Comp 11 Lectures

Mike Shah

Tufts University

July 5, 2017

Please do not distribute or host these slides without prior permission.

Functions 2

Comp 11 - Pre-Class warm up

In bowling, if you get a strike on your first bowl, it is called a strike. You get the values of your next two balls as a bonus. So if you get a strike, then on your next bowl, if you get a 4 and a 5, you get (10+4+5) for that frame, and then the 9 for the current frame.

Lecture

Famous Lecturer TBD

Figure 1: Alonzo Church is best known for his mathematical contributions to Computer Science. He developed a lambda calculus which was considered equal in power to Alan Turings Turing Machine discoveries. Alan Turing in fact, was Church's student!

Three more ways to view functions

Today we will discuss the following

- Member Functions
- Recursion (recursive functions)
- lambda's

Where are we in the course

- We are moving onto the next section of the course. We are going to be talking about working with data.
- C++ is a general purpose language in which we can program nearly anything.
- The ideology for how C++ achieves this is known as an object-oriented(OO) language.
- That is, we create objects(as the programming language understands them), and then they can perform operations.
- We will jump head first into this in a few weeks, but this OO idea important to keep in mind.

Another look at strings

- I have a secret for you, a string is actually an object in C++.
- This is a good thing for us, because that means we can perform some operations on them.
- These operations are built into the string data structure (remember, a string is simply a sequence of characters)
- Every string that we create as a variable, we can act on it independently.

A string member - substr

```cpp
#include <iostream>
#include <string>

int main(){
  std::string fullName = "Mike Shah";
    // Here we introduce the 'substr' member function
    // It allows us to grab a sub-string,
    // (i.e. a part of a string.)
  std::string firstName = fullName.substr(0,4);
  std::string lastName = fullName.substr(5,9);

  std::cout << "First: " << firstName << "\n";
  std::cout << "Last: " << lastName << "\n";

  return 0;
}
```

Listing 1: substr member function

What we are reviewing

- There is a . (DOT) that we place after our variable name.
- This . (DOT) is an operator, similar to +,-,*,/
- However, this can be thought of as operating on the object.
- In our previous example, we are operating on an object, which is a string, called fullName.
- We are going to get use to thinking of blocks of code as objects. C++ is an Object-oriented programming language.

Another example - length of string

```cpp
#include <iostream>
#include <string>

int main(){

    std::string s1 = "12345";
    std::string s2 = "abc";
    std::string s3 = "Mike";

    std::cout << "Length of s1: " << s1.length() << "\n";
    std::cout << "Length of s2: " << s2.length() << "\n";
    std::cout << "Length of s3: " << s3.length() << "\n";

    return 0;
}
```

Listing 2: Lengths of a string

at(index) accessor

```cpp
// Add our new "string" library
#include <string>
#include <iostream>

int main(){
  std::string s = "Hello";
  std::cout << "s[0]" << s[0] << std::endl;
  std::cout << "s[1]" << s[1] << std::endl;
  std::cout << "s.at(2)" << s.at(2) << std::endl;
  std::cout << "s.at(3)" << s.at(3) << std::endl;
  std::cout << "s.at(4)" << s.at(4) << std::endl;
  // Now remember, a char is returned.
  // So we store in a char if we want to use it.
  char lastChar = s.at(4);

  // Why not the line below?
  // std::cout << "s[5]" << s[0] << std::endl;
  return 0;
}
```

Listing 3: Lengths of a string

Constructor and Destructor

- When we create our own custom data structure (using a struct), there are two special member functions given to us for free.
- They are called the constructor and the destructor.
- The constructor is the code that executes as soon as we create the object.
- The destructor is the code that executes as soon as the object is destroyed. Which means it either leaves scope, or we use the **delete** keyword.

Default Constructor

```cpp
#include <iostream>
struct Student{
    int age;
    void print(){
        std::cout << "Students age: " << age << "\n";
    }
};

int main(){
    Student s1; // Create a student (default constructor
    called)
    s1.print(); // Print whatever default constructor assigns
     age to
    return 0;
}
```

Listing 4: The C++ Constructor given to us for free

Default Constructor

```cpp
#include <iostream>
struct Student{
    int age;
    // Our default constructor redefined
    Student(){
        std::cout << "Default constructor called\n";
        age = 10;
    }
    void print(){
         std::cout << "Students age: " << age << "\n";
    }
};
int main(){
    Student s1;
    s1.print();
    return 0;
}
```

Listing 5: Defining our own C++ Constructor

Define our Destructor

```cpp
#include <iostream>
struct Student{
    int age;
    Student(){
        std::cout << "Default constructor called\n";
        age = 10;
    }
    // Define our destructor
    ~Student(){ // Note the ~ and no parameters are passed.
        std::cout << "Destructor called\n";
    }
    void print(){
        std::cout << "Students age: " << age << "\n";
    }
};
int main(){
    Student s1;    s1.print();       return 0;
}
```

Listing 6: Defining our own C++ Constructor

Recursion

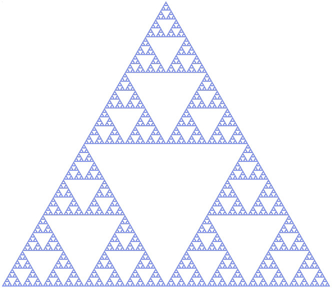
What is recursion

Recursion Defined

The repeated application of a recursive procedure or definition.

- The problem must eventually terminate to be useful in computers(i.e. have a base case).
- Can think of mathematically if that is helpful.
- Recursion is generally more concise and provides more elegant solutions.

Some Examples of Recursion



A First Recursion Example

```
#include <iostream>

int countDown(int input){
  if(input == 0){
    return 0;
  }

  std::cout << "input = " <<  input << "\n";

  return coundDown(input - 1);
}

int main(){

  coundDown(10);

  return 0;
}
```

Listing 7: First Example

Structure of a Recursive Function

```
// Recursion - a function that calls itself
// Solve a smaller subset of the problem one at a time
// Then

returnValue functionCall(Parameters){
  (1) Base Case

  (2) Computation

  return functionCall(Parameters);
}
```

Listing 8: Template for recursive functions

A Second Look at Recursion Example

```
#include <iostream>
int countDown(int input){
  // Base Case for when our countdown reaches 0
  if(input == 0){
    return 0;
  }
  std::cout << "input = " << input << "\n";
  // Not that we are subtracting 1 from our input every time.
  return coundDown(input - 1);
}
int main(){
  // Call Countdown function and check result
  coundDown(10);
  return 0;
}
```

Listing 9: Annotated sample with comments but will it always work?

A third Look at Recursion Example

```cpp
#include <iostream>
int countDown(int input){
  // Base Case for when our countdown reaches 0
  if(input <= 0){
    return 0;
  }
  std::cout << "input = " <<  input << "\n";
  // Not that we are subtracting 1 from our input every time.
  return coundDown(input - 1);
}
int main(){
  // Call Countdown function and check result
  coundDown(10);
  // Made a fix to catch a case of less than 0! (see basecase)
    coundDown(-10);
  return 0;
}
```

Listing 10: Recursion can have infinite loops and never terminate as well! (Perhaps this example would eventually terminate because of underflow however)

Recursion is a stack of operations

Table 1: First Call

countDown(10)

Recursion is a stack of operations

Table 2: Second Call, still need to return from first call, but we have made another function call

countDown(9)
countDown(10)

Recursion is a stack of operations

Table 3: Third Call, still need to return from second call, but we have made another function call

countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 4: Fourth Call, still need to return from third call, but we have made another function call

countDown(7)
countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 5: etc

countDown(6)
countDown(7)
countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 6: etc etc

countDown(5)
countDown(6)
countDown(7)
countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 7: etc etc etc

countDown(4)
countDown(5)
countDown(6)
countDown(7)
countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 8: etc etc etc etc

countDown(3)
countDown(4)
countDown(5)
countDown(6)
countDown(7)
countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 9: etc etc etc etc etc

countDown(2)
countDown(3)
countDown(4)
countDown(5)
countDown(6)
countDown(7)
countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 10: etc etc etc etc etc etc

countDown(1)
countDown(2)
countDown(3)
countDown(4)
countDown(5)
countDown(6)
countDown(7)
countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 11: Ah, finally hit our base-case and return

countDown(0)
countDown(1)
countDown(2)
countDown(3)
countDown(4)
countDown(5)
countDown(6)
countDown(7)
countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 12: Go to our previous function call and finish any work to be done

countDown(1)
countDown(2)
countDown(3)
countDown(4)
countDown(5)
countDown(6)
countDown(7)
countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 13: Continue this process

countDown(2)
countDown(3)
countDown(4)
countDown(5)
countDown(6)
countDown(7)
countDown(8)
countDown(9)
countDown(10)

Recursion is a stack of operations

Table 14: Continue this process

countDown(3)
...
...
...
...
...
...
countDown(10)

Recursion is a stack of operations

Table 15: Finally Finish

countDown(10)

Recursion - iterate through an array

```
#include <iostream>

void iterate(int* array, int size, int pos){
  if(pos==size){ // base case
      return;
    }else{
      std::cout << array[pos] << "\n";
      iterate(array,size,pos+1); // update position
    }
}

int main(){
  int myArray[] = {1,3,5,7,9};
    // Call our function
    iterate(myArray,5,0);

  return 0;
}
```

Listing 11: Using recursion for iteration

Lambdas

Bonus Material - Lambdas

- New C++11 style of defining and using functions.
- No requirement in this class to use them, but you should know they exist and at least see one.

Nameless Functions - Lambdas

- Also sometimes called anonymous functions (because of the lack of a name).
- These functions do not have a name, but they can have parameters.
- Based off of lambda calculus developed by Alonzo Church, for which functions have exactly one input parameter and one output.

Why use a lambda?

- Sometimes we want the ability to define a function on the fly.
- Generally more concise.
- Usually used to compose bigger operations (can create functions easily anywhere).
- Best motivated by an example (next slide)

C++ 11 - Lambdas

```
#include <iostream>

int main(){

auto addFunc = [](int a, int b) { return a + b; };
auto squareFunc = [](int x) { return x*x; };


std::cout << "Lambda addFunc(5,2)=" << addFunc(5,2) << "\n";
std::cout << "Lambda squareFunc(7)=" << squareFunc(7) << "\n"
    ;

    return 0;
}
```

Listing 12: Lambda function examples do not forget to set std to c++11 when compiling

In-Class Activity

`http://www.mshah.io/comp/11/activities/activity10/activity.pdf`

Activity Discussion

Review of what we learned

- (At least) Two students
- Tell me each 1 thing you learned or found interesting in lecture.

5-10 minute break

To the lab!

Lab: `http://www.mshah.io/comp/11/labs/lab9/lab.pdf`
[1]

[1] You should have gotten an e-mail and hopefully setup an account at `https://www.eecs.tufts.edu/~accounts` prior to today. If not–no worries, we'll take care of it during lab!