CS 161, Lecture 10: Detailed Functions – 2

February 2018

Complete Binary Conversion Program

```cpp
 1 #include <iostream>
 2 #include <string>
 3 #include <cmath>
 4
 5 using namespace std;
 6
 7 bool check_bin(string bin) {
 8         for(int i=0; i < bin.length(); i++) {
 9                 if ((bin[i] != '0') && (bin[i] != '1')) {
10                         return false;
11                 }
12         }
13         return true;
14 }
15
16 string get_bin() {
17
18         string bin = "";
19
20         do {
21                 cout << "Please provide a binary number: ";
22                 getline(cin, bin);
23         } while (check_bin(bin) == false);
24
```

24,0-1   Top

```
16 string get_bin() {
17
18          string bin = "";
19
20          do {
21                  cout << "Please provide a binary number: ";
22                  getline(cin, bin);
23          } while (check_bin(bin) == false);
24
25          return bin;
26 }
27
28 void unit_test_check_bin() {
29          cout << "Testing check_bin()" << endl;
30          cout << "Input: \"0110\", Expected Output: True, Actual Output: "
   ;
31          if (check_bin("0110")) {
32                  cout << "True , PASS" << endl;
33          }
34          else {
35                  cout << "False, FAIL" << endl;
36          }
37          cout << "Input: \"a110\", Expected Output: False, Actual Output:
   ";
```

24,0-1   35%

```cpp
28 void unit_test_check_bin() {
29         cout << "Testing check_bin()" << endl;
30         cout << "Input: \"0110\", Expected Output: True, Actual Output: "
;
31         if (check_bin("0110")) {
32                 cout << "True , PASS" << endl;
33         }
34         else {
35                 cout << "False, FAIL" << endl;
36         }
37         cout << "Input: \"a110\", Expected Output: False, Actual Output:
";
38         if (check_bin("a110")) {
39                 cout << "True, FAIL" << endl;
40         }
41         else {
42                 cout << "False, PASS" << endl;
43         }
44 }
45
46 int convert_to_decimal(string bin) {
47         int len = bin.length();
48         int res = 0;
49         for(int i=0; i < len; i++) {
```

31,1-8   64%

```cpp
46 int convert_to_decimal(string bin) {
47         int len = bin.length();
48         int res = 0;
49         for(int i=0; i < len; i++) {
50                 if (bin[i] == '1') {
51                         res += pow(2, (len-i-1));
52                 }
53         }
54         return res;
55 }
56
57
58 int main() {
59         unit_test_check_bin();
60         string bin = get_bin();
61         int res = convert_to_decimal(bin);
62         cout << res << endl;
63         return 0;
64 }
~
~
~
~
~
~
```

Type here to search   9:20 AM
                                                             2/1/2018

Function Scope

- Similar to conditionals and loops, functions have a defined scope
- Local variables: variables declared in a scope only exist with in that scope therefore are of limited accessibility
- Global variables: variables declared in the global scope and therefore accessible by everything

Scope Issues: what is wrong here?

```
void compute_sum();
int main () {
        int x=2, y=3;
        compute_sum();
        sum = x+y;
        return 0;
}
void compute_sum() {
        int sum = x+y;
}
```

Default Arguments

- If an argument is omitted it can be replaced with a default argument
- Only use for call-by-value parameters (more details on week 6 but this is what we are currently doing)
- Default arguments are defined the first time the function is declared or defined, must be in rightmost position

int sum (int a, int b = 1);

int sum (int a, int b) {

      return a+b;

}

- Assumes omission of rightmost argument

Overloading Functions

- Two or more function definitions for the same name
- Ex:

```
int sum (int a, int b) {
        return a+b;
}
int sum (int a, int b, int c) {
        return a+b+c;
}
```

- In order for the compiler to know the difference:
  - Need different amount of parameters
  - Need different data types on the parameters
  - Different return values will NOT be enough

Overloading Functions Continued

- Compiler decides based on the following:
  - Exact match: if the number and types of arguments exactly match a definition (without any automatic type conversion), then that is the definition used
  - Match using automatic type conversion: if there is no exact match but there is using automatic type conversion, then the match is used
  - Ambiguity:

  void f (int n, double m);

  void f (double n, int m);

  f (98, 99);

  *Savitch, Walter. *Absolute C W/ MyProgrammingLab, 5th Edition* . 5th ed. Boston, MA: Addison-Wesley , 2012.

Demo