Unit 11a

Function Sequencing
Pass-by-Value
Functions Overview

- Functions (aka procedures, subroutines, or methods) are the unit of code decomposition and abstraction
  - **Decomposition**: Breaking programs into smaller units of code
  - **Abstraction**: Generalizing an action or concept without specifying how the details are implemented
Function Signatures/Prototypes

• We think of a function as a blackbox (don't know how it does the task internally) where we can provide inputs and get back a value

• A function has:
  – A name
  – Zero or more input parameters
  – 0 or 1 return (output) values
    • We only specify the type

• The signature (or prototype) of a function specifies these aspects so others know how to "call" the function
Why Functions? (1)

Desired Program Output:

```c++
#include <iostream>
using namespace std;

int main()
{
    // Print flag of 3 rows
    for(int i=0; i < 3; i++){
        for(int k=0; k < 3-i; k++){
            cout << '/';
        }
        cout << endl;
    }
    // Print flag of 5 rows
    for(int i=0; i < 5; i++){
        for(int k=0; k < 5-i; k++){
            cout << '/';
        }
        cout << endl;
    }
    return 0;
}
```

- Functions are best used to perform code that would otherwise have to be duplicated.
- By "factoring" common code into its own function and possibly parameterizing it, we can make flexible, reusable blocks of code.
Why Functions? (2)

Desired Program Output:

```cpp
#include <iostream>
using namespace std;

void printFlag(int rows);

int main()
{
    printFlag(3);
    printFlag(5);
    return 0;
}

void printFlag(int rows)
{
    for(int i=0; i < rows; i++){
        for(int k=0; k < rows-i; k++){
            cout << '/';
        }
        cout << endl;
    }
}
```

- Here we have factored the common code into its own function parameterized based on how many rows are desired.
Functions Calling Functions

- We could create 1 or 2 functions to do this job (probably could all be done in `printFlag` but we want you to see how one function can call another)
- Anytime a function calls another, the caller pauses and the called function begins
- When a function ends it returns to the previous function (the one that called it)

Program Output:

```
///
//
/
/////
////////
////////
///
///
```
Argument Passing (Pass-by-Value)

• Passing an argument to a function makes a copy of the argument

• It is like e-mailing an attached document
  – You still have the original on your PC
  – The recipient has a copy which she can modify but it will not be reflected in your version

• Communication is essentially one-way
  – Caller communicates arguments to callee, but callee cannot communicate back because she is working on copies...
  – The only communication back to the caller is via a return value.
Pass by Value (1)

• **Fact**: Function arguments/parameters act like local variables to that function
  - They only as long as the function is executing and then get deallocated.

• When arguments are passed a **copy** of the actual argument value (e.g. 3) is given to the function's input argument
  - So the function is operating on a copy and that copy only lives as long as the function

```cpp
void dec(int); int main()
{
    int y = 3;
    dec(y);
    cout << y << endl;
    return 0;
} void dec(int y)
{
    y--;
}
```

```cpp
void dec(int); int main()
{
    int y = 3;
    dec(y);
    cout << y << endl;
    return 0;
} void dec(int y)
{
    y--;
}
```
Pass by Value (2)

- Wait! But they have the same name, 'y'
  - What's in a name...Each function is a separate entity and so two 'y' variables exist (one in main and one in decrement it)
  - The only way to communicate back to main is via return
  - Try to change the code appropriately

- **Main Point:** Each function is a completely separate "sandbox" (i.e. is isolated from other functions and their data) and copies of data are passed and returned between them

```cpp
#include <iostream>

void dec(int y)
{
    y--;
}

int main()
{
    int y = 3;
    dec(y);
    std::cout << y << std::endl;
    return 0;
}
```
Pass by Value Solution

• Wait! But they have the same name, 'y'
  — What's in a name...Each function is a separate entity and so two 'y' variables exist (one in main and one in decrement it)
  — The only way to communicate back to main is via return
  — Try to change the code appropriately

• **Main Point**: Each function is a completely separate "sandbox" (i.e. is isolated from other functions and their data) and copies of data are passed and returned between them
Common Function Call Mistakes

• Problem 1: Don't list return type when you call a function. It will substitute the return value in place of call
• Problem 2: Need to save return value

```cpp
int dec(int val); // prototype
int main()
{
    int y = 3;
    int dec(y); // y = dec(y);
    cout << y << endl;
    return 0;
}
int dec(int val)
{
    val--; // return val;
    return val;
}
```

• Problem 1: Don't relist the type of the argument when you make the call
• Problem 2: Need to pass a variable that exists in the calling function

```cpp
int dec(int val); // prototype
int main()
{
    int y = 3;
    y = dec(int val); // y = dec(y);
    cout << y << endl;
    return 0;
}
int dec(int val)
{
    val--; // return val;
    return val;
}
```
Tracing Exercise 1

- **Order It**: The numbers below represent lines of code in the program to the right. Order the lines of code from 1-8 [1=first to get executed / 8=last to be executed]. Note: Order them when they START execution not when they finish execution.

  - ____ 5
  - ____ 6
  - ____ 7
  - ____ 8
  - ____ 13
  - ____ 20
  - ____ 26
  - ____ 27

- **Scope It**: Notice that there are many 'char c' declarations and parameters. List the value of the character 'c' (or N/A if no possible value for 'c' exists) just BEFORE the following lines of code get executed:

  - ____ 6
  - ____ 8
  - ____ 13
  - ____ 14
  - ____ 20
  - ____ 21
  - ____ 27

```c
// Assume necessary prototypes have // been declared for Q2 and Q3
1 int main()
2 {
3   double f=1.0, g=2.0, h=3.0;
4   int x=5, y=6, z =7;
5   char c='U';
6   x = myfunc();
7   f = doit(x,y,c);
8 }
9
double doit(int him, int her, char c)
10 {
11   double x = (double) him;
12   c = 'S';
13   x = x / (double) her;
14   return x;
15 }
16
void yourfunc(char c)
17 {
18   c = 'C';
19   return;
20 }
21
int myfunc()
22 {
23   yourfunc('!');
24   return 2;
25 }
26
```
Functions As Independent Units (1)

- Functions should be self contained units
  - Cannot access variables from other functions
  - Should implement a general "recipe" for how to do a task given generic inputs (parameter names)
  - Consider the arguments/parameters as generic names that represent specific inputs when the function is invoked

Reminders:
- Functions can only access:
  - The input arguments
  - The local variables they declare
- Functions can only return one value
  - All other values (local variables & input arguments) "die" at the end of the function

// Doesn't work
#include <iostream>
using namespace std;
void getInput(); // prototype
void getInput()
{
    cout << "Enter an int: ";
    cin >> x; // Can't access // x from main
}
int main()
{
    int x; // get from user
    getInput();
    cout << x << endl;
    return 0;
}

// Does work
#include <iostream>
using namespace std;
int getInput(); // prototype
int getInput()
{
    int num;
    cout << "Enter an int: ";
    cin >> num;
    return num;
}
int main()
{
    int x; // get from user
    x = getInput();
    cout << x << endl;
    return 0;
}
Functions As Independent Units (2)

• Generally don't do input/output in a function (unless the function specifically indicates it should)...just process values and return something
  – Example: Take the maximum of 2 numbers and print them out
  – Easy to extend to other tasks: take max of 3 numbers

```cpp
#include <iostream>
using namespace std;

int max(int a, int b);

int main()
{
    int x, y, mx;
    cin >> x >> y;
    cout << max(x,y) << endl;
    return 0;
}

int max(int a, int b)
{
    if(a > b)
        return a;
    else
        return b;
}
```

Could you extend to take the max of 4?
Functions As Independent Units (3)

- Generally don't do input/output in a function (unless the function specifically indicates it will)...just process values and return something
  - Example: Take the maximum of 2 numbers and print them out
  - Easy to extend to other tasks: take max of 3 numbers

```c++
// Bad function decomposition
// -- performs I/O.
#include <iostream>
using namespace std;

int max();

int main()
{
    int mx;
    mx = max();
    cout << mx << endl;
    return 0;
}

int max()
{
    int x, y;
    cin >> x >> y;
    if(x > y)
        return x;
    else
        return y;
}
```

This code works for the stated task (output max of 2 inputs) but cannot easily be reused for 3 or more numbers! [Can't do: `max(max());`]
- Would `cin` 4 numbers
- Doesn't take in an input
Exercises

• Exercises
  – extract-method1
  – is-lower-vowel
  – is-vowel
  – draw-square
Tracing Exercise 1 (Solution)

- **Order It**: The numbers below represent lines of code in the program to the right. Order the lines of code from 1-8 [1=first to get executed / 8=last to be execute]. Note: Order them when they START execution not when they finish execution.
  - __1__ 5
  - __2__ 6
  - __6__ 7
  - __8__ 8
  - __7__ 13
  - __4__ 20
  - __3__ 26
  - __5__ 27

- **Scope It**: Notice that there are many 'char c' declarations and parameters. List the value of the character 'c' (or N/A if no possible value for 'c' exists) just BEFORE the following lines of code get executed:
  - __'U'__ 6
  - __'U'__ 8
  - __'U'__ 13
  - __'S'__ 14
  - __'!'__ 20
  - __'C'__ 21
  - __N/A__ 27

```c
// Assume necessary prototypes have // been declared for Q2 and Q3
1  int main()
2   {
3     double f=1.0, g=2.0, h=3.0;
4     int x=5, y=6, z =7;
5     char c='U';
6     x = myfunc();
7     f=dooit(x,y,c);
8   }
9
dooit(int him, int her, char c)
10  {
11     double x = (double) him;
12     c = 'S';
13     x = x / (double) her;
14     return x;
15  }
16
yourfunc(char c)
17  {
18     c = 'C';
19     return;
20  }
21
myfunc()
22  {
23     yourfunc('!');
24     return 2;
25  }
26
// Assume necessary prototypes have // been declared for Q2 and Q3
```