Module 19: Introduction to Functions

Intro to Computer Science 1 - C++

Professor Scott Frees

Textbook

The following topics are covered in sections 6.1-6.3. Please read these sections carefully!

Sum from x to y

Write a program that computes the sum of the the integers between 1 and 10.

Now also compute the sum between 20 and 37…

And between 35 and 49….

Sum from x to y

```
int sum = 0
for ( int i = 1; i <= 10; i++ )
    sum += i
cout << sum << endl;
```

```
int sum = 0
for ( int i = 20; i <= 37; i++ )
    sum += i
cout << sum << endl;
```

```
int sum = 0
for ( int i = 35; i <= 49; i++
)
    sum += i
cout << sum << endl;
```

Sum from x to y

```
int sum = 0
for ( int i = 1; i <= 10; i++ )
    sum += i
cout << sum << endl;
```

```
int sum = 0
for ( int i = 20; i <= 37; i++ )
    sum += i
cout << sum << endl;
```

```
int sum = 0
for ( int i = 35; i <= 49; i++
)
    sum += i
cout << sum << endl;
```

They are all the same, except for the start and end values of i....

Reusable Groups

A function is best thought of as a reusable group of code.

- Functions can accept parameters
- Functions can return results

We've already seen how to **call** them

```
double x = pow(2, 8);  // x will be 256
double y = sqrt(9); // y will be 3
```

Defining a new function

Functions have names - following the same rules as variables

```
int sum(int x, int y) {
    int s = 0;
    for ( int i = x; i <= y; i++ )
        s+= i;
    return s;
}
```

Defining a new function

Functions have names - following the same rules as variables

```
int sum(int x, int y) {
    int s = 0;
    for ( int i = x; i <= y; i++ )
        s += i;
    return s;
}
```

The x and y parameters will be defined when we call the actual function

The answer will be *returned* and can be used by the caller

Calling your function

```
int sum(int x, int y) {
    int s = 0;
    for ( int i = x; i <= y; i++ )
        s += i;
    return s;
}

int main() {
    int a = sum(1, 10);
    int b = sum(20, 37);
    int c = sum(35, 49);
    cout << a << " " << b << " " << c << endl;
}
```

Each time **`computeSum`** is called, x and y are different, and there is a different return value

Parameters

When calling a function, parameters can be any expression that yields the right data type

```
a = sum(5, 6);
a = sum(w, z);
a = sum(w + 1, z + w);
a = sum(sum(5, 10), sum(30, 100));
```

Its important that you see that there is nothing really special about any of these calls!

Order of declaration

For now - remember the *compiler* reads your code from the top down

Error - compiler hasn't seen this function yet!

```
int main() {
    int s = computeSum(3, 10);
}
int computeSum(int x, int y) {
    int sum = 0
    for ( int i = x; i <= y; i++ )
        sum += i
    return sum;
}
```

```
int computeSum(int x, int y) {
    int sum = 0
    for ( int i = x; i <= y; i++ )
        sum += i
    return sum;
}

int main() {
    int s = computeSum(3, 10);
}
```

OK, compiler has seen this function already

Data and functions

- **Local Variables**: Variable used *within a single function*. This data is only "visible" within that function!

- **Parameter Variables**: Communication **from** *calling function* **to** the function.

- **Return Values**: Data passed **from** the called function **back to** its caller

Programming Example 23

We've used the cmath library to calculate exponents using the pow function.

Lets write our own… so we don't need to include cmath

Note: While its nice to do things yourself while you are learning - we'd never want to implement functions that are already in cmath in the "real world"…

1. You'll have more important things (code) to do!
2. Libraries have been tested over decades - they probably will be better!