

COMPILATION REVISITED

DATA REPRESENTATION

Problem Solving with Computers-I

C++

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

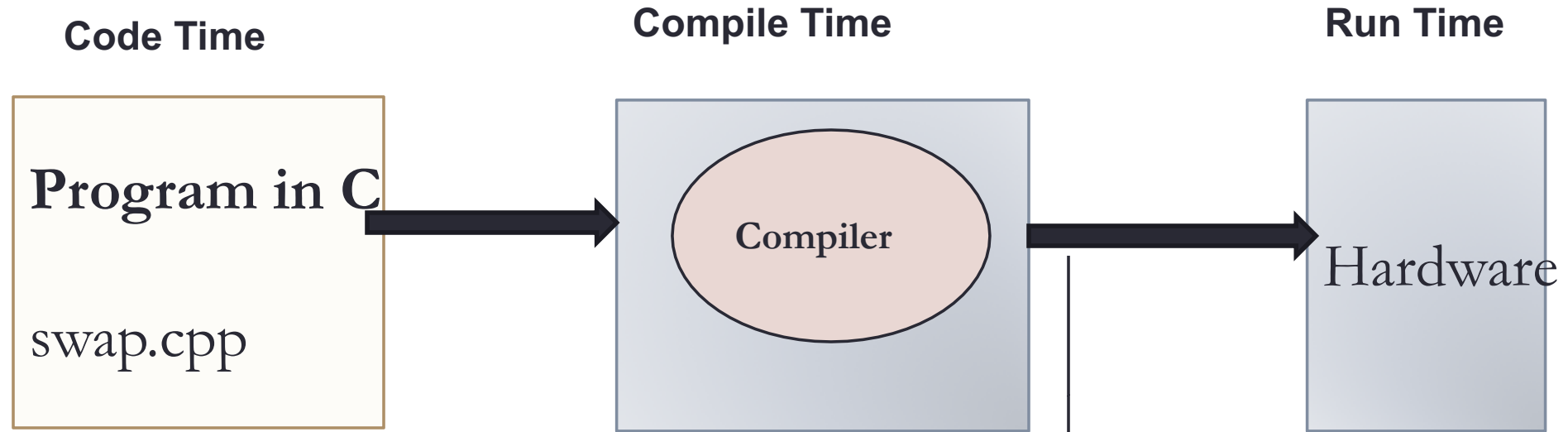
int main(){
    cout<<"Hola Facebook!";
    return 0;
}
```



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
- By the end of this you should know 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

Steps in program translation



Program:

Text file stored on computers hard disk or some secondary storage

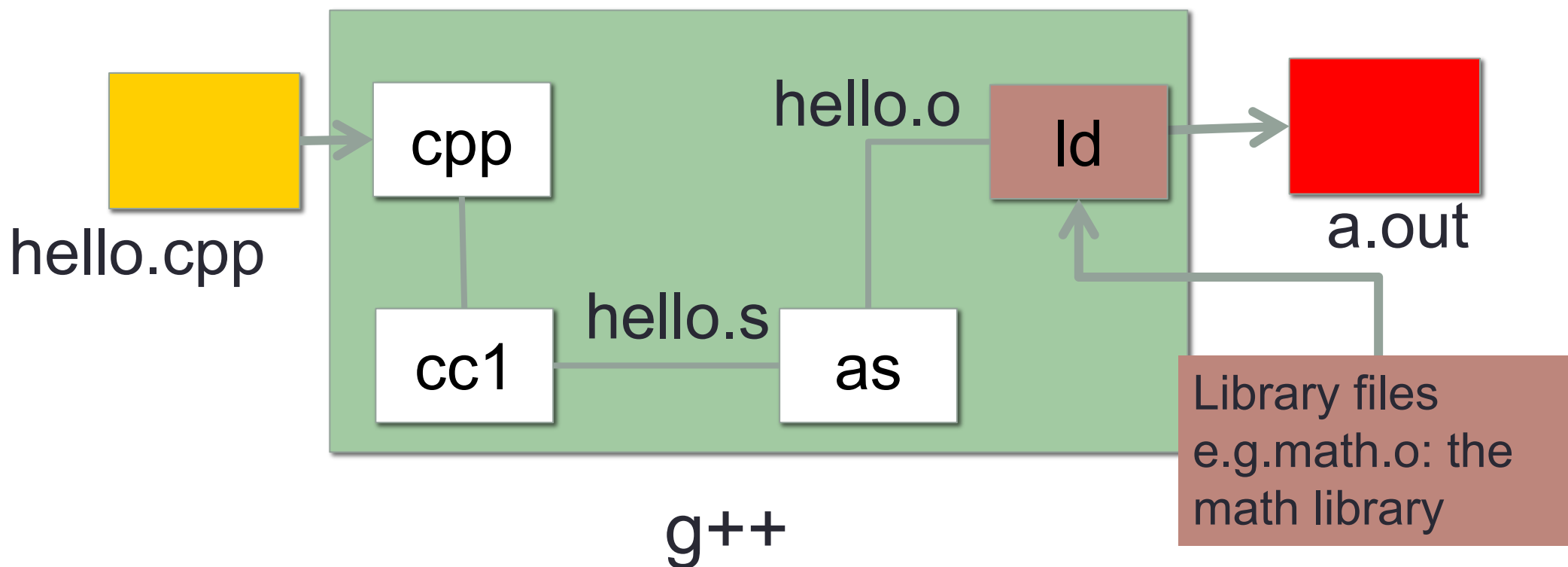
Executable:

Program in machine code
+Data in binary

```
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
```

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

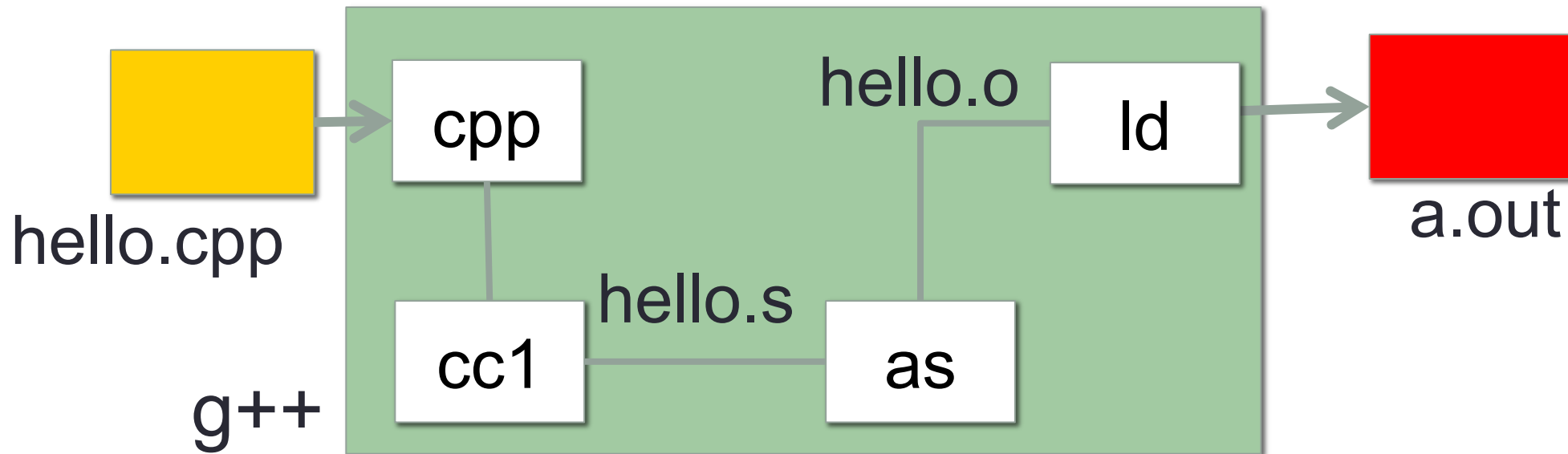
- Ask compiler to show temporary files:

```
$ 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:

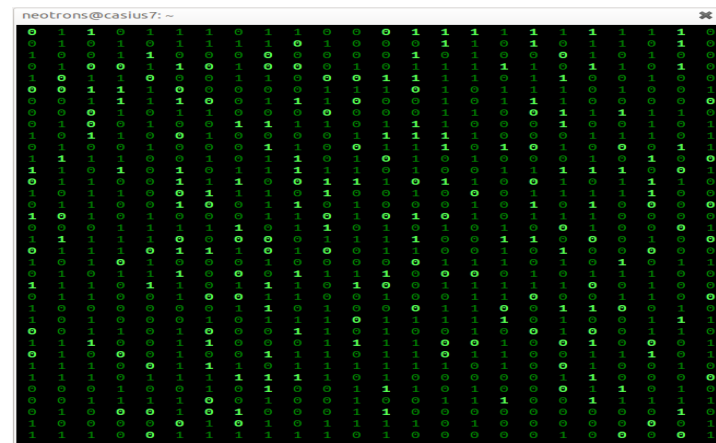
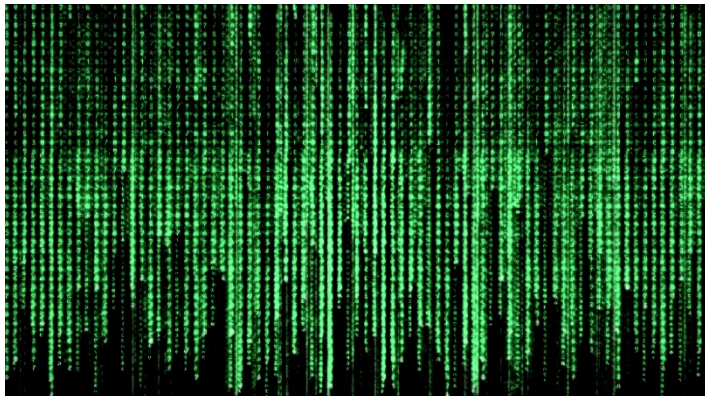
```
CXX=g++  
CXXFLAGS=-Wall  
  
BINARIES=proj1
```

```
# 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
```

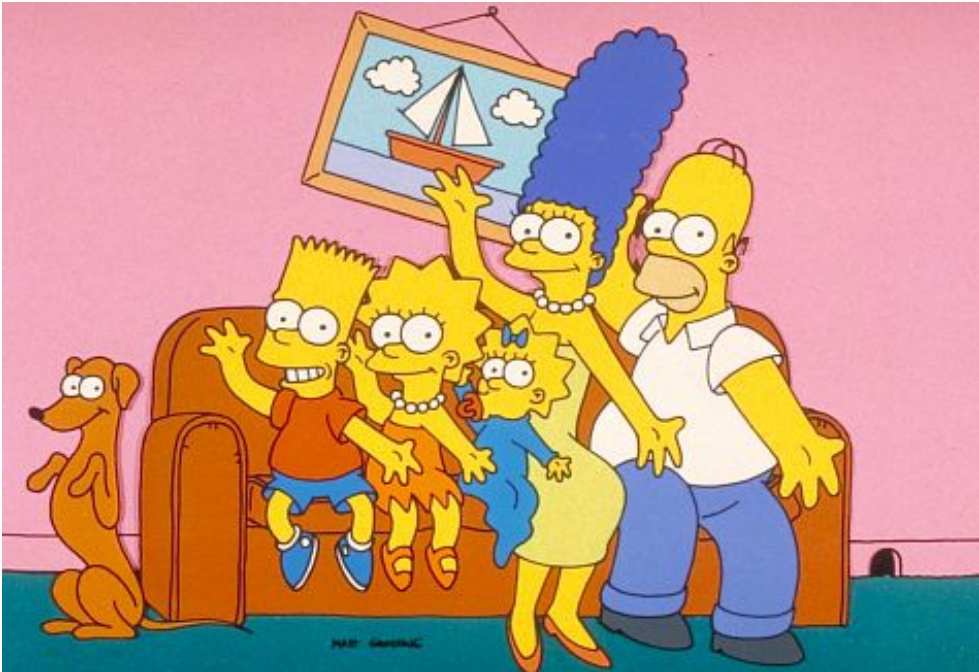
What does 'data' on a computer look like?

- Imagine diving deep into a computer
- Expect to see all your data as high and low voltages
- In CS we use the abstraction:
 - High voltage: 1 (true)
 - Low voltage: 0 (false)



Decimal (base ten)

- Why do we count in base ten?
- Which base would the Simpson's use?



External vs. Internal Representation

- **External representation:**
 - Convenient for programmer
- **Internal representation:**
 - Actual representation of data in the computer's memory and registers: Always binary (1's and 0's)

Positional encoding for non-negative numbers

- Each position represents some power of the base

Why is each base important??

$101_5 = ?$ In decimal

A. 26

B. 51

C. 126

D. 130

Binary representation (base 2)

- On a computer all data is stored in binary
- Only two symbols: 0 and 1
- Each position is called a *bit*
- *Bits take up space*
- 8 bits make a *byte*
- *Example of a 4-bit number*

Converting between binary and decimal

Binary to decimal: $1\ 0\ 1\ 1\ 0_2 = ?_{10}$

Decimal to binary: $34_{10} = ?_2$

Hex to binary

- Each hex digit corresponds directly to four binary digits
- Programmers love hex, why?

$25B_{16} = ?$ In binary

00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Hexadecimal to decimal

$$25B_{16} = ? \text{ Decimal}$$

Hexadecimal to decimal

- Use polynomial expansion
- $25B_{16} = 2*256 + 5*16 + 11*1 = 512 + 80 + 11$
 $= 603$
- Decimal to hex: $36_{10} = ?_{16}$

Binary to hex: 1000111100

A. 8F0

B. 23C

C. None of the above

BIG IDEA: Bits can represent anything!!

Numbers	Binary Code
----------------	--------------------

0	
---	--

1	
---	--

2	
---	--

3	
---	--

How many (minimum) bits are required to represent the numbers 0 to 3?

BIG IDEA: Bits can represent anything!!

Colors



Binary code

How many (minimum) bits are required to represent the three colors?

BIG IDEA: Bits can represent anything!!

Characters

'a'

'b'

'c'

'd'

'e'

N bits can represent at most 2^N things

What is the minimum number of bits required to represent all the letters in the English alphabet in lower case?

- A. 3
- B. 4
- C. 5
- D. 6
- E. 26

BIG IDEA: Bits can represent anything!!

- Logical values?

- 0 \Rightarrow False, 1 \Rightarrow True

- colors ?

- Characters?

- 26 letters \Rightarrow 5 bits ($2^5 = 32$)
- upper/lower case + punctuation \Rightarrow 7 bits (in 8) (“ASCII”)

- standard code to cover all the world’s languages \Rightarrow 8,16,32 bits (“Unicode”)

www.unicode.com

- locations / addresses? commands?

- **MEMORIZE:** N bits \Leftrightarrow at most 2^N things



What is the maximum positive value that can be stored in a byte?

- A. 127
- B. 128
- C. 255
- D. 256

Data types

Binary numbers in memory are stored using a finite, fixed number of bits typically:

- 8 bits (byte)
- 16 bits (half word)
- 32 bits (word)
- 64 bits (double word or quad)

Data type of a variable determines the:

- exact representation of variable in memory
- number of bits used (fixed and finite)
 - range of values that can be correctly represented

Next time

- Arrays