Class 15

Recursion

Re-cap:

Call by value

- When passing values to a function, C++ creates a copy of the values stored in the variable
- The function operates on those copies of values

Call by reference

- When you want to pass the actual variable to the function, you mark this in the title line by putting an & between the type and name of the parameter

Void Function

```
string fullName1 (string first, string last){
    string result = first + " " + last;
    return result;
}

int main(){
   string firstName = "Bob";
   string lastName = "Gallagher";
   // next line prints Bob Gallagher
   cout << fullName1(firstName, lastName);
   // next line stores result of function call in variable
   string fullName = fullName1(firstName, lastName);
   return 0;
}
```

```
void fullName2 (string first, string last){
    cout << first << " " << last;
}

int main(){
   string firstName = "Bob";
   string lastName = "Gallagher";
   // next line prints Bob Gallagher
   fullName2(firstName, lastName);
   // void functions cannot return a value that
   // can be stored in a variable
   return 0;
}
```

Void Function

```
int positiveCube(int a){
   if(a < 0) return a * a * a * -1;
   else return a * a * a;
}

int main(){
   int a, b;
   cout << “Enter two numbers: “;
   cin >> a >> b;
   // update each to store the positive cube
   a = positiveCube(a);
   b = positiveCube(b);
   cout << a << “ “ << b << endl;
   return 0;
}
```

```
void positiveCubes(int &a, int &b){
    if(a < 0) a = a * a * a * -1;
    else a = a * a * a;
    if(b < 0) b = b * b * b * -1;
    else b = b * b * b;
}

int main(){
   int a, b;
   cout << “Enter two numbers: “;
   cin >> a >> b;
   // update each to store the positive cube
   positiveCubes(a, b);
   cout << a << “ “ << b << endl;
   return 0;
}
```

Recursion

- Use a dictionary to look up an unknown word
- What if the definition in the dictionary contains a word we don’t know?
- We use the same dictionary to look up this new word
- Continue looking up unknown words until we have learned the meaning of all the unknown words

Recursion

- In a similar manner, we might have a function that solves a problem by using itself to solve a smaller version of a problem
- Recursion means “when a thing is defined in terms of itself”
- In programming, recursion happens when a function calls itself *within its own definition*
- Paradox? How can we tell C++ to perform a task by asking it to use that task?

--> the key is to ask it to use a **simpler** version of the task.

Example 1

- Factorial function

Constructing a recursive function

Recursive functions have two parts:

1. A base case, in which the function can return the result immediately
1. A recursive case, in which the function must *call itself* to break the current problem down to a simpler level

Example 2

- Given integer *n*, write function to return left-most digit

Recursion

- Recursion is a programming technique
- Pro: Sometimes it is easier to write a recursive solution than an iterative solution
- Con: Sometimes the recursive solution requires too much memory to be workable

Benefits of Recursion

- While it takes a bit of practice to easily recognize how to decompose problems into recursive formulations, it can be one of the quickest ways to design an algorithm
- A recursive version of a function can sometimes be much simpler than an iterative version

Example 3

- write_vertical
  - Writes digits of a number vertically on a screen

Example 4

- number of digits in an integer

Summery on constructing a recursive function

- A recursive function contains a call to the function being defined
- The recursive call must accomplish a smaller version of the task (“Progress Condition”)
- The function must have one or more cases in which the task is accomplished without using a recursive call (“Base Cases” or “Stopping Conditions”)