Unit 4a

Calling and Using Functions
Unit 4

• **Unit 1**: Scalar processing
  – aka IPO=Input-Process-Output Programs

• **Unit 2**: Linear (1D) Processing

• **Unit 3**: Multidimensional Processing

• **Unit 4**: Divide & Conquer
  (Functional Decomposition)
Functional Decomposition Overview

- **Idea**: Extract *common* (small) code sequence into separate blocks (aka functions, procedures, subroutines, or methods) that we can "call" from anywhere in our code.

- By decomposing our software into functions, we can:
  - Reduce coding effort
  - Reuse code
  - Increase maintainability
  - Increase readability (the name of a function is often a "comment" about what that function's code does)
  - Build up large solutions from smaller pieces

```c++
int main() {
    // setup array with data
    int n, val, data[100];
    cin >> n;
    for (int i=0; i < n; i++)
        { cin >> data[i]; }
    bool found100 = false, found0 = false;
    // Find 100
    for (int i=0; i < n; i++) {
        if (100 == data[i]){
            found100 = true;
            break;
        }
    }
    // Find 0
    for (int i=0; i < n; i++) {
        if (0 == data[i]){
            found0 = true;
            break;
        }
    }
    cout << "found 100: " << found100 << endl;
    cout << "found 0: " << found0 << endl;
    return 0;
}
```
Functional Decomposition Overview

**Idea:** Extract *common* (small) code sequence into separate blocks (aka functions, procedures, subroutines, or methods) that we can "call" from anywhere in our code

By decomposing our software into functions, we can:

- Reduce coding effort
- Reuse code
- Increase maintainability
- Increase readability (the name of a function is often a "comment" about what that function's code does
- Build up large solutions from smaller pieces

```cpp
bool find(int d[], int len, int v) {
    for(int i=0; i < len; i++) {
        if(v == d[i]) { return true; }
    }
    return false;
}

int main() {
    // setup array with data
    int n, val, data[100];
    cin >> n;
    for(int i=0; i < n; i++)
        { cin >> data[i]; }
    bool found100 = false, found0 = false;
    // Find 100
    found100 = find(data, n, 100);
    // Find 0
    found0 = find(data, n, 0);
    cout << "found 100: " << found100 << endl;
    cout << "found 0: " << found0 << endl;
    return 0;
}
```
Functions Overview

• Functions (aka procedures, procedures, 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
Recall: Walking a Square in Scratch

- We can define a function (i.e. block of code) once and then "call" it any time we want to execute that block of code.
- Can provide different input values (aka "arguments" / "parameters") and even get an output (aka "return" value).
Function Signatures/Prototypes

• We think of a function as a blackbox (don't know or care how it does the task internally) of code where **we can provide inputs and get back a value**
  - Or think of it as a web-app (or form) where you supply data to "named" inputs and get back a value

• In C/C++, a function has:
  - A **name**
  - Zero or more input parameters
  - 0 or 1 return (output) values
    - We only specify the type
    - 0 return values is indicated with void type
  - The signature (or **prototype**) of a function specifies these aspects so others know how to "call" the function

```c
int max(int a, int b);
```
Common Functions

**pow**

### Function Signature/Prototype

```c
double pow (double base, double exp);
```

### Raise to power

Returns *base* raised to the power *exponent*:

\[ base^{exp} \]

**rand**

### Function Signature/Prototype

```c
int rand (void);
```

**Generate random number**

Returns a pseudo-random integral number in the range between 0 and `RAND_MAX`.

This number is generated by an algorithm that returns a sequence of apparently non-related numbers each time it is called. This algorithm uses a seed to generate the series, which should be initialized to some distinctive value using function `srand`.

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.
"Functional" Programming

• While we can write arithmetic expressions directly in C++, let's practice using functions to perform the same operations.

• Suppose you are given:
  – int add(int p, int q); // returns p+q
  – int sub(int p, int q); // returns p-q
  – int mul(int p, int q); // returns p*q
  – int div(int p, int q); // returns p/q

• Convert the following expressions to use functions and no operators (+, -, *, /)

• Key Ideas:
  – Execution works from inside to outside (i.e. f(g(x)) invokes g(x) first)
  – The return value of a function is substituted and used in the larger expression

// Add 3 numbers
a = x + y + z;
// which upholds the order of ops
a = add(add(x,y),z);
a = add(x,add(y,z));

// Exercise 1
a = x / y + y * z - x;
a = ________________________________

// Exercise 2
a = x * (y - z) / z;
a = ________________________________

Disclaimer: These functions (add, sub, etc.) are fictitious and in C++ we just use the +, -, etc. operators, but this is to practice using functions.
Function call statements

• Reminder that you can call a function anywhere
• Result is replaced into bigger expression
• Take care to "save" the result
  – If you don't save the return value into a variable or use it immediately, the result is lost

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

int main()
{
    // can call functions
    // in an assignment
    double res = cos(0); // res = 1.0

    // can call functions in an
    // expression
    sqrt(2) / 2; // forgot to save result

    res = sqrt(2) / 2; // save 1.414/2 in res

    cout << max(34, 56) << endl; // outputs 56

    return 0;
}
```

http://www.cplusplus.com/reference/cmath/
Reading Documentation

• Much of programming is calling other library functions which do small pieces of work in an effort to accomplish the overall application
  – Learn to read documentation

• Documentation at:
SOLUTIONS
"Functional" Programming

- While we can write arithmetic expressions directly in C++, let's practice using functions to perform the same operations.
- Suppose you are given:
  - `add(p, q)` // returns `p+q`
  - `sub(p, q)` // returns `p-q`
  - `mul(p, q)` // returns `p*q`
  - `div(p, q)` // returns `p/q`
- Convert the following expressions to use functions and no operators (+, -, *, /)
- Key Ideas:
  - Execution works from inside to outside (i.e. `f(g(x))` invokes `g(x)` first)
  - The return value of a function is substituted and used in the larger expression

```
// Add 3 numbers
a = x + y + z;
// which upholds the order of ops
a = add(add(x,y),z);
a = add(x,add(y,z));

// Exercise 1
a = x / y + y * z - x;
a = sub(add(div(x,y),mul(y,z)),x);

// Exercise 2
a = x * (y - z) / z;
a = div(mul(x, sub(y,z)), z);
```

Disclaimer: These functions (add, sub, etc.) are fictitious and in C++ we just use the +, -, etc. operators, but this is to practice using functions.