Lab # 14: Functions - IV

EC-102 – Computer Systems and Programming

Usman Ayub Sheikh

School of Mechanical and Manufacturing Engineering (SMME),
National University of Sciences and Technology (NUST)

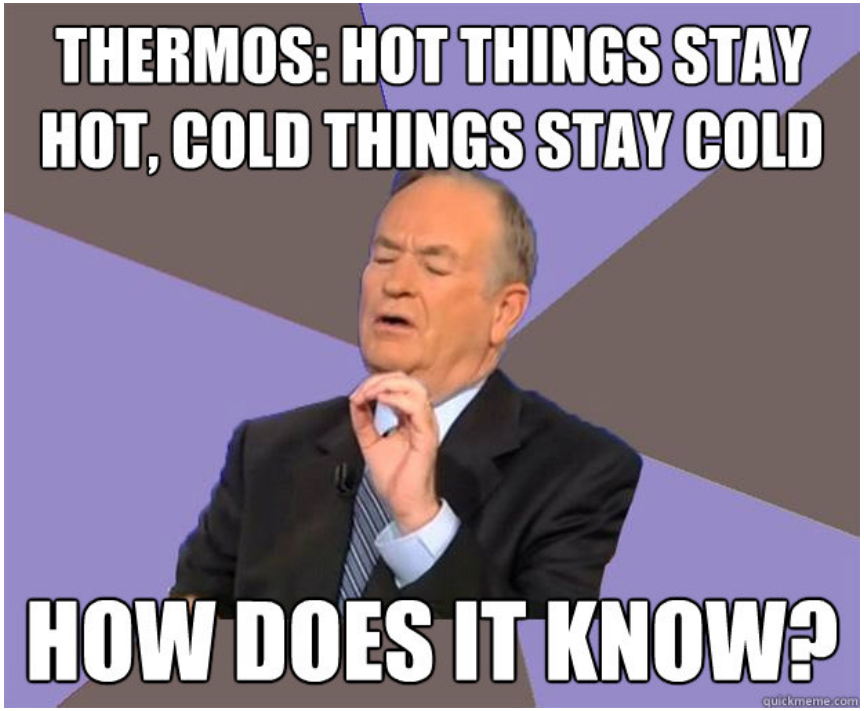
Monday 23rd November, 2015

Outline

Overloaded Functions

A function that

- Appears to perform different activities depending on the kind of data sent to it
- Performs one operation on one kind of data but another operation on a different kind



Different Number of Arguments

```
1  // demonstrates function overloading
2  #include <iostream>
3  using namespace std;
4
5  void repchar();
6  void repchar(char);
7  void repchar(char, int);
8
9  int main()
10 {
11     repchar();
12     repchar('-');
13     repchar('~', 10);
14 }
```

Different Number of Arguments

```
15 // displays 45 asterisks
16 void repchar()
17 {
18     for(int i = 0; i < 45; i++)
19     {
20         cout << '*';
21     }
22     cout << endl;
23 }
24
25 // displays 45 copies of a specified character
26 void repchar(char ch)
27 {
28     for(int i = 0; i < 45; i++)
29     {
30         cout << ch;
31     }
32     cout << endl;
33 }
```

Different Number of Arguments

```
34 // displays specified number of copies of a specified character
35 void repchar(char ch, int n)
36 {
37     for(int i = 0; i < n; i++)
38     {
39         cout << ch;
40     }
41     cout << endl;
42 }
```

Different Kinds of Arguments

```
1  // demonstrates overloaded functions
2  #include <iostream>
3  using namespace std;
4
5  struct Distance
6  {
7      int feet;
8      float inches;
9  };
10
11 void engldisp(Distance);
12 void engldisp(float);
```

Different Kinds of Arguments

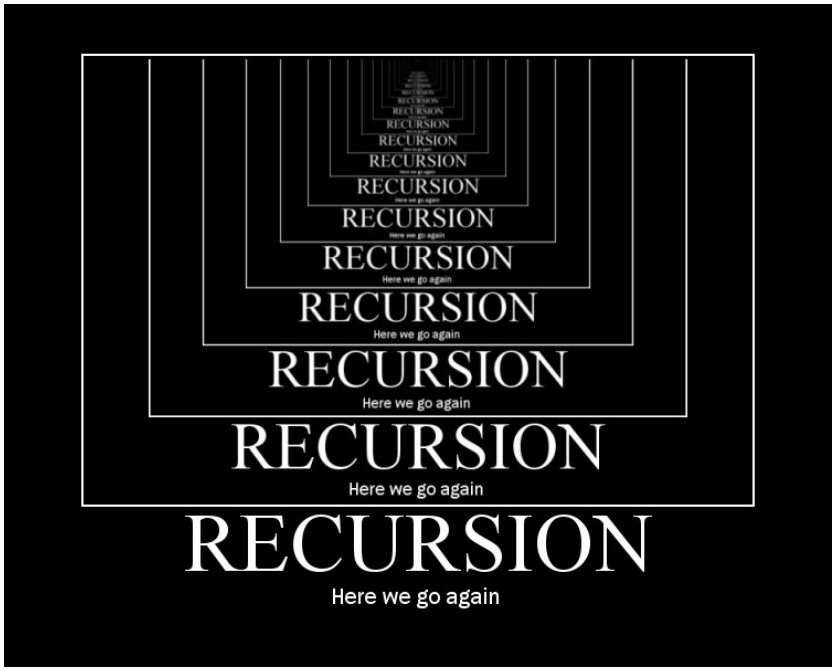
```
14 int main()
15 {
16     Distance d1;
17     float d2;
18
19     cout << "Enter feet: "; cin >> d1.feet;
20     cout << "Enter inches: "; cin >> d1.inches;
21
22     cout << "Enter entire distance in inches: "; cin >> d2;
23
24     cout << "\nd1 = ";
25     engldisp(d1);
26     cout << "\nd2 = ";
27     engldisp(d2);
28     cout << endl;
29     return 0;
30 }
```

Different Kinds of Arguments

```
31 // displays structure of type Distance in feet and inches
32 void engldisp(Distance dd)
33 {
34     cout << dd.feet << "\'-" << dd.inches << "\"";
35 }
36
37 // display variable of type float in feet and inches
38 void engldisp(float dd)
39 {
40     int feet = dd / 12;
41     float inches = dd - feet * 12;
42     cout << feet << "\'-" << inches << "\"";
43 }
```

Recursion

- A function calling itself
- When used correctly, this technique can be surprisingly powerful



How to Use Recursion?

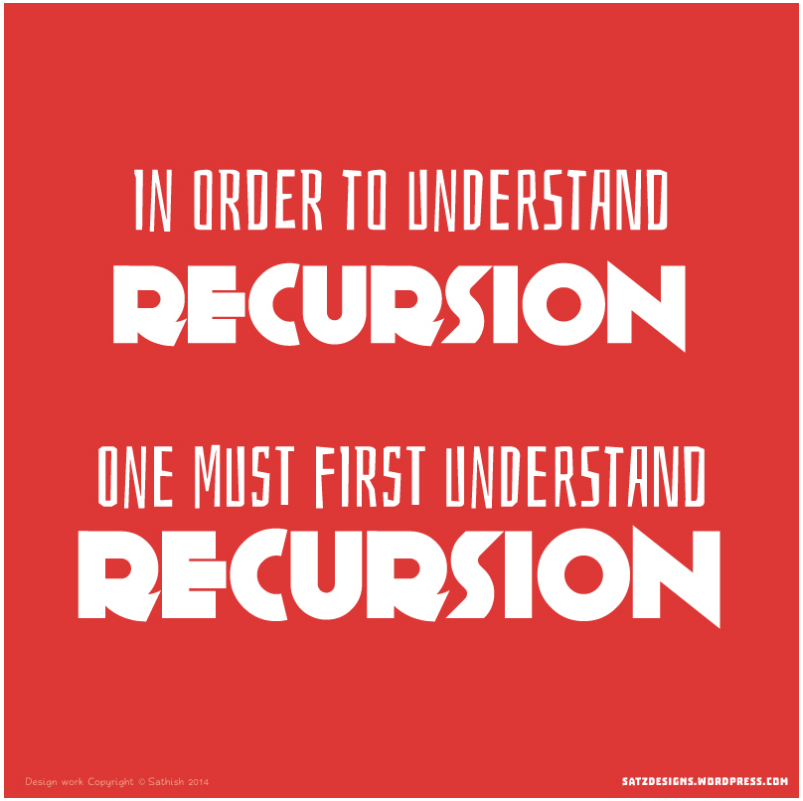
```
1  // calculates factorials using recursion
2  #include <iostream>
3  using namespace std;
4
5  unsigned long factfunc(unsigned long);
6
7  int main()
8  {
9      int n;
10     unsigned long fact;
11
12     cout << "Enter an integer: "; cin >> n;
13     fact = factfunc(n);
14     cout << "Factorial of " << n << " is: " << fact << endl;
15     return 0;
16 }
```

How to Use Recursion?

```
19 unsigned long factfunc(unsigned long n)
20 {
21     if(n > 1)
22     {
23         return n * factfunc(n - 1);
24     }
25     else
26     {
27         return 1;
28     }
29 }
```

How to Use Recursion?

- Every recursive function must be provided with a way to end the recursion
- Otherwise it will call itself forever and crash the program



Default Arguments

- A function can also be called without specifying all of its arguments
- But this won't work on just any function
- The function declaration must provide default values for those arguments that are not specified

How to Use Default Arguments?

```
1  // demonstrates missing and default arguments
2  #include <iostream>
3  using namespace std;
4
5  void repchar(char = '*', int = 45);
6
7  int main()
8  {
9      repchar();
10     repchar('+');
11     repchar('^', 30);
12     return 0;
13 }
```

How to Use Default Arguments?

```
14 void repchar(char ch, int n)
15 {
16     for(int j = 0; j < n; j++)
17     {
18         cout << ch;
19     }
20     cout << endl;
21 }
```

Why Use Default Arguments?

- They are useful if you don't want to to go through the trouble of writing arguments that have the same value most of the time
- They are also useful if the programmer decides to increase the capability of a function by adding another argument

Exercise 1

Create a series of **overloaded functions** named as `firstn` such that

- if `firstn` is called without any argument, first 10 numbers divisible by 2 are printed,
- if `firstn` is called with one argument, first 10 numbers divisible by the specified number are printed, and
- if `firstn` is called with two arguments, first m numbers divisible by n are printed.

Exercise 2

Create a function named as `firstn` using **default arguments** such that

- if `firstn` is called without any argument, first 10 numbers divisible by 2 are printed,
- if `firstn` is called with one argument, first 10 numbers divisible by the specified number are printed, and
- if `firstn` is called with two arguments, first m numbers divisible by n are printed.

Exercise 3

Create a function named as `backward_counting` using **recursion** that

- Takes an integer as an argument, and
- Prints out backward counting starting at the specified number

Exercise 4

Part 1:

Create a function named as `fibonacci` using **recursion** that

- Takes a number `n` as an argument, and
- returns n^{th} Fibonacci number

Fibonacci sequence goes as follows: `1, 1, 2, 3, 5, 8, 13, ...`

The function should work as follows:
`fibonacci(0)` and fibonacci(1) should equal 1,
`fibonacci(5)` should equal 8, and so on.

Part 2:

Write a program that

- exercises this functions by obtaining a number `x` from the user, and
- printing out first `x` numbers in the Fibonacci sequence