MORE FUNCTIONS
MAKEFILES
RUNTIME STACK

Problem Solving with Computers-I

How difficult do you find the course so far?

A. Too easy
B. Easy, I sail through the labs with little effort
C. Moderately easy/difficult – I have to apply concepts and can complete the labs and homeworks with moderate effort
D. I understand the material but my partner does everything --- I don't really have the confidence to code.
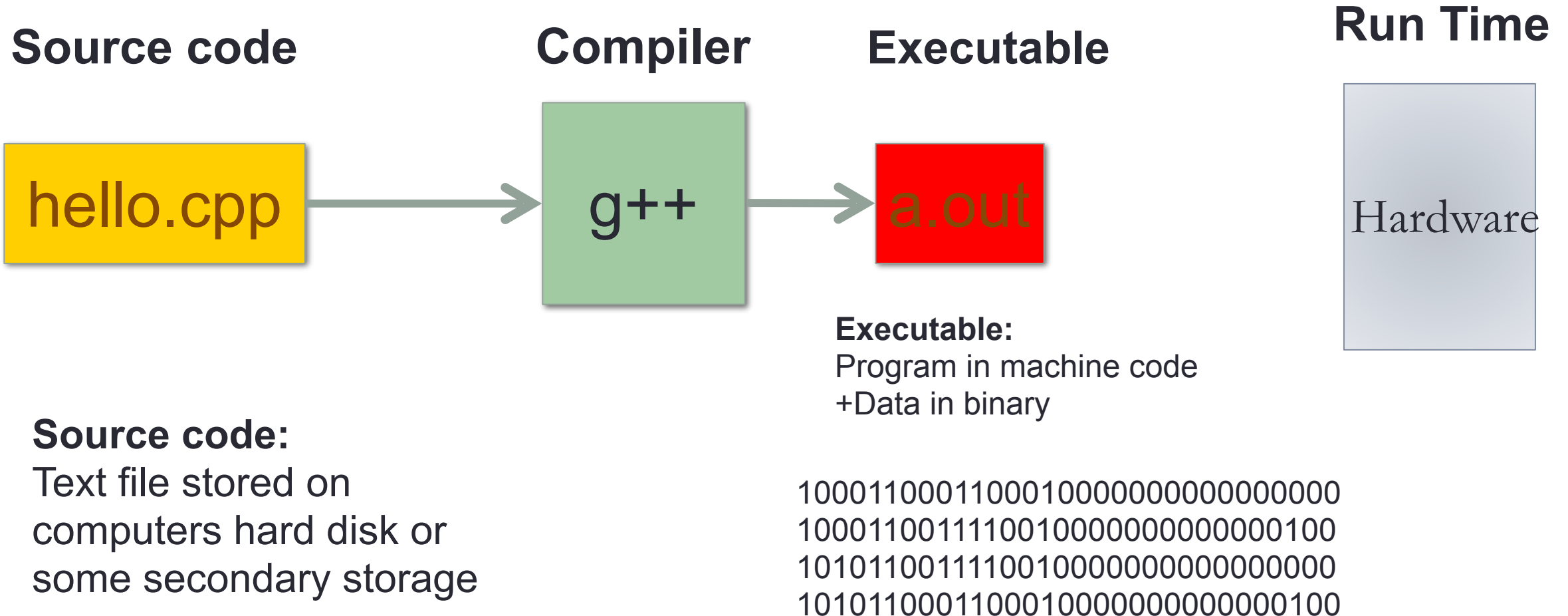E. I am really struggling and feel underprepared for this class

Writing code that works - its not magic :)

Write a function that RETURNS a string representing

an isosceles triangle with a given width

```
s = drawTriangle(5);
cout<<s;

  *
 ***
*****
```

The compilation process

Source code   **Compiler**   **Executable**   **Run Time**

hello.cpp   →   g++   →   a.out

Hardware

Executable:
Program in machine code
+Data in binary

Source code:
Text file stored on computers hard disk or some secondary storage

```
1000110011001000000000000000
1000110011110010000000000000100
1010110011110010000000000000000
1010110011001000000000000000100
```

g++ is composed of a number of smaller programs

- Code written by others (libraries) can be included
- ld (linkage editor) merges one or more object files with the relevant libraries to produce a single executable
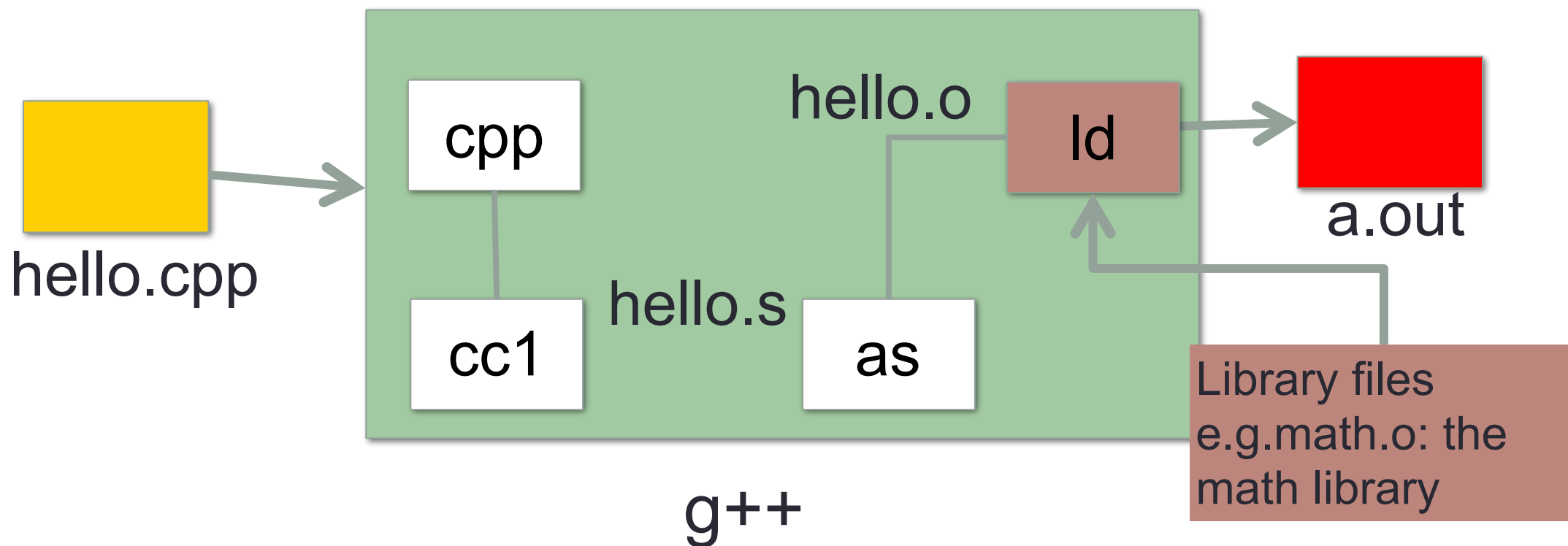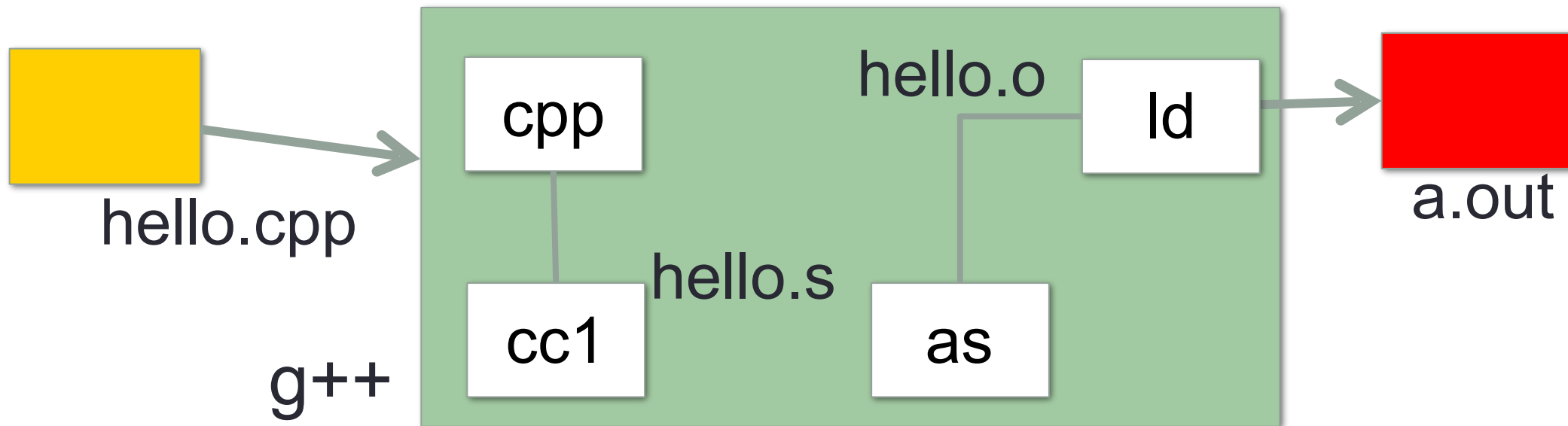
Steps in gcc

- Ask compiler to show temporary files:

$ g++ –S hello.cpp

$ g++ –c hello.o

$ g++ –o hello hello.cpp

$ g++ functions.o main.o –o myhello

Make and makefiles

- The unix make program automates the compilation process as specified in a Makefile
- Specifies how the different pieces of a program in different files fit together to make a complete program
- In the makefile you provide a recipe for compilation
- When you run make it will use that recipe to compile the program

```
$ make
g++ testShapes.o shapes.o tdd.o –o testShapes
```

Specifying a recipe in the makefile

- **Comments** start with a #
- **Definitions** typically are a variable in all caps followed by an equals sign and a string, such as:

```
CXX=g++
CXXFLAGS=-Wall

BINARIES=proj1
```

```
# testShapes is the target - it is what we want to produce
# To produce the executable testShapes we need all the .o files
# Everything to the right of ":" is a dependency for testShapes

testShapes: testShapes.o shapes.o tdd.o
        #The recipe for producing the target (testshapes) is below
        g++ testShapes.o shapes.o tdd.o -o testShapes
```

Demo

- Basics of code compilation in C++ (review)
- Makefiles (used to automate compilation of medium to large projects) consisting of many files
- We will start by using a makefile to compile just a single program
- Extend to the case where your program is split between multiple files
- Understand what each of the following are and how they are used in program compilation
  - Header file (.h)
  - Source file (.cpp)
  - Object file (.o)
  - Executable
  - Makefile
  - Compile-time errors
  - Link-time errors

The runtime Stack

Stack: A region in program memory to "manage" local variables

Every time a function is called, its local variables are created on the stack

When the function returns, local variables are removed from the stack

Local variables are created and deleted on the stack using a Last in First Out principle

```cpp
int sum(int a, int b){
        cout<< a+b;
}
int main(){
        int result =0;
        int x =10, y =20;
        result = sum(x, y);
        cout<<result;
}
```

Print vs return

What is the output of the following code

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

}
int main(){
        int result =0;
        int x =10, y =20;
        result = sum(x, y);
        cout<<result;

}
```

Function call mechanics

What is the output of the following code

```
int sum(int a, int b){
        int result= a+b;
        exit(0);
}

int main(){
        int result =0;
        int x =10, y =20;
        result = sum(x, y);
        cout<<result;
}
```

Next time

- Files