CS 161
Intro to CS I

Decomposition/Begin Functions
Odds and Ends...

• Exam I – Friday, 10/20
• Keep working on Assignment #3!!!
  – Design due Sunday on Canvas!!!

• Don’t miss Demo, and be patient waiting
• READ, READ, READ!!!
• Ask TA pointed questions
• Just THINK! KISS😊
Programming Errors

• Syntax errors
  – Misuse of C++ language
  – How are they caught?

• Logic errors
  – Doesn’t perform task correctly (aka. bugs)
  – How are they caught?

• Runtime errors
  – Stops your program from running
  – How are they caught?
Syntax Error Examples

- Missing main function
- Use of identifier not declared
- Misspelled Words
- Forget a Semicolon
- Forget Required Keyword
- Missing quote, curly brace, and parenthesis
- Use of single quotes instead of double
Logic Error Examples

• Poorly written programs
  – Add instead of subtract (incorrect operation)
  – Using last two digits for date
  – Same error message for different errors
  – Program that never ends
  – Add one to the largest integer (could be syntax)
Runtime Error Examples

• Segmentation fault or Core dump
  – Read a file that doesn’t exist
  – Go outside of memory bounds
  – Infinite loop that eats memory
  – Divide by variable that is zero
Debugging Errors

• Syntax:
  – READ compiler errors (pay attention to line #)
  – Use google to search for error

• Logic/Runtime
  – Use std::cout to find where the code is breaking
    • Print variable values
    • Print indicator messages
  – Trace through the code
  – Comment out code
Demo...
Decomposition

• Divide Problem (task) Into Subtasks
  – Procedural Decomposition
  – Examples: cooking, cleaning, etc.

• Incremental Programming
  – Iterative Enhancement (Stepwise Refinement)

• Examples: Replicating Code
Functions

• What is a function?
  – Block of code to perform action/subroutine

• When have we seen functions already?
  – Predefined

• What is the purpose?
  – Reduce
  – Reuse
  – Readability
Predefined Functions

- sqrt()
- pow()
- abs()
- rand()
- srand()

What is the difference b/w srand() and others?
Procedural Decomposition

• Functions
  – int main() {
  
  – User defined
    void draw_box() {
    
• Function Call
  – draw_box();
#include <iostream>
using std::cout;

int main() {
    cout << "+--------+
    cout << "|       |
    cout << "+--------+
    cout << "|       |
    cout << "+--------+
    cout << "|       |
    return 0;
}

void draw_box() {
    cout << "+--------+
    cout << "|       |
    cout << "+--------+
    cout << "|       |
    cout << "+--------+
    cout << "|       |
}

#include <iostream>
using std::cout;

void draw_box();  //Declare function
int main() {
    draw_box();  //Use function
draw_box();
    return 0;
}
void draw_box() {  //Define function
cout << "+--------+
    cout << "|       |
    cout << "+--------+
}
# Functions Calling Other Functions

```cpp
#include <iostream>

void draw_box();
void draw_top_bottom();
void draw_sides();

int main() {
    draw_box();
    return 0;
}

void draw_box() {
    draw_top_bottom();
    draw_sides();
    draw_top_bottom();
}

void draw_top_bottom() {
    std::cout << "+--------+
    
}

void draw_sides() {
    std::cout << "|           |
    
}
```
Generalization

• Does a function make a task more specific or more general?
  – Justification
  – Examples
void Functions

• Doesn’t return a value
• Still has arguments/parameters
Programming Demo
Scope (Visibility)

- Part of program in which a declaration is valid
- Local variable
  - Declared inside a function only accessible inside function
- Localizing variables
  - Declaring variable in innermost scope
Illegal access outside loops

```cpp
for(x = 0; x < 10; x++) {
    int y = 10;
    cout << “The value of x * y is: ” << x*y << endl;
}
cout << “The value of y is: ” << y << endl; /*y outside scope*/
```

• How do we fix this?
• What about if/else blocks?
Illegal access in functions

```c
int main () {
    int x=2, y=3;
    compute_sum();
    sum = x+y; //error: sum hasn’t been declared
    return 0;
}

void compute_sum() {
    int sum = x+y; //error: x and y outside scope
}
```
Back to **break, exit, and return**

- **break** – used with switch and loops, breaking out of the closest associated case or loop (for, while, or do while). **This statement can only occur in a loop or case**, otherwise the compiler yells!

- **return** – leave the current function, which exits the program when in the main() function. You can put this **anywhere inside any function**, otherwise the compiler yells!

- **exit()** – exit the entire program, no matter where this is encountered. You can put this **anywhere inside any function**, **as long as you include** `<cstdlib>`, otherwise the compiler yells!
Demo...