    cout << "in swap chars" << endl;
}

int main() {
    // When you call it, it will call the
    proper function based on input
    int a = 10, b = 20;

    char c = 'a', d = 'b';
    swap(a, b);
    swap(c, d);
}
```

Another important note!!

SCOPE!

Scope

Variables have a “scope” of declaration.

Scope is limited by the {brackets} they were declared in (unless if it is a global variable)

AND Variables come into scope **AFTER** they are declared

BUT

{ A variable declared is available to the {brackets within} the brackets it is declared in }

Once the scope of the variable ends, the variable’s life ends ☹ and the memory is reclaimed

Lets look at some examples

```
if (conditional) {
// Everything in here is part of
// the "if block"
}
else {
// Everything in here is part of
// the "else block"
}
```

```
while (conditional) {
// Everything in here is part of
// the "while block"
}
```

```
int main() {
// Everything in here is part of
// the "main block," which is really
// just a "function block"
}
```

```
{
// This is just a plain ole block!
}

// This is a chip off the ole block
string chip = "-_-";
```

The general idea **in C++** is this:

First, look in the current block and see if that identifier is defined. If it is, use that definition, otherwise...

Look for a definition of that identifier in the **next block up**. If it is defined there, then use that definition. Keep looking in the next enclosing block, and if I don't find it in the global space...

I get a compile error, because that identifier isn't defined anywhere in the code.

```
int a = 0;

if(true) {
    int b = 1;

    while(true) {
        int c = 2;
        // a, b, c accessible
        // code to manipulate a, b, c
        break;
    }
    // a, b accessible
    // c NOT accessible
    // code to manipulate a, b
}
// a accessible
// b, c NOT accessible
```

Scope

The scope of a variable does not extend to function calls!

```
include <iostream>
using namespace std;

int my_function() {
    cout << a <<endl; //ERROR!!
    // The variable a is not declared in the scope of this function
}

int main() {
    int a = 0;

    my_function();
    return 0;
}
```

Global scope

Variables declared in global range are available to **ALL** functions **defined after** they are declared

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

int a = 0;

int main() {
    // can access a here
    // cant do anything with b because we havent see it yet!!
    return 0;
}

int b=0;

int my_function() {
    // can access BOTH a & b here!!!
}
```

Global scope continued

When global variables are altered, they **retian** their new values because they reside in the global scope

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

//declare functions
int fun1();
int fun2();

int a = 0;

int main() {
    // can access a here
    // cant do anything with b because we havent see it yet!!
    a = 10;
    fun1();
    fun2();
    return 0;
}

int b=0;

int fun1() {
    // can access BOTH a & b here!!!
    a = 5;
    b = 20;
}

int fun2() {
    // can access BOTH a & b here!!!
    cout << a <<endl; // prints 5
    cout << b <<endl; // prints 20
}
```

Unit Tests

Make sure individual parts of the program are working correctly to make sure all components are correct

Compare expected output with actual output. Success if both match

Assert statement takes a boolean value. If false, terminates program with error telling which line of code failed

```
#include <cassert>
```

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

int main () {
   string s1 = "Sup! ";
   string s2 = "Y'all";

   assert(s1.length() == s2.length());
   string s3 = s1+s2;

   assert(s3 != "Sup world");
   assert(s3 == "Sup! Y'all");
}
```

Next week. You'll look at arrays

What if I wanted to declare a whole bunch of integers... like 1000!

Should I declare them one at a time

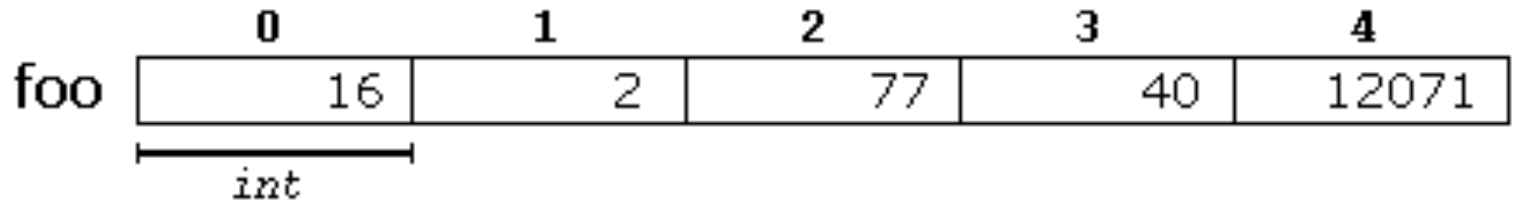
- Any sensible person would tell me I'm crazy at that point

We can declare a **list** or array of integers.

We can even declare 2D lists, or 3D, or 4D (don't get too excited)

Arrays are contiguous in memory. They are put in one after another.

```
int foo [5] = { 16, 2, 77, 40, 12071 };
```




```
// setting values
foo [2] = 75;
```

```
// getting values
int x = foo[2];
```

Proj 3 Advice

Build incrementally and test often. Implement one feature at a time

- No beats
- Regular beats
- Numbered beats etc…

Don’t be afraid to use extra variables to help you keep track of things