THE UNIVERSITY OF ALABAMA AT BIRMINGHAM

CS330 Lab2

Make
C: Intro, Variables, Loops, Functions

Spring 2023

Compiled by John Bedingfield and Dylan Calvin

Feedback from Lab 1 Assignment

- Great Job!
- Additional how-to info on Canvas:
  - VSCode Remote Development Config Video
  - Command Line Video (YouTube)
- If you had issues, or couldn’t complete the assignment, please come see us !!
- Any questions?

- Going forward:
  - We’ll ask for source code, no more binary
  - We’ll ask for all the files to be placed in a .zip file
  - We’ll ask for a Makefile

Compiled by John Bedingfield and Dylan Calvin

Today’s Agenda

- Make
- Variables
- Looping, for
- Control Flow: if, else
- Functions

UAB THE UNIVERSITY OF ALABAMA AT BIRMINGHAM.

Make

Brief overview of Makefiles

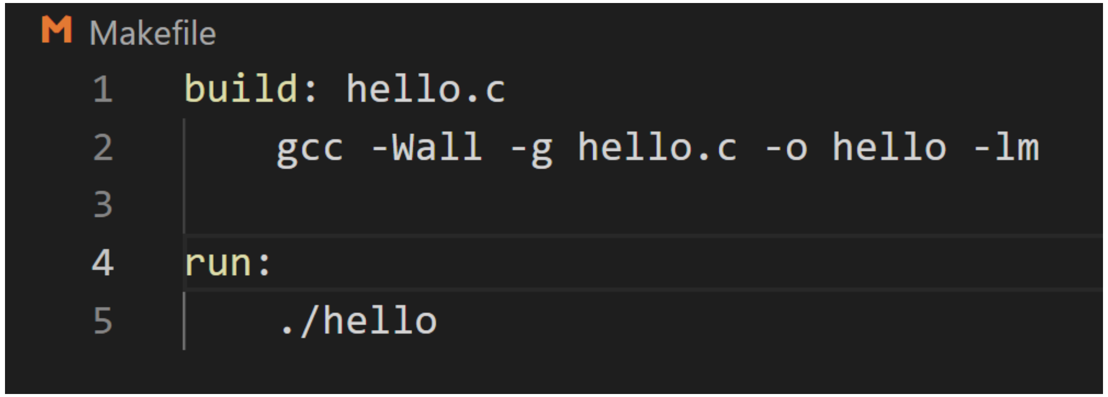
- A Makefile is full of quick scripts, usually meant for the preparation of your program. For our purposes, we want to write down all the instructions we use to compile our C programs. Make will only compile those files that have changed.
- Upon running "make", the first group of instructions in the file will be executed.
- You can also add multiple groups of instructions for different purposes, and invoke them with "make [name]“
- A Makefile is made up of rules, each consisting of:

$$\text{rule(target): prerequisites}$$
$$\quad\text{recipe (action)}$$

Compiled by John Bedingfield and Dylan Calvin

Makefiles, continued

Here is an example Makefile, saved as Makefile, no extension:



```
build: hello.c
	gcc -Wall -g hello.c -o hello -lm

run:
	./hello
```

Note: second line (recipe line) must start with a tab

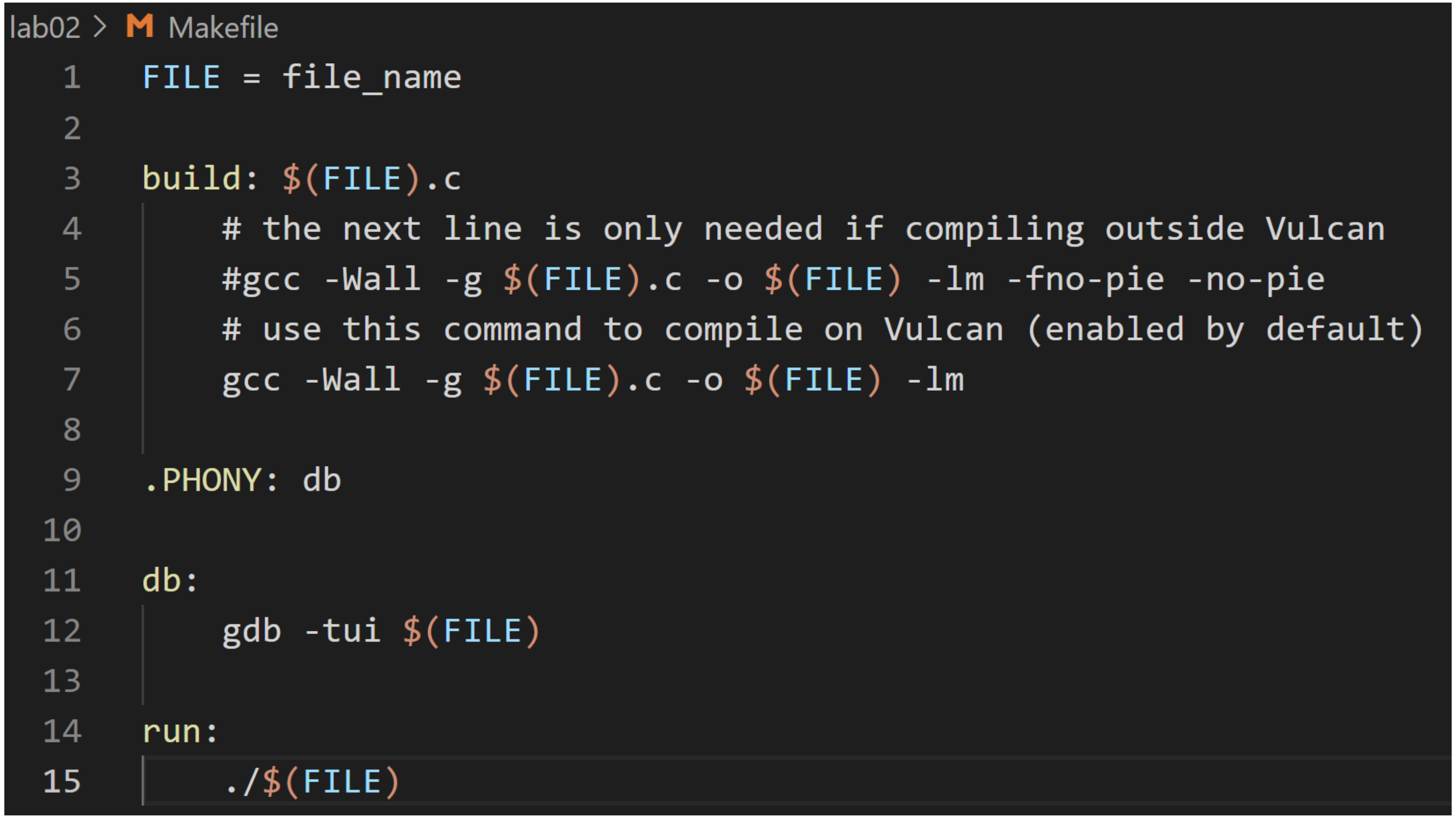
make will run everything inside ‘build’, and make run will run everything inside ‘run’

Manual: https://www.gnu.org/software/make/manual/

Ref: https://ocw.mit.edu/courses/civil-and-environmental-engineering/1-124j-foundations-of-software-engineering-fall-2000/lecture-notes/makefile_primer/

Compiled by John Bedingfield and Dylan Calvin

Makefile on Canvas to use as a template



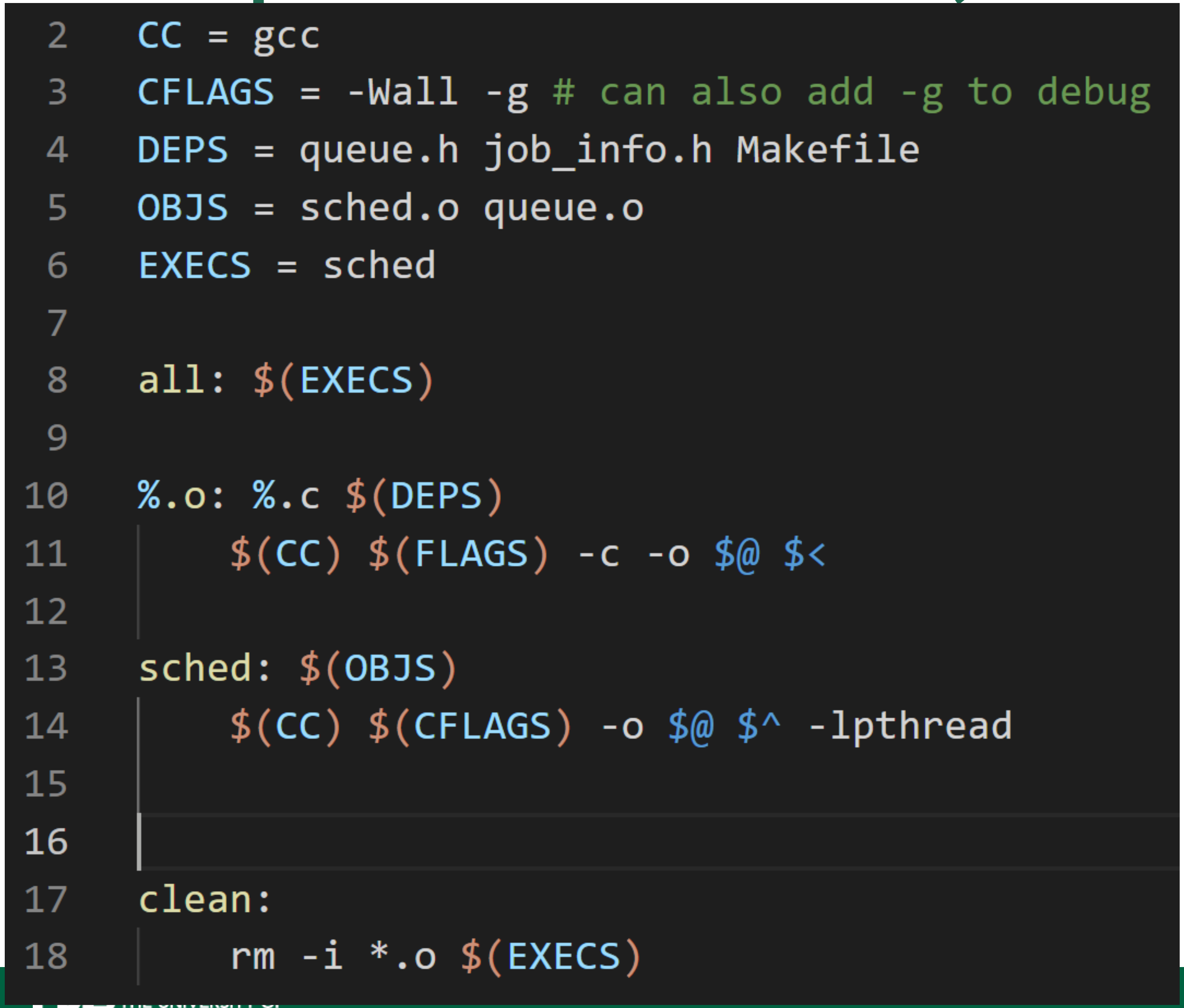
```
FILE = file_name

build: $(FILE).c
    # the next line is only needed if compiling outside Vulcan
    #gcc -Wall -g $(FILE).c -o $(FILE) -lm -fno-pie -no-pie
    # use this command to compile on Vulcan (enabled by default)
    gcc -Wall -g $(FILE).c -o $(FILE) -lm

.PHONY: db

db:
    gdb -tui $(FILE)

run:
    ./$(FILE)
```

- $\#$
    - a comment
- $\mathbf{FILE} =$
    - Sets a variable for later use
    - Just replace file_name with the name of your file (no extension)
- $\$(\mathbf{FILE})$
    - Is replaced with variable FILE

Must be saved as Makefile, capital M, no extension

Compiled by John Bedingfield and Dylan Calvin

More powerful Makefile (not required for this class)



```
CC = gcc
CFLAGS = -Wall -g # can also add -g to debug
DEPS = queue.h job_info.h Makefile
OBJS = sched.o queue.o
EXECS = sched

all: $(EXECS)

%.o: %.c $(DEPS)
	$(CC) $(FLAGS) -c -o $@ $<

sched: $(OBJS)
	$(CC) $(CFLAGS) -o $@ $^ -lpthread


clean:
	rm -i *.o $(EXECS)
```

- Syntax:
  - <RULE: DEPENDENCY LINE>
    <tab><ACTION LINE>
  - Dependency line is: <target files:> [source files]
- % is pattern matching
  - See: https://www.gnu.org/software/make/manual/make.html#Pattern-Rules
- Automatic Variables
  - $@ file name of the rule target
  - $< for first prerequisite source file name
  - $^ for all prerequisites separated by spaces
  - See: https://www.gnu.org/software/make/manual/make.html#Automatic-Variables

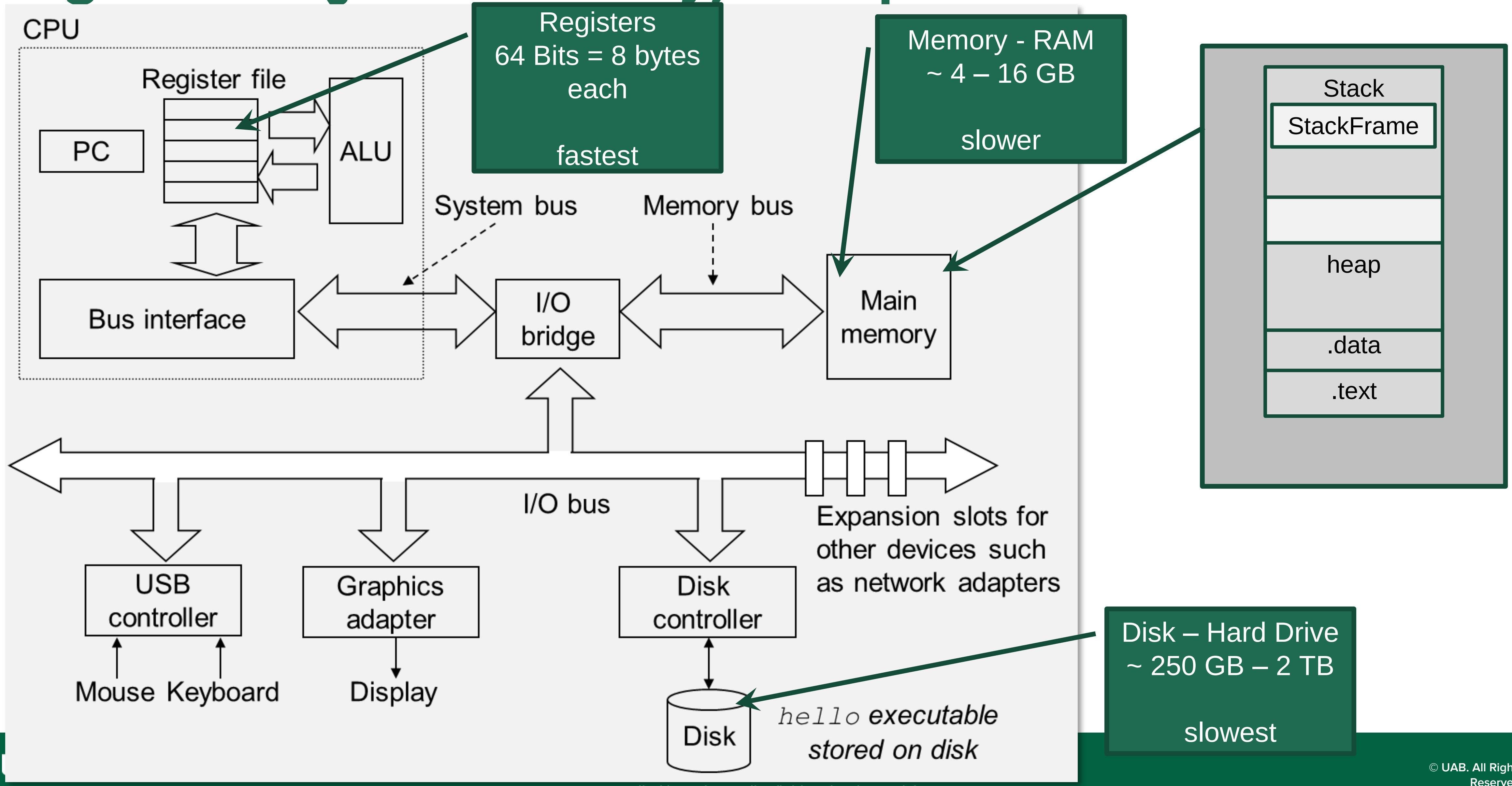
THE UNIVERSITY OF
ALABAMA AT BIRMINGHAM.

It’s all about the memory

And our mental model of a Computer

Compiled by John Bedingfield and Dylan Calvin

High Level (just for today) Computer Architecture



Registers
64 Bits = 8 bytes each

fastest

Memory - RAM
~ 4 – 16 GB

slower

Disk – Hard Drive
~ 250 GB – 2 TB

slowest

THE UNIVERSITY OF ALABAMA AT BIRMINGHAM.

C

The Basics

Compiled by John Bedingfield and Dylan Calvin

The ANSI Standard

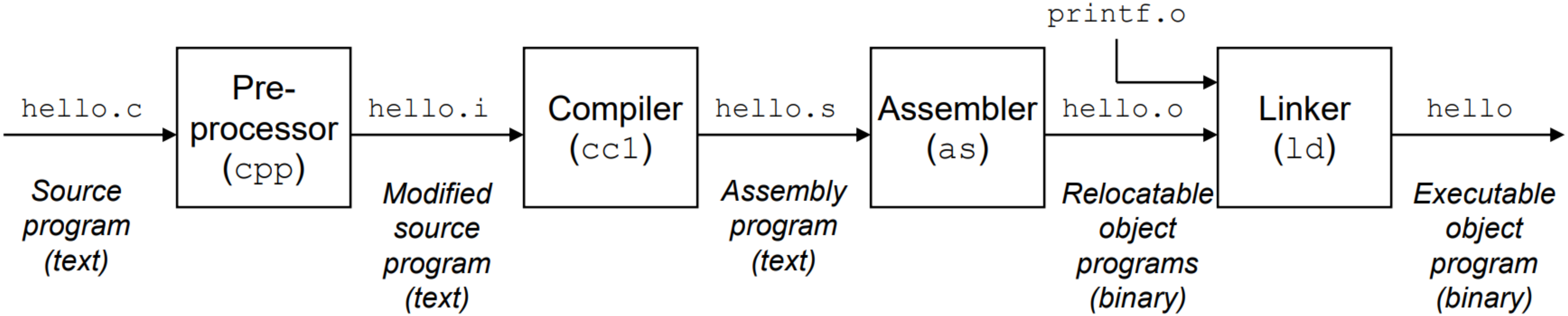
- The rapid expansion of the C language
  - Many companies developed their own C compilers
- In 1983, the American National Standard Institute (ANSI) began the development of the C standard that was completed and formally approved, in 1989, as ANSI C or Standard C
    - C89
    - C90
    - C95
    - C99
    - C11
    - C18

C (and Assembly) is a Compiled Language

When we type:

gcc hello.c –o hello

The code is compiled as follows:



printf.o

hello.c

Source program (text)

Pre-processor (cpp)

hello.i

Modified source program (text)

Compiler (cc1)

hello.s

Assembly program (text)

Assembler (as)

hello.o

Relocatable object programs (binary)

Linker (ld)

hello

Executable object program (binary)

Compilation System

THE UNIVERSITY OF ALABAMA AT BIRMINGHAM

Variables

Compiled by John Bedingfield and Dylan Calvin

C Primitive Types

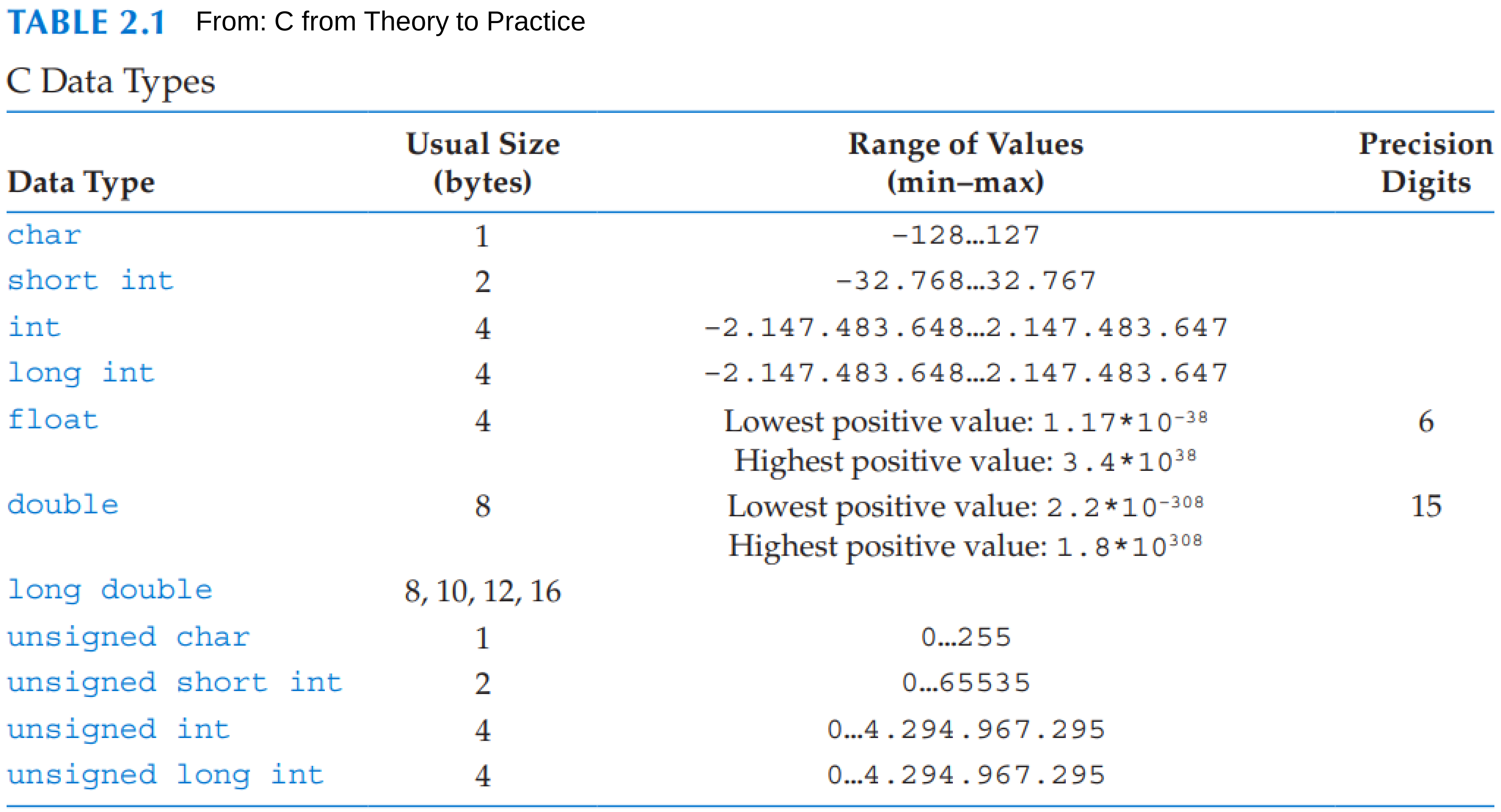


TABLE 2.1 From: C from Theory to Practice

C Data Types

Data Type	Usual Size (bytes)	Range of Values (min–max)	Precision Digits
char	1	-128...127	
short int	2	-32.768...32.767	
int	4	-2.147.483.648...2.147.483.647	
long int	4	-2.147.483.648...2.147.483.647	
float	4	Lowest positive value: $1.17*10^{-38}$ Highest positive value: $3.4*10^{38}$	6
double	8	Lowest positive value: $2.2*10^{-308}$ Highest positive value: $1.8*10^{308}$	15
long double	8, 10, 12, 16		
unsigned char	1	0...255	
unsigned short int	2	0...65535	
unsigned int	4	0...4.294.967.295	
unsigned long int	4	0...4.294.967.295	

Variable Declaration and Storage Classes

Variable Declaration Syntax:
[Storage Class] <type> <name>;

Note: There is also a keyword: **const** (constant) Variable can be initialized, but not changed

Storage Class	Scope	Lifetime	Default Value
auto (default)	Same block	Until the block completes	Garbage Value
static	Same block	Until the program completes	0 (for int)
extern	Program	Until the program completes	0 (for int)
registers (fast, not guaranteed)	Same block	Until the block completes	Garbage Value

Scope: where variable can be used Lifetime: how long the variable is in memory

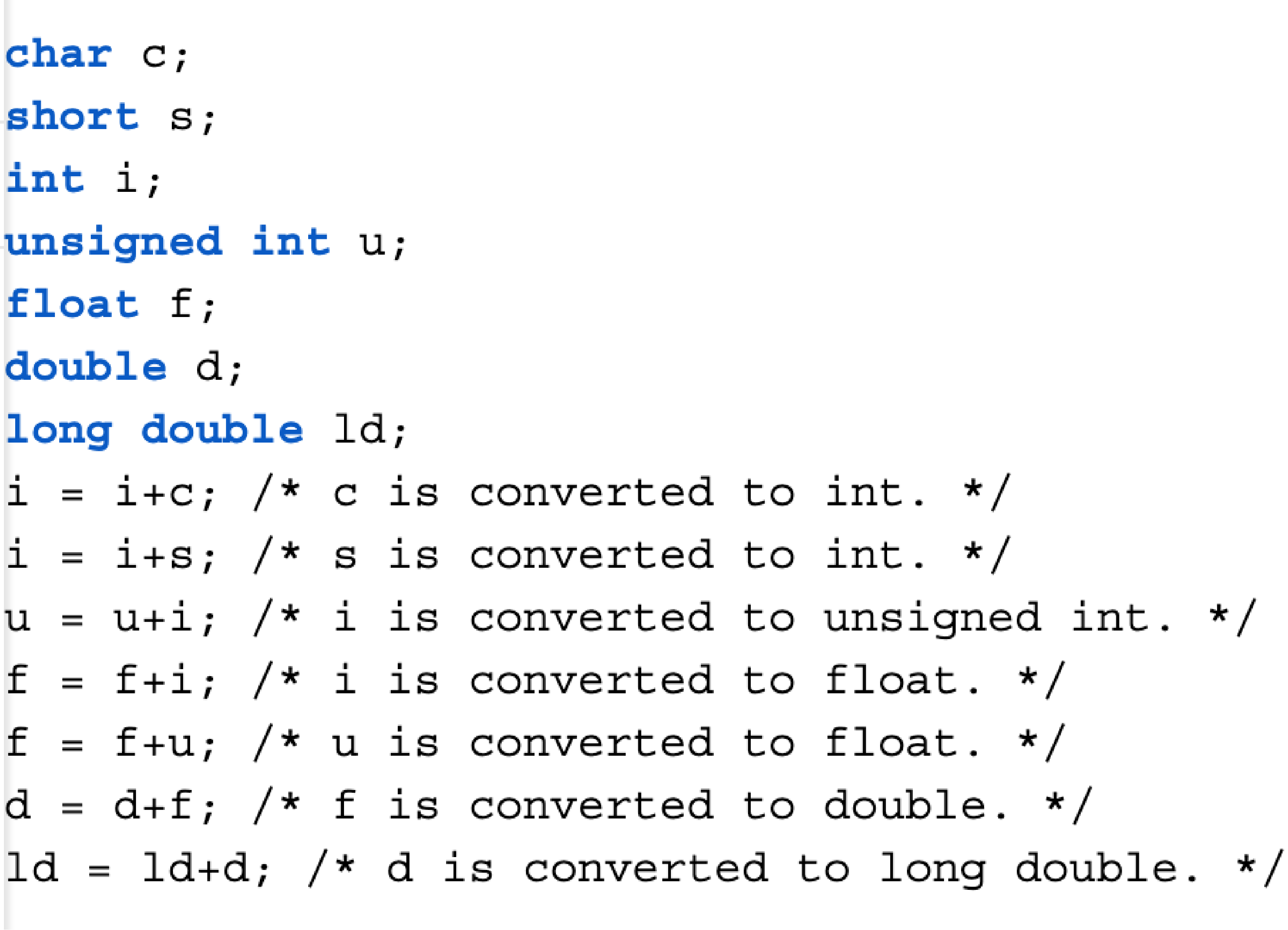
Compiled by John Bedingfield and Dylan Calvin

Variable Names

- The name can contain letters, digits, and underscore characters _.
- The name must begin with either a letter or underscore
- C is case sensitive (distinguishes btw uppercase and lowercase)
- The following keywords cannot be used as variable names because they have special significance to the C compiler:

auto
break
case
char
const
continue
default
do
double
else
enum
extern
float
for
goto
if
int
long
register
return
short
signed
sizeof
static
struct
switch
typedef
union
unsigned
void
volatile
while

Arithmetic Conversions



```
char c;
short s;
int i;
unsigned int u;
float f;
double d;
long double ld;
i = i+c; /* c is converted to int. */
i = i+s; /* s is converted to int. */
u = u+i; /* i is converted to unsigned int. */
f = f+i; /* i is converted to float. */
f = f+u; /* u is converted to float. */
d = d+f; /* f is converted to double. */
ld = ld+d; /* d is converted to long double. */
```

Compiled by John Bedingfield and Dylan Calvin

Arithmetic Operators

- + - / * %
- int/int = cuts off the decimal part

```
int a=7;
int b=5;
a/b will be equal to 1
```

also, be careful with the % operator

```
if ((n%2)==1) is dangerous**
if((n%2)!=0)  is safe
```

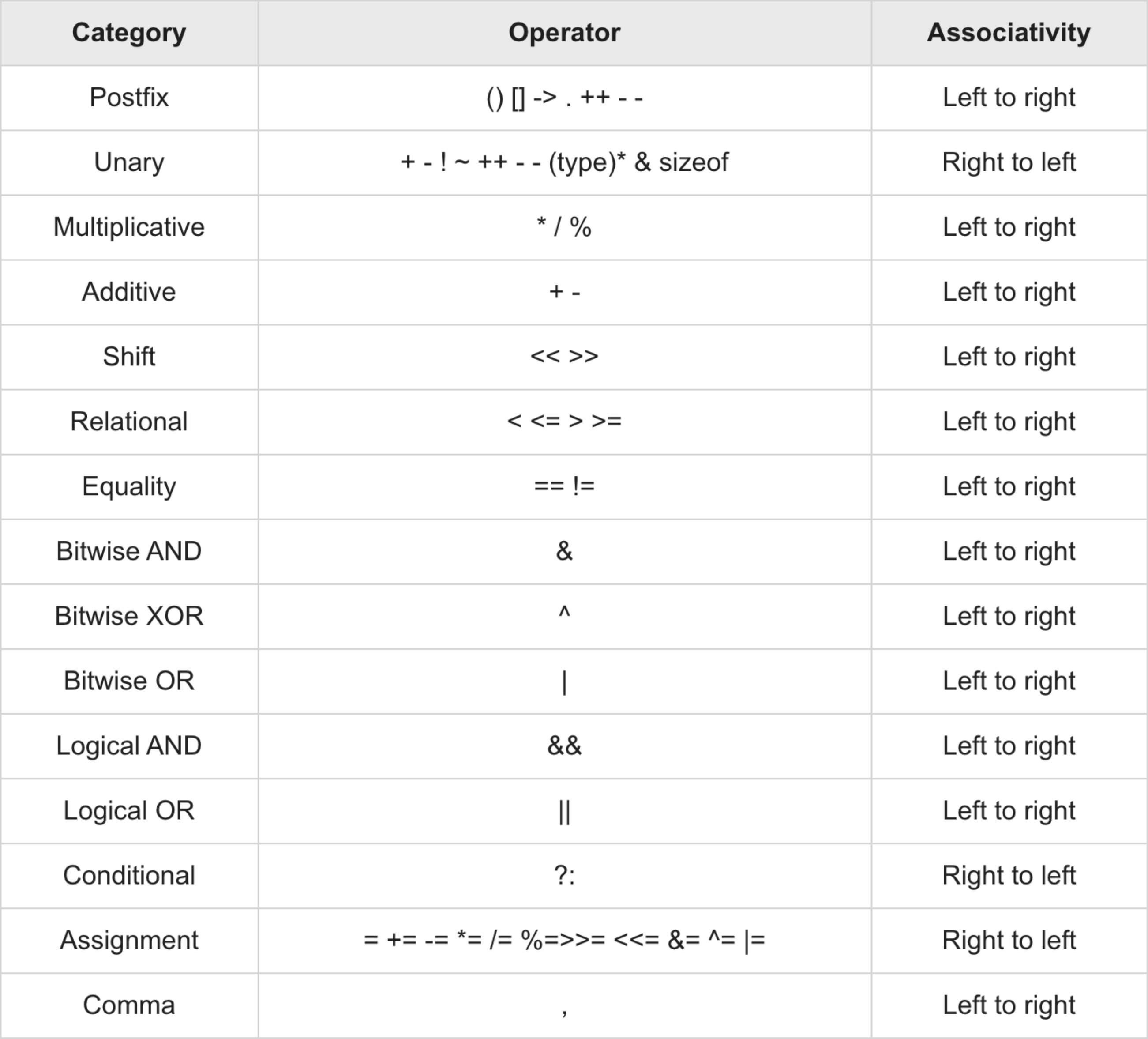
** if n is odd and negative

Comparisons

- >   >=   <   <=   !=   ==
- if (a == 10)

	Logical	Bitwise
NOT	!	~ (one's complement)
AND	&&	&
OR	||	| (inclusive OR) ^ (exclusive OR)
Leftwise bit shift		<<
Rightwise bit shift		>>

Operator Precedence

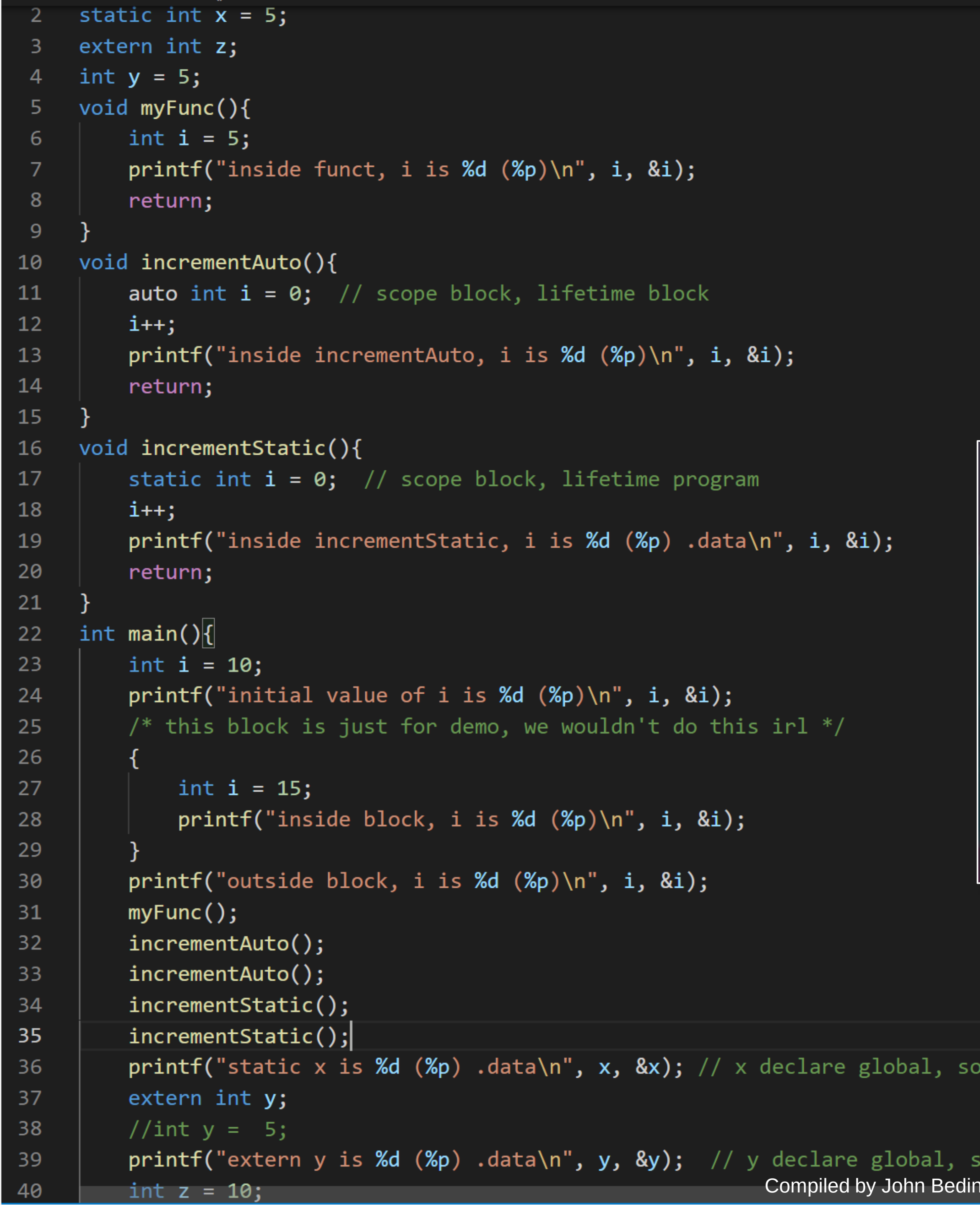


Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

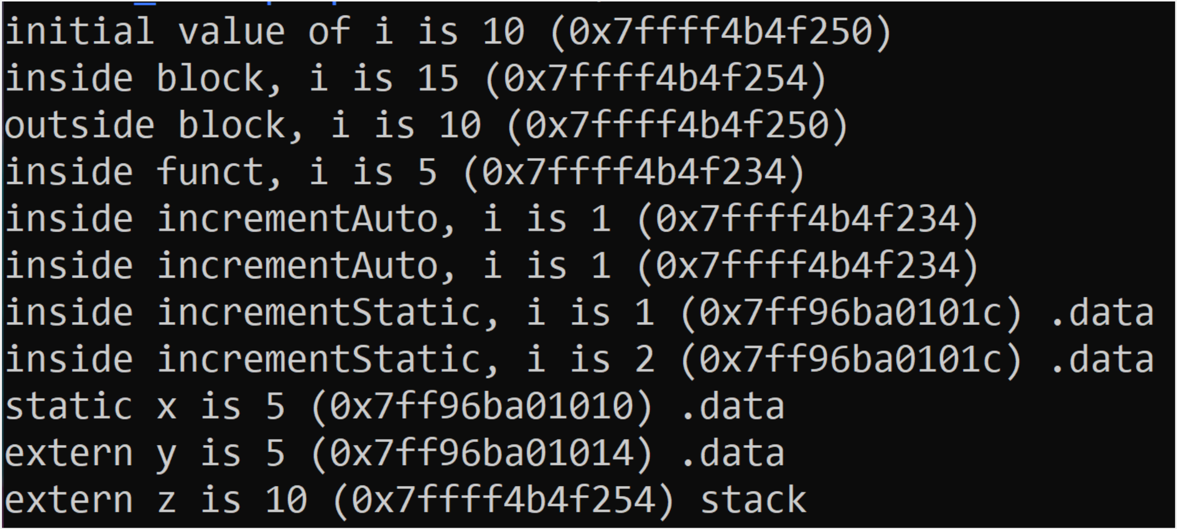
https://www.tutorialspoint.com/cprogramming/c_operators_precedence.htm

Compiled by John Bedingfield and Dylan Calvin

Vars.c



```
static int x = 5;
extern int z;
int y = 5;
void myFunc(){
    int i = 5;
    printf("inside funct, i is %d (%p)\n", i, &i);
    return;
}
void incrementAuto(){
    auto int i = 0;  // scope block, lifetime block
    i++;
    printf("inside incrementAuto, i is %d (%p)\n", i, &i);
    return;
}
void incrementStatic(){
    static int i = 0;  // scope block, lifetime program
    i++;
    printf("inside incrementStatic, i is %d (%p) .data\n", i, &i);
    return;
}
int main(){
    int i = 10;
    printf("initial value of i is %d (%p)\n", i, &i);
    /* this block is just for demo, we wouldn't do this irl */
    {
        int i = 15;
        printf("inside block, i is %d (%p)\n", i, &i);
    }
    printf("outside block, i is %d (%p)\n", i, &i);
    myFunc();
    incrementAuto();
    incrementAuto();
    incrementStatic();
    incrementStatic();
    printf("static x is %d (%p) .data\n", x, &x); // x declare global, so it's in .data
    extern int y;
    //int y =  5;
    printf("extern y is %d (%p) .data\n", y, &y);  // y declare global, so it's in .data
    int z = 10;
```



```
initial value of i is 10 (0x7ffff4b4f250)
inside block, i is 15 (0x7ffff4b4f254)
outside block, i is 10 (0x7ffff4b4f250)
inside funct, i is 5 (0x7ffff4b4f234)
inside incrementAuto, i is 1 (0x7ffff4b4f234)
inside incrementAuto, i is 1 (0x7ffff4b4f234)
inside incrementStatic, i is 1 (0x7ff96ba0101c) .data
inside incrementStatic, i is 2 (0x7ff96ba0101c) .data
static x is 5 (0x7ff96ba01010) .data
extern y is 5 (0x7ff96ba01014) .data
extern z is 10 (0x7ffff4b4f254) stack
```

https://youtu.be/hxh8cORcerM

THE UNIVERSITY OF
ALABAMA AT BIRMINGHAM.

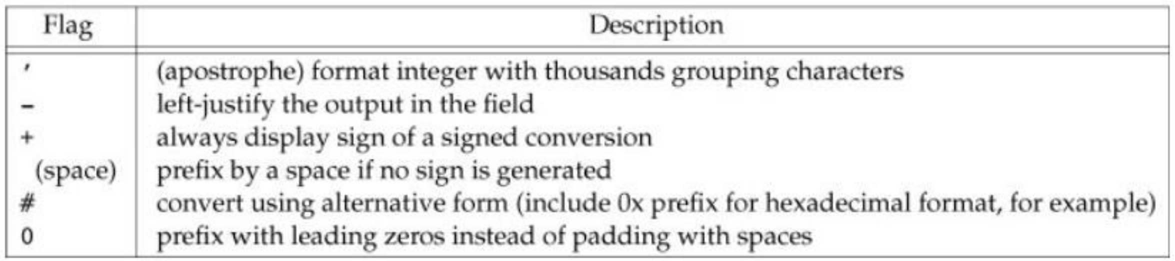
printf

Compiled by John Bedingfield and Dylan Calvin

printf

printf Syntax: % conversion specifier

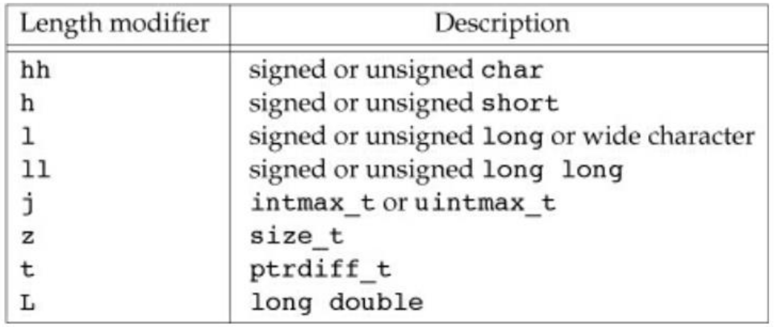
% [flags] [min field width] [precision] [length] <conversion type>



Flag	Description
'	(apostrophe) format integer with thousands grouping characters
-	left-justify the output in the field
+	always display sign of a signed conversion
(space)	prefix by a space if no sign is generated
#	convert using alternative form (include 0x prefix for hexadecimal format, for example)
0	prefix with leading zeros instead of padding with spaces

Figure 5.7 The flags component of a conversion specification

Precision:
e.g. .3 is three digits after the decimal
Default is 6 digits, if precision is 0 no decimal appears (f)



Length modifier	Description
hh	signed or unsigned `char`
h	signed or unsigned `short`
l	signed or unsigned `long` or wide character
ll	signed or unsigned `long long`
j	`intmax_t` or `uintmax_t`
z	`size_t`
t	`ptrdiff_t`
L	`long double`

Figure 5.8 The length modifier component of a conversion specification

Conversion Type	Description
d	signed decimal
f	double floating-point number
c	character
s	string
p	pointer
x, X	unsigned hexadecimal
%	% character
o	unsigned octal
u	unsigned decimal
a, A	double floating-point number in hexadecimal exponential format
e, E	double floating-point number in exponential format

Use “\” as escape character. e.g. “\n” newline, “\t” tab, “\\” is backslash

For more info, see man page: https://linux.die.net/man/3/printf
Also: http://www.pixelbeat.org/programming/gcc/format_specs.html

THE UNIVERSITY OF
ALABAMA AT BIRMINGHAM.

Looping – for loop

```
int i;
// NOTE: on Vulcan, must declare the variable outside for loop
for(i = 0; i < 5; i++) {
        // do stuff
}
```

THE UNIVERSITY OF
ALABAMA AT BIRMINGHAM.

Control Flow – if-else

```
if(expression1){
  // do stuff if expression1 evaluates to true
} else if (expression2) {
  // do other stuff if expression2 evaluates to true

} else {
  // do stuff if all other expressions are false
}
```

Compiled by John Bedingfield and Dylan Calvin

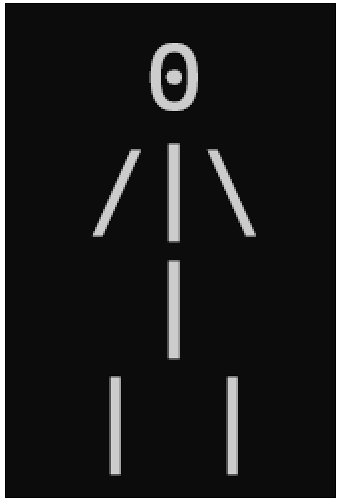
THE UNIVERSITY OF ALABAMA AT BIRMINGHAM

Functions

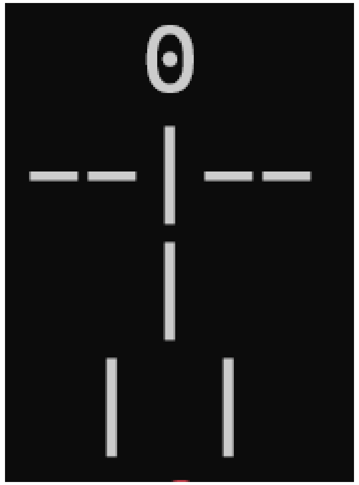
Compiled by John Bedingfield and Dylan Calvin

Exercise – Jumping Jack Person

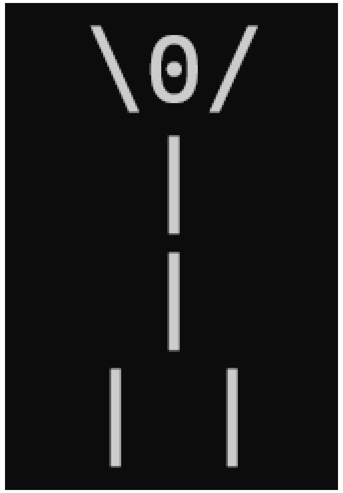
- Write a C program to generate the image (ASCII) of a person doing 10 jumping jacks
    - Each individual image should be a function
- To clear the screen:
    - system("clear");
    - Don't forget to #include<stdlib.h>
- To pause, you can use:
    - sleep([seconds])
    - usleep([milliseconds])
    - Don't forget to #include<unistd.h>
    - Note: if you printf before you sleep you need to clear the buffer with a newline "\n" or fflush(stdout);



```
 0
/|\
 |
| |
```



```
  0
--|--
  |
 | |
```



```
\0/
 |
 |
| |
```