

More C

1

## Layers of Abstraction

`x += y`

“Higher-level” language: C

`add %rbx, %rax`

Assembly: X86-64

60 03

Machine code: Y86

(we’ll talk later)

Logic and Registers

2

## selected anonymous feedback (1)

“If I finish the first lab before my lab time, do I still have to go?”

No.

“how much extra credit is offered per each phase?”

+2/30 (no explosions) or +1/30 (otherwise)

“Can we get key to previous quizzes?”

Sorry, I don’t have a convenient way of getting pre-Fall-2016

Also, there’s a small chance we might reuse quiz questions

3

## selected anonymous feedback (2)

“The quiz questions are overly difficult, and Lab 1’s 20 explosions rule is overly punitive. Bear in mind that some of us only have a few years of CS experience prior to this class.”

4

## bomblab notes

b \*0x12345678

set breakpoint at address 0x12345678

5

## non-anonymous feedback

6

## Last time

compile / assemble / link

x86-64 review

C data types, lack of bool

pointer arithmetic

arrays and pointers

7

## Arrays and Pointers

\*(foo + bar) **exactly the same** as foo[bar]

arrays **'decay'** into pointers

8

## Arrays: not quite pointers (1)

```
int array[100];  
int *pointer;
```

Legal: `pointer = array;`  
same as `pointer = &(array[0]);`

9

## Arrays: not quite pointers (1)

```
int array[100];  
int *pointer;
```

Legal: `pointer = array;`  
same as `pointer = &(array[0]);`

Illegal: ~~`array = pointer;`~~

9

## Arrays: not quite pointers (2)

```
int array[100];  
int *pointer = array;
```

`sizeof(array) == 400`  
size of all elements

10

## Arrays: not quite pointers (2)

```
int array[100];  
int *pointer = array;
```

`sizeof(array) == 400`  
size of all elements

`sizeof(pointer) == 8`  
size of address

10

## Arrays: not quite pointers (2)

```
int array[100];
int *pointer = array;
```

```
sizeof(array) == 400
    size of all elements
```

```
sizeof(pointer) == 8
    size of address
```

```
sizeof(&array[0]) == ???
    (&array[0] same as &(array[0]))
```

10

## initializing array versus pointer

```
const char *pointer = "Hello, World!";
char array[] = "Hello, World!";
```

pointer — probably points to **read-only data**

array — copy of string (including `\0`)

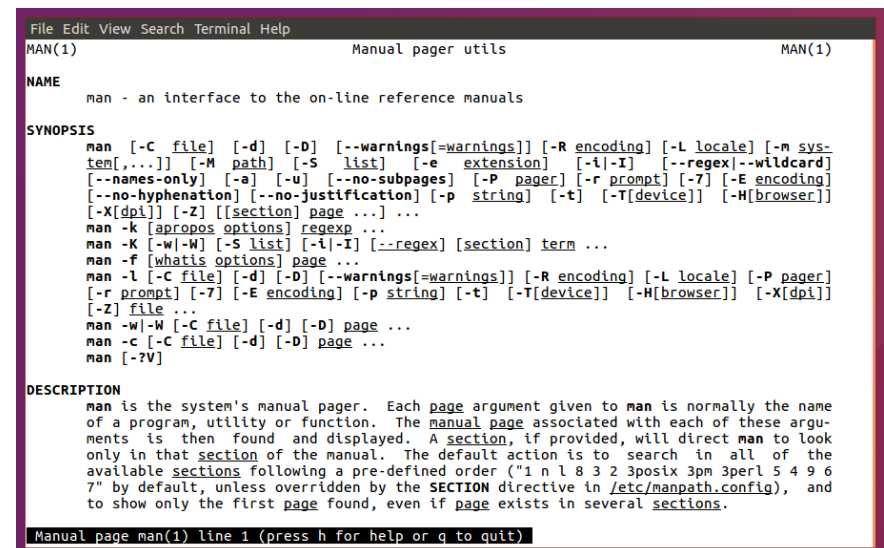
11

## Interlude: Command Line Tips

```
cr4bd@reiss-lenovo:~$ man man
```

12

## man man



```
File Edit View Search Terminal Help
MAN(1) Manual pager utils MAN(1)
NAME
  man - an interface to the on-line reference manuals
SYNOPSIS
  man [-C file] [-d] [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-m sys-
  tem,...] [-M path] [-S list] [-e extension] [-i|-I] [--regex|--wildcard]
  [--names-only] [-a] [-u] [--no-subpages] [-P pager] [-r prompt] [-7] [-E encoding]
  [--no-hyphenation] [--no-justification] [-p string] [-t] [-T[device]] [-H[browser]]
  [-X[dpi]] [-Z] [[section] page ...] ...
  man -k [apropos options] regexp ...
  man -K [-w|-W] [-S list] [-i|-I] [--regex] [section] term ...
  man -f [whatis options] page ...
  man -l [-C file] [-d] [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-P pager]
  [-r prompt] [-7] [-E encoding] [-p string] [-t] [-T[device]] [-H[browser]] [-X[dpi]]
  [-Z] file ...
  man -w|-W [-C file] [-d] [-D] page ...
  man -c [-C file] [-d] [-D] page ...
  man [-?V]
DESCRIPTION
  man is the system's manual pager. Each page argument given to man is normally the name
  of a program, utility or function. The manual page associated with each of these argu-
  ments is then found and displayed. A section, if provided, will direct man to look
  only in that section of the manual. The default action is to search in all of the
  available sections following a pre-defined order ("1 n l 8 3 2 3posix 3pm 3perl 5 4 9 6
  7" by default, unless overridden by the SECTION directive in /etc/manpath.config), and
  to show only the first page found, even if page exists in several sections.
Manual page man(1) line 1 (press h for help or q to quit)
```

13

# man man

```
File Edit View Search Terminal Help
EXAMPLES
man ls
  Display the manual page for the item (program) ls.

man -a intro
  Display, in succession, all of the available intro manual pages contained within
  the manual. It is possible to quit between successive displays or skip any of
  them.

man -t alias | lpr -Pps
  Format the manual page referenced by 'alias', usually a shell manual page, into the
  default troff or groff format and pipe it to the printer named ps. The default
  output for groff is usually PostScript. man --help should advise as to which pro-
  cessor is bound to the -t option.

man -l -Tdvi ./foo.1x.gz > ./foo.1x.dvi
  This command will decompress and format the nroff source manual page ./foo.1x.gz
  into a device independent (dvi) file. The redirection is necessary as the -T flag
  causes output to be directed to stdout with no pager. The output could be viewed
  with a program such as xdvi or further processed into PostScript using a program
  such as dvips.

man -k printf
  Search the short descriptions and manual page names for the keyword printf as regu-
  lar expression. Print out any matches. Equivalent to apropos printf.

man -f smail
  Lookup the manual pages referenced by smail and print out the short descriptions of
  any found. Equivalent to whatis smail.

Manual page man(1) line 68 (press h for help or q to quit)
```

14

# man chmod

```
File Edit View Search Terminal Help
CHMOD(1) User Commands CHMOD(1)
NAME
  chmod - change file mode bits

SYNOPSIS
  chmod [OPTION]... MODE[,MODE]... FILE...
  chmod [OPTION]... OCTAL-MODE FILE...
  chmod [OPTION]... --reference=RFILE FILE...

DESCRIPTION
  This manual page documents the GNU version of chmod. chmod changes the file mode bits
  of each given file according to mode, which can be either a symbolic representation of
  changes to make, or an octal number representing the bit pattern for the new mode bits.

  The format of a symbolic mode is [ugoa...][[+=][perms...][...], where perms is either
  zero or more letters from the set rwXst, or a single letter from the set ugo. Multi-
  ple symbolic modes can be given, separated by commas.

  A combination of the letters ugoa controls which users' access to the file will be
  changed: the user who owns it (u), other users in the file's group (g), other users not
  in the file's group (o), or all users (a). If none of these are given, the effect is
  as if (a) were given, but bits that are set in the unmask are not affected.

  The operator + causes the selected file mode bits to be added to the existing file mode
  bits of each file; - causes them to be removed; and = causes them to be added and
  causes unmentioned bits to be removed except that a directory's unmentioned set user
  and group ID bits are not affected.

  The letters rwXst select file mode bits for the affected users: read (r), write (w),
  Manual page chmod(1) line 1/125 27% (press h for help or q to quit)
```

15

# chmod

```
chmod --recursive og-r /home/USER
```

16

# chmod

```
chmod --recursive og-r /home/USER
```

others and group (student)  
- remove  
r read

16

## chmod

```
chmod --recursive og-r /home/USER
```

user (yourself) / group / others  
- remove / + add  
read / write / execute or search

16

## tar

the standard Linux/Unix file archive utility

Table of contents: tar tf filename.tar

eXtract: tar xvf filename.tar

Create: tar cvf filename.tar directory

(v: verbose; f: file — default is tape)

17

## Tab completion and history

18

## Back To C

19

# stdio.h

C does not have <iostream>

Instead <stdio.h>

20

# stdio

```
cr4bd@power1
: /if22/cr4bd ; man stdio
```

```
...
STDIO(3)                                Linux Programmer's Manual                                STDIO(3)
```

```
NAME
    stdio - standard input/output library functions
```

```
SYNOPSIS
    #include <stdio.h>
```

```
FILE *stdin;
FILE *stdout;
FILE *stderr;
```

```
DESCRIPTION
    The standard I/O library provides a simple and efficient buffered stream I/O interface. Input and output is mapped into logical data streams and the physical I/O characteristics are concealed. The functions and macros are listed below; more information is available from the individual man pages.
```

21

# stdio

```
STDIO(3)                                Linux Programmer's Manual                                STDIO(3)
```

```
NAME
    stdio - standard input/output library functions
```

...

## List of functions

Function	Description
clearerr	check and reset stream status
fclose	close a stream

...

```
printf    formatted output conversion
```

...

22

# printf

```
int custNo = 1000;
const char *name = "Jane_Smith"
printf("Customer_#%d : %s \n",
      custNo , name);
// "Customer #1000: Jane Smith"
// same as:
cout << "Customer_#" << custNo
    << ":_#" << name << endl;
```

23

## printf

```
int custNo = 1000;
const char *name = "Jane_Smith"
printf("Customer_#%d: %s \n",
      custNo, name);
// "Customer #1000: Jane Smith"
// same as:
cout << "Customer_#" << custNo
      << ":_ " << name << endl;
```

23

## printf

```
int custNo = 1000;
const char *name = "Jane_Smith"
printf("Customer_#%d : %s\n",
      custNo , name);
// "Customer #1000: Jane Smith"
// same as:
cout << "Customer_#" << custNo
      << ":_ " << name << endl;
```

23

## printf

```
int custNo = 1000;
const char *name = "Jane_Smith"
printf("Customer_#%d : %s \n",
      custNo , name);
// "Customer #1000: Jane Smith"
// same as:
cout << "Customer_#" << custNo
      << ":_ " << name << endl;
```

format string must **match types** of argument

23

## printf formats quick reference

Specifier	Argument Type	Example(s)
%s	char *	Hello, World!
%p	any pointer	0x4005d4
%d	int/short/char	42
%u	unsigned int/short/char	42
%x	unsigned int/short/char	2a
%ld	long	42
%f	double/float	42.000000 0.000000
%e	double/float	4.200000e+01 4.200000e-19
%g	double/float	42, 4.2e-19
%%	(no argument)	%

24



## printf formats quick reference

Specifier	Argument Type	Example(s)
%s	char *	Hello, World!
%p	any pointer	0x4005d4
%d	int/short/char	42
%u	unsigned int/short/char	42
%x	unsigned int/short/char	42
%ld	long	42
%f	double/float	42.000000 0.000000
%e	double/float	4.200000e+01 4.200000e-19
%g	double/float	42, 4.2e-19
%%	(no argument)	%

detailed docs: `man 3 printf`

24

## goto

```
for (...) {
    for (...) {
        if (thingAt(i, j)) {
            goto found;
        }
    }
}
printf("not_found!\n");
return;
found:
printf("found!\n");
```

25

## goto

```
for (...) {
    for (...) {
        if (thingAt(i, j))
            goto found;
    }
}
printf("not_found!\n");
return;
found:
printf("found!\n");
```

assembly:  
jmp found

assembly:  
found:

25

## struct

```
struct rational {
    int numerator;
    int denominator;
};
// ...
struct rational two_and_a_half;
two_and_a_half.numerator = 5;
two_and_a_half.denominator = 2;
struct rational *pointer = &two_and_a_half;
printf("%d/%d\n",
        pointer->numerator,
        pointer->denominator);
```

26

## struct

```
struct rational {
    int numerator;
    int denominator;
};
// ...
struct rational two_and_a_half;
two_and_a_half.numerator = 5;
two_and_a_half.denominator = 2;
struct rational *pointer = &two_and_a_half;
printf("%d/%d\n",
       pointer->numerator,
       pointer->denominator);
```

26

## typedef struct (1)

```
struct other_name_for_rational {
    int numerator;
    int denominator;
};
typedef struct other_name_for_rational rational;
// ...
rational two_and_a_half;
two_and_a_half.numerator = 5;
two_and_a_half.denominator = 2;
rational *pointer = &two_and_a_half;
printf("%d/%d\n",
       pointer->numerator,
       pointer->denominator);
```

27

## typedef struct (1)

```
struct other_name_for_rational {
    int numerator;
    int denominator;
};
typedef struct other_name_for_rational rational;
// ...
rational two_and_a_half;
two_and_a_half.numerator = 5;
two_and_a_half.denominator = 2;
rational *pointer = &two_and_a_half;
printf("%d/%d\n",
       pointer->numerator,
       pointer->denominator);
```

27

## typedef struct (2)

```
struct other_name_for_rational {
    int numerator;
    int denominator;
};
typedef struct other_name_for_rational rational;
// same as:
typedef struct other_name_for_rational {
    int numerator;
    int denominator;
} rational;
```

28

## typedef struct (2)

```
struct other_name_for_rational {
    int numerator;
    int denominator;
};
typedef struct other_name_for_rational rational;
// same as:
typedef struct other_name_for_rational {
    int numerator;
    int denominator;
} rational;
```

28

## typedef struct (2)

```
struct other_name_for_rational {
    int numerator;
    int denominator;
};
typedef struct other_name_for_rational rational;
// same as:
typedef struct other_name_for_rational {
    int numerator;
    int denominator;
} rational;
// almost the same as:
typedef struct {
    int numerator;
    int denominator;
} rational;
```

28

## linked lists / dynamic allocation

```
typedef struct list_t {
    int item;
    struct list_t *next;
} list;
// ...
```

29

## linked lists / dynamic allocation

```
typedef struct list_t {
    int item;
    struct list_t *next;
} list;
// ...
```

29

## linked lists / dynamic allocation

```
typedef struct list_t {
    int item;
    struct list_t *next;
} list;
// ...

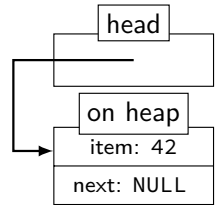
list* head = malloc(sizeof(list));
/* C++: new list; */
head->item = 42;
head->next = NULL;
// ...
free(head);
/* C++: delete list */
```

29

## linked lists / dynamic allocation

```
typedef struct list_t {
    int item;
    struct list_t *next;
} list;
// ...

list* head = malloc(sizeof(list));
/* C++: new list; */
head->item = 42;
head->next = NULL;
// ...
free(head);
/* C++: delete list */
```



29

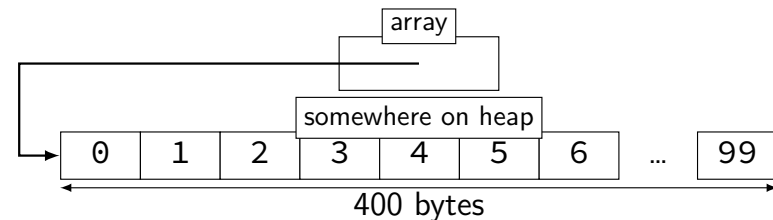
## dynamic arrays

```
int *array = malloc(sizeof(int)*100);
// C++: new int[100]
for (i = 0; i < 100; ++i) {
    array[i] = i;
}
// ...
free(array); // C++: delete[] array
```

30

## dynamic arrays

```
int *array = malloc(sizeof(int)*100);
// C++: new int[100]
for (i = 0; i < 100; ++i) {
    array[i] = i;
}
// ...
free(array); // C++: delete[] array
```



30

## Miss vector? (1)

```
typedef struct range_t {
    int size;
    int *data;
} range;
```

31

## Miss vector? (1)

```
typedef struct range_t {
    int size;
    int *data;
} range;

range vec;
vec.size = 100;
vec.data = malloc(sizeof(int) * 100);
// like: vector<int> vec(100);
```

31

## Miss vector? (2)

```
typedef struct range_t {
    int size;
    int *data;
} range;

range vec2;
vec2.size = vec.size;
vec2.data = malloc(sizeof(int) * vec.size);
for (int i = 0; i < vec.size; ++i) {
    vec2.data[i] = vec.data[i];
}
// like: vector<int> vec2 = vec;
```

32

## Miss vector? (2)

```
typedef struct range_t {
    int size;
    int *data;
} range;

range vec2;
vec2.size = vec.size;
vec2.data = malloc(sizeof(int) * vec.size);
for (int i = 0; i < vec.size; ++i) {
    vec2.data[i] = vec.data[i];
}
// like: vector<int> vec2 = vec;

Why not range vec2 = vec?
```

32

## unsigned and signed types

type	min	max
signed int = signed = int	$-2^{31}$	$2^{31} - 1$
unsigned int = unsigned	0	$2^{32} - 1$
signed long = long	$-2^{63}$	$2^{63} - 1$
unsigned long	0	$2^{64} - 1$

⋮

33

## unsigned/signed comparison trap

```
int x = -1;
unsigned int y = 0;
printf("%d\n", x < y);
```

34

## unsigned/signed comparison trap

```
int x = -1;
unsigned int y = 0;
printf("%d\n", x < y);
```

result is 0

34

## unsigned/signed comparison trap

```
int x = -1;
unsigned int y = 0;
printf("%d\n", x < y);
```

result is 0

short solution: don't compare signed to unsigned:

```
(long) x < (long) y
```

34

```
int x = -1;
unsigned int y = 0;
printf("%d\n", x < y);
```

compiler converts both to **same type** first

**int** if all possible values fit

otherwise: first operand (x, y) type from this list:

```
unsigned long
long
unsigned int
int
```

35

## C evolution and standards

1978: Kernighan and Ritchie publish *The C Programming Language* — “K&R C”

**very** different from modern C

36

## C evolution and standards

1978: Kernighan and Ritchie publish *The C Programming Language* — “K&R C”

**very** different from modern C

1989: ANSI standardizes C — C89/C90/ansi

compiler option: `-ansi`, `-std=c90`

looks mostly like modern C

36

## C evolution and standards

1978: Kernighan and Ritchie publish *The C Programming Language* — “K&R C”

**very** different from modern C

1989: ANSI standardizes C — C89/C90/ansi

compiler option: `-ansi`, `-std=c90`

looks mostly like modern C

1999: ISO (and ANSI) update C standard — C99

compiler option: `-std=c99`

adds: declare variables in middle of block

adds: `//` comments

36

## C evolution and standards

1978: Kernighan and Ritchie publish *The C Programming Language* — “K&R C”

very different from modern C

1989: ANSI standardizes C — C89/C90/ansi

compiler option: `-ansi`, `-std=c90`

looks mostly like modern C

1999: ISO (and ANSI) update C standard — C99

compiler option: `-std=c99`

adds: declare variables in middle of block

adds: `//` comments

2011: Second ISO update — C11

36

## Middle of blocks?

Examples of things not allowed in 1989 ANSI C:

```
printf("Before calling malloc()\n");  
int *pointer = malloc(sizeof(int) * 100);
```

```
for (int x = 0; x < 10; ++x)
```

37

## undefined behavior example (1)

```
#include <stdio.h>  
#include <limits.h>  
int test(int number) {  
    return (number + 1) > number;  
}  
  
int main(void) {  
    printf("%d\n", test(INT_MAX));  
}
```

38

## undefined behavior example (1)

```
#include <stdio.h>  
#include <limits.h>  
int test(int number) {  
    return (number + 1) > number;  
}  
  
int main(void) {  
    printf("%d\n", test(INT_MAX));  
}
```

without optimizations: 0

38



## undefined behavior example (1)

```
#include <stdio.h>
#include <limits.h>
int test(int number) {
    return (number + 1) > number;
}

int main(void) {
    printf("%d\n", test(INT_MAX));
}
```

without optimizations: 0

with optimizations: 1

38

## undefined behavior example (2)

```
int test(int number) {
    return (number + 1) > number;
}
```

Optimized:

```
test:
    movl    $1, %eax        ; eax ← 1
    ret
```

Less optimized:

```
test:
    leal   1(%rdi), %eax ; eax ← rdi + 1
    cmpl  %eax, %edi
    setl  %al           ; al ← eax < edi
    movzbl %al, %eax   ; eax ← al
    ret
```

39

## undefined behavior

compilers can do **whatever they want**

- what you expect
- crash your program
- ...

common types:

- signed integer overflow/underflow
- out-of-bounds pointers
- integer divide-by-zero
- writing read-only data
- out-of-bounds shift (later)

40

## undefined versus varies

size of types is not undefined behavior

even though it varies between compilers

undefined behavior

- is inconsistent — even within same program
- need not be documented

41

# Bit-twiddling

# some truth tables

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

&&, &

||, |

(nothing), ^

# Logical Operators

Treat value as true (1) or false (0)

Recall: false = 0 (only)

1	&&	1	==	1	1		1	==	1
2	&&	4	==	1	2		4	==	1
1	&&	0	==	0	1		0	==	1
0	&&	0	==	0	0		0	==	0
-1	&&	-2	==	1	-1		-2	==	1
""	&&	""	==	1	""		""	==	1

# Short-Circuit (&&)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() && one());
6     printf(">_%d\n", one() && zero());
7     return 0;
8 }
```

zero()

> 0

one()

zero()

> 0

AND	0	1
0	0	0
1	0	1

## Short-Circuit (&&)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() && one());
6     printf(">_%d\n", one() && zero());
7     return 0;
8 }
```

zero()

> 0	AND	0	1
one()	0	0	0
zero()	1	0	1
> 0			

45

## Short-Circuit (&&)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() && one());
6     printf(">_%d\n", one() && zero());
7     return 0;
8 }
```

zero()

> 0	AND	0	1
one()	0	0	0
zero()	1	0	1
> 0			

45

## Short-Circuit (&&)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() && one());
6     printf(">_%d\n", one() && zero());
7     return 0;
8 }
```

zero()

> 0	AND	0	1
one()	0	0	0
zero()	1	0	1
> 0			

45

## Short-Circuit (&&)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() && one());
6     printf(">_%d\n", one() && zero());
7     return 0;
8 }
```

zero()

> 0	AND	0	1
one()	0	0	0
zero()	1	0	1
> 0			

45

## Short-Circuit (||)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() || one());
6     printf(">_%d\n", one() || zero());
7     return 0;
8 }
```

zero()

one()

> 1

one()

> 1

OR	0	1
0	0	1
1	1	1

46

## Short-Circuit (||)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() || one());
6     printf(">_%d\n", one() || zero());
7     return 0;
8 }
```

zero()

one()

> 1

one()

> 1

OR	0	1
0	0	1
1	1	1

46

## Short-Circuit (||)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() || one());
6     printf(">_%d\n", one() || zero());
7     return 0;
8 }
```

zero()

one()

> 1

one()

> 1

OR	0	1
0	0	1
1	1	1

46

## Short-Circuit (||)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() || one());
6     printf(">_%d\n", one() || zero());
7     return 0;
8 }
```

zero()

one()

> 1

one()

> 1

OR	0	1
0	0	1
1	1	1

46

## Short-Circuit (||)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() || one());
6     printf(">_%d\n", one() || zero());
7     return 0;
8 }
```

zero()

one()

> 1

one()

> 1

OR	0	1
0	0	1
1	1	1

46

## Bitwise AND — &

Treat value as **array of bits**

1 & 1 == 1

1 & 0 == 0

0 & 0 == 0

2 & 4 == 0

10 & 7 == 2

47

## Bitwise AND — &

Treat value as **array of bits**

1 & 1 == 1

1 & 0 == 0

0 & 0 == 0

2 & 4 == 0

10 & 7 == 2

&	...	0	0	1	0
	...	0	1	0	0
<hr/>					
	...	0	0	0	0

47

## Bitwise AND — &

Treat value as **array of bits**

1 & 1 == 1

1 & 0 == 0

0 & 0 == 0

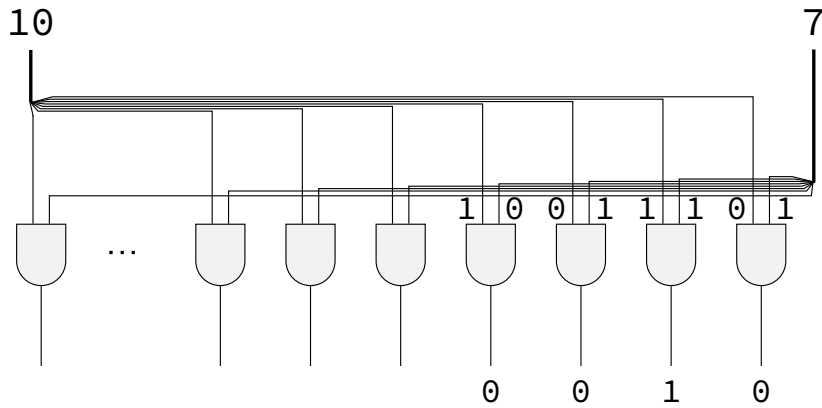
2 & 4 == 0

10 & 7 == 2

&	...	0	0	1	0
	...	0	1	0	0
<hr/>					
	...	0	0	0	0
	...	1	0	1	0
&	...	0	1	1	1
<hr/>					
	...	0	0	1	0

47

## Bitwise hardware (10 & 7 == 2)



48

## Bitwise OR — |

$$1 | 1 == 1$$

$$1 | 0 == 1$$

$$0 | 0 == 0$$

$$2 | 4 == 6$$

$$10 | 7 == 15$$

49

## Bitwise OR — |

$$1 | 1 == 1$$

$$1 | 0 == 1$$

$$0 | 0 == 0$$

$$2 | 4 == 6$$

$$10 | 7 == 15$$

$$\begin{array}{r} \dots 0 0 1 0 \\ | \dots 0 1 0 0 \\ \hline \dots 0 1 1 0 \end{array}$$

49

## Bitwise OR — |

$$1 | 1 == 1$$

$$1 | 0 == 1$$

$$0 | 0 == 0$$

$$2 | 4 == 6$$

$$10 | 7 == 15$$

$$\begin{array}{r} \dots 0 0 1 0 \\ | \dots 0 1 0 0 \\ \hline \dots 0 1 1 0 \\ \dots 1 0 1 0 \\ | \dots 0 1 1 1 \\ \hline \dots 1 1 1 1 \end{array}$$

49

## Bitwise xor — ^

$$1 \wedge 1 == 0$$

$$1 \wedge 0 == 1$$

$$0 \wedge 0 == 0$$

$$2 \wedge 4 == 6$$

$$10 \wedge 7 == 13$$

$$\begin{array}{r} \dots 0 \ 0 \ 1 \ 0 \\ \wedge \dots 0 \ 1 \ 0 \ 0 \\ \hline \dots 0 \ 1 \ 1 \ 0 \end{array}$$

$$\begin{array}{r} \dots 1 \ 0 \ 1 \ 0 \\ \wedge \dots 0 \ 1 \ 1 \ 1 \\ \hline \dots 1 \ 1 \ 0 \ 1 \end{array}$$

50

## Negation / Not — ~

~ ('complement') is bitwise version of !:

$$!0 == 1$$

$$!notZero == 0$$

$$\sim 0 == (\text{int}) \ 0xFFFFFFFF$$

32 bits

$$\begin{array}{r} \sim \ 0 \ 0 \ \dots \ 0 \ 0 \ 0 \ 0 \\ \hline 1 \ 1 \ \dots \ 1 \ 1 \ 1 \ 1 \end{array}$$

51

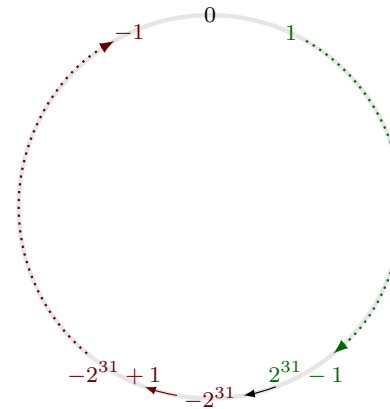
## Two's complement refresher

$$-1 = \begin{array}{ccccccc} -2^{31} & +2^{30} & +2^{29} & & +2^2 & +2^1 & +2^0 \\ 1 & 1 & 1 & \dots & 1 & 1 & 1 \end{array}$$

52

## Two's complement refresher

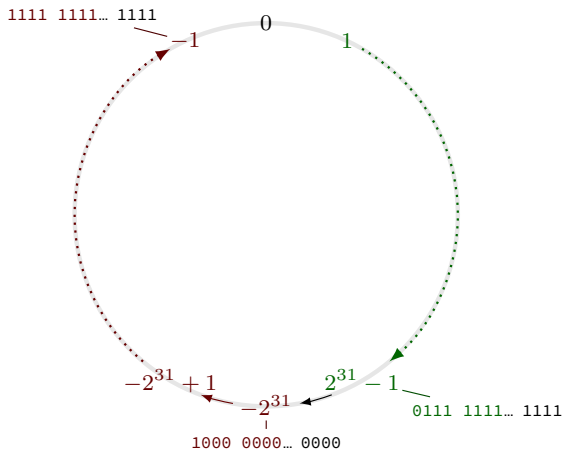
$$-1 = \begin{array}{ccccccc} -2^{31} & +2^{30} & +2^{29} & & +2^2 & +2^1 & +2^0 \\ 1 & 1 & 1 & \dots & 1 & 1 & 1 \end{array}$$



52

## Two's complement refresher

$$-1 = \overset{-2^{31} + 2^{30} + 2^{29}}{1} \overset{+2^2 + 2^1 + 2^0}{111} \dots 111$$



52

## Two's Complement and $\sim$

$$-x == \sim x + 1 \text{ (flip the bits and add one)}$$

53

## Two's Complement and $\sim$

$$-x == \sim x + 1 \text{ (flip the bits and add one)}$$

$$-x - 1 == \sim x$$

53

## Negation / Not — $\sim$

$\sim$  ('complement') is bitwise version of !:

$$!\text{0} == 1$$

$$!\text{notZero} == 0$$

$$\sim 0 == -1$$

$$\begin{array}{r} \sim \quad \text{32 bits} \\ \text{0} \text{---} \text{0} \text{---} \dots \text{0} \text{---} \text{0} \text{---} \text{0} \text{---} \text{0} \\ \hline \text{1} \text{---} \text{1} \text{---} \dots \text{1} \text{---} \text{1} \text{---} \text{1} \text{---} \text{1} \end{array}$$

54



## Negation / Not — ~

~ ('complement') is bitwise version of !:

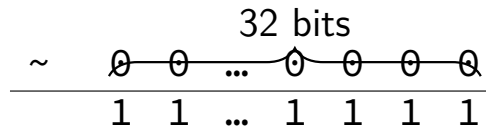
$$!0 == 1$$

$$!notZero == 0$$

$$\sim 0 == -1$$

$$\sim 2 == -3$$

$$\sim -7 == 6$$



54

## Negation / Not — ~

~ ('complement') is bitwise version of !:

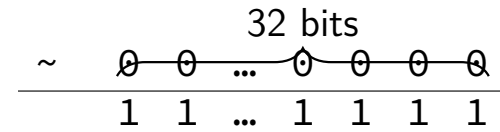
$$!0 == 1$$

$$!notZero == 0$$

$$\sim 0 == -1$$

$$\sim 2 == -3$$

$$\sim -7 == 6$$



$$\sim((\text{unsigned}) 2) == 0\text{xFFFFFFFD}$$

54

## Left shift

$$1 \ll 0 == 1$$

$$1 \ll 1 == 2$$

$$1 \ll 2 == 4$$

0000 0001

0000 0010

0000 0100

$$10 \ll 0 == 10$$

$$10 \ll 1 == 20$$

$$10 \ll 2 == 40$$

0000 1010

0001 0100

0010 1000

55

## Left shift

$$1 \ll 0 == 1$$

$$1 \ll 1 == 2$$

$$1 \ll 2 == 4$$

0000 0001

0000 0010

0000 0100

$$10 \ll 0 == 10$$

$$10 \ll 1 == 20$$

$$10 \ll 2 == 40$$

0000 1010

0001 0100

0010 1000

$$x \ll y = x \times 2^y$$

55

## Right shift

Undefined:  ~~$x \lll = 1$~~  Instead:  $x \gg 1$

$1 \gg 0 == 1$             0000 0001  
 $1 \gg 1 == 0$             0000 0000  
 $1 \gg 2 == 0$             0000 0000

$10 \gg 0 == 10$         0000 1010  
 $10 \gg 1 == 5$          0000 0101  
 $10 \gg 2 == 2$          0000 0010

56

## Right shift

Undefined:  ~~$x \lll = 1$~~  Instead:  $x \gg 1$

$1 \gg 0 == 1$             0000 0001  
 $1 \gg 1 == 0$             0000 0000  
 $1 \gg 2 == 0$             0000 0000

$10 \gg 0 == 10$         0000 1010  
 $10 \gg 1 == 5$          0000 0101  
 $10 \gg 2 == 2$          0000 0010

$$x \gg y = \lfloor x \times 2^{-y} \rfloor$$

56

## Shifts and negative numbers

$-10 \gg 1 == ???$  ( $-10 = 1111 \dots 1111 0110$ )

binary ?111 ... 1111 1011

57

## Shifts and negative numbers

$-10 \gg 1 == ???$  ( $-10 = 1111 \dots 1111 0110$ )

binary ?111 ... 1111 1011

Option 1: binary 1111 ... 1011 =

$$-5 = -10 \times 2^{-k}$$

copy sign bit

Option 2: binary 0111 ... 1011 =  $2^{31} - 5$

always use zero

57

## Shifts and negative numbers

$-10 \gg 1 == ???$  ( $-10 = 1111 \dots 1111 0110$ )

binary  $?111 \dots 1111 1011$

Option 1: binary  $1111 \dots 1011 =$

$$-5 = -10 \times 2^{-k}$$

copy sign bit  
arithmetic

Option 2: binary  $0111 \dots 1011 = 2^{31} - 5$

always use zero  
logical

57

## Shifts and negative numbers

$-10 \gg 1 == ???$  ( $-10 = 1111 \dots 1111 0110$ )

binary  $?111 \dots 1111 1011$

Option 1: binary  $1111 \dots 1011 =$

$$-5 = -10 \times 2^{-k}$$

copy sign bit  
arithmetic

Option 2: binary  $0111 \dots 1011 = 2^{31} - 5$

always use zero  
logical

57

## Aside: implementation-defined behavior

C standard: negative  $\gg 1$  is

implementation-defined

compiler chooses

58

## Arithmetic shift

$-1 \gg 0 == -1$                       1111 1111

$-1 \gg 1 == -1$                       1111 1111

$-1 \gg 2 == -1$                       1111 1111

$-10 \gg 0 == -10$                     1111 0110

$-10 \gg 1 == -5$                      1111 1011

$-10 \gg 2 == -3$                      1111 1101

$10 \gg 2 == 2$                         0000 0010

$$x \gg y = \lfloor x \times 2^{-y} \rfloor$$

59

## Arithmetic shift

```
-1 >> 0 == -1      1111 1111
-1 >> 1 == -1      1111 1111
-1 >> 2 == -1      1111 1111
```

```
-10 >> 0 == -10   1111 0110
-10 >> 1 == -5    1111 1011
-10 >> 2 == -3    1111 1101
```

```
10 >> 2 == 2      0000 0010
```

$$x \gg y = \lfloor x \times 2^{-y} \rfloor$$

59

## signed versus unsigned types

```
/*signed*/ int x = -10;
/* arithmetic: */
x >> 1 == -5
x >> 4 == -1
```

```
unsigned int y = 0xFFFFFFFF6;
/* logical */
y >> 1 == 0x7FFFFFFB
y >> 4 == 0x0FFFFFFF
```

60

## Sign-extension vs. zero-extension

```
signed char x = -10; //      1111 0110
int y = x;           // 1111.. 1111 0110
printf("%d\n", y);  // outputs "-10"
```

```
unsigned char x = 0xF6; //      1111 0110
int y = x;             // 0000.. 1111 0110
printf("%d\n", y);    // outputs "246"
```

61

## Aside: integer promotions

```
unsigned short number = 1;
unsigned short offset = 20;
printf("0x%x\n", number << offset);
```

Outputs (on lab machines)?

- A. 0x100000 ( $2^{20}$ )
- B. 0x0
- C. Undefined behavior — varies

62

## Aside: integer promotions

```
unsigned short number = 1;
unsigned short offset = 20;
printf("0x%x\n", number << offset);
```

Outputs (on lab machines)?

- A.** 0x100000 ( $2^{20}$ )                      **B.** 0x0  
**C.** Undefined behavior — varies

62

## Aside: integer promotions

```
unsigned short number = 1;
unsigned short offset = 20;
printf("0x%x\n", number << offset);
```

Outputs (on lab machines)?

- A.** 0x100000 ( $2^{20}$ )                      **B.** 0x0  
**C.** Undefined behavior — varies

integers types smaller than **int** converted to **int**

62

## Shifts: undefined behavior

0 >> 32 is **undefined behavior**

0 << 32 is **undefined behavior**

(**long**) 0 << 32 is okay

(**long**) 0 << 64 is **undefined behavior**

63

## Summary

**struct** — functionless classes

**typedef struct** or write **struct** typeName

malloc, free instead of new/delete.

undefined behavior — who knows what'll happen

logical operators — &&, ||, !: only care if 0/not 0

bitwise operators — &, |, ^, ~: all bits in parallel

64

## Summary

**struct** — functionless classes

**typedef struct** or write **struct** typeName

malloc, free instead of new/delete.

undefined behavior — who knows what'll happen

logical operators — &&, ||, !: only care if 0/not 0

bitwise operators — &, |, ^, ~: all bits in parallel

bitshift — <<, >>: same as multiplying by  $2^{\pm x}$

arithmetic right shift — borrow sign bit

64

## Backup Slides

65

## What's in those files?

hello.c

```
#include <stdio.h>
int main(void) {
    puts("Hello, World!");
    return 0;
}
```

66

## What's in those files?

hello.c

```
#include <stdio.h>
int main(void) {
    puts("Hello, World!");
    return 0;
}
```

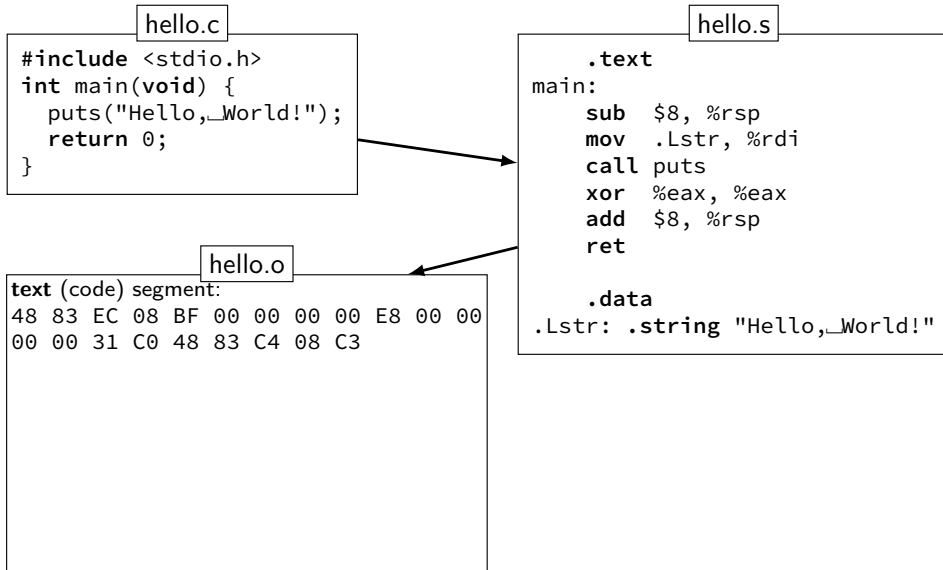
hello.s

```
.text
main:
    sub    $8, %rsp
    mov    .Lstr, %rdi
    call  puts
    xor    %eax, %eax
    add    $8, %rsp
    ret

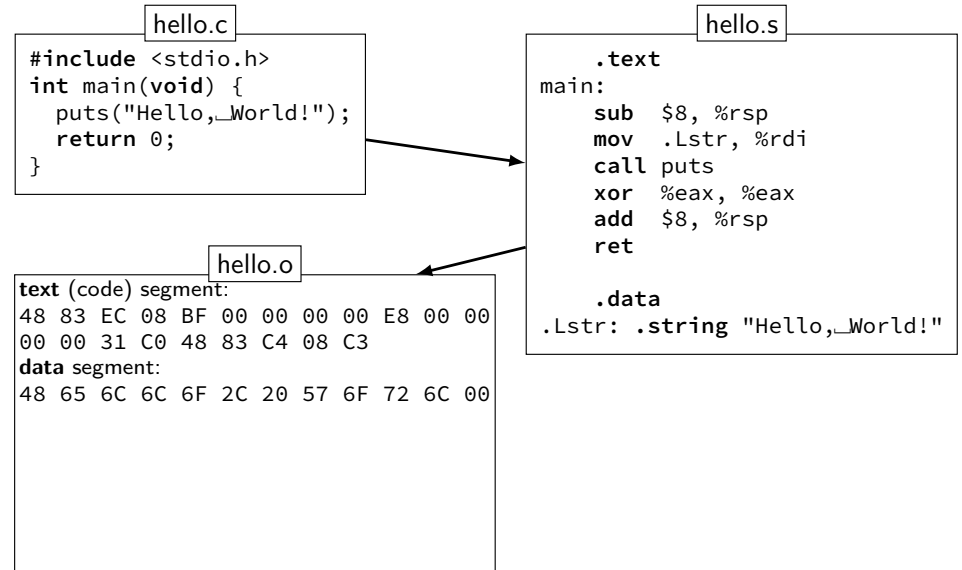
.data
.Lstr: .string "Hello, World!"
```

66

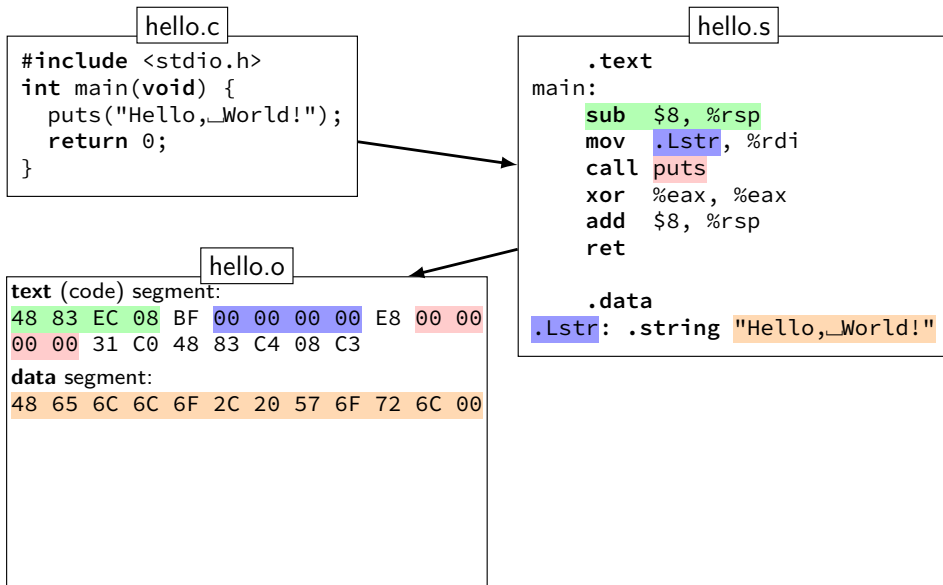
# What's in those files?



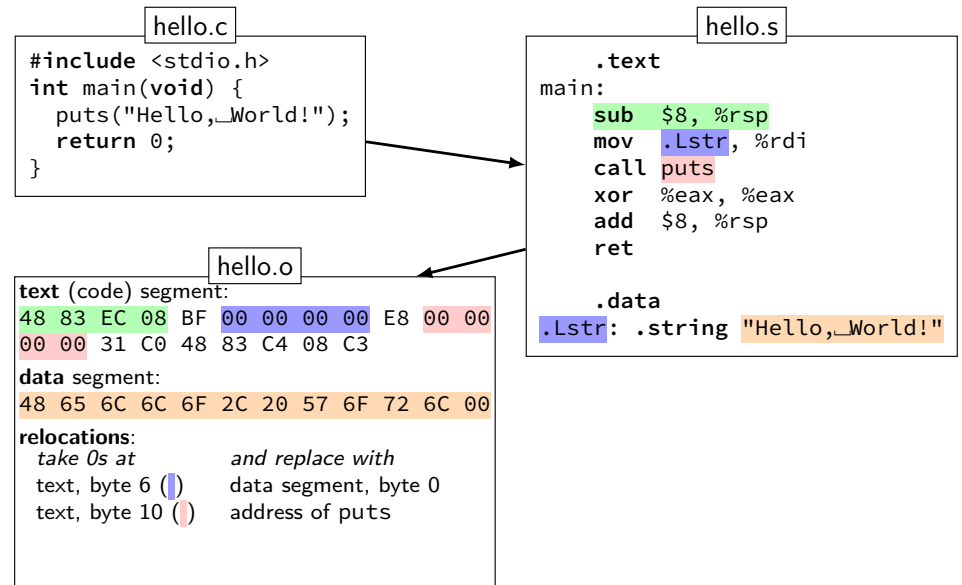
# What's in those files?



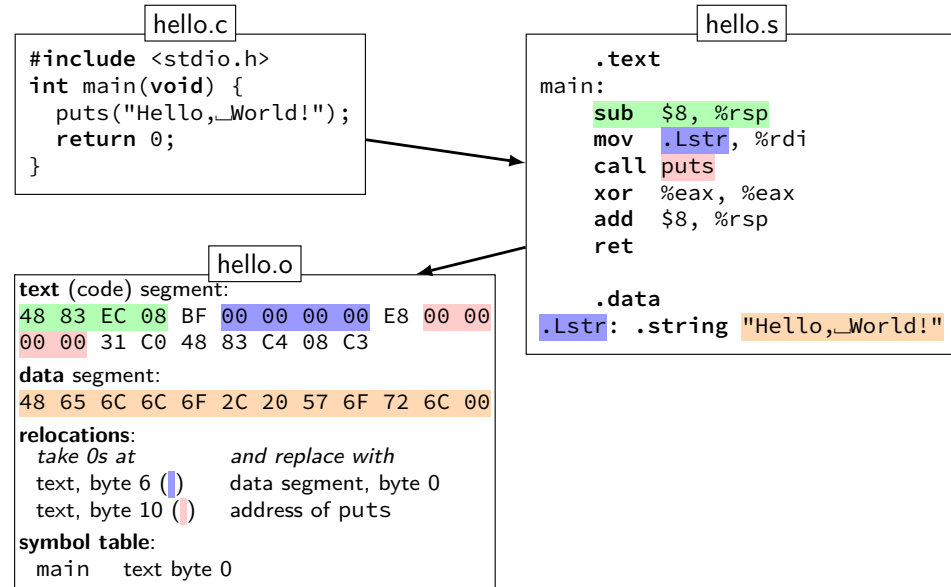
# What's in those files?



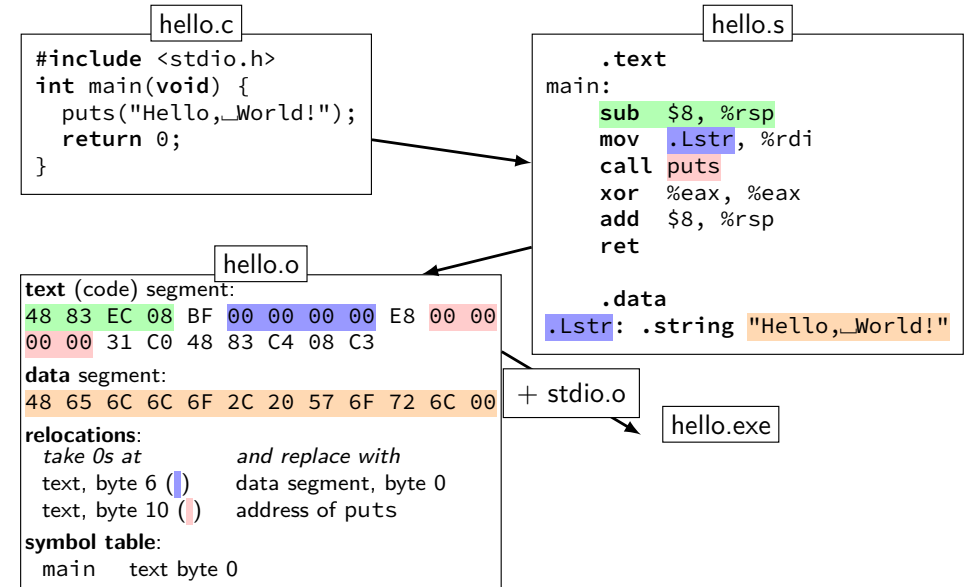
# What's in those files?



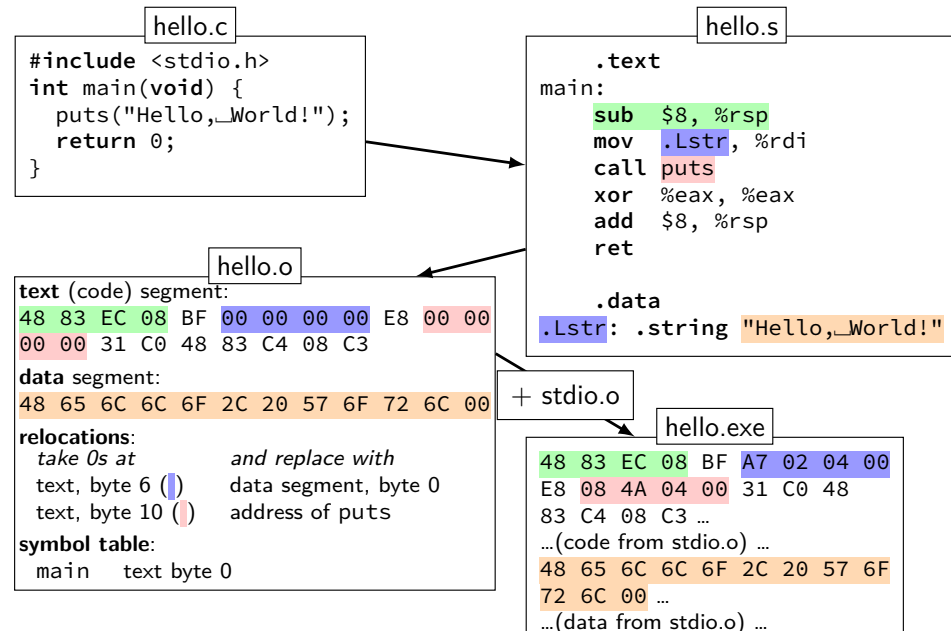
# What's in those files?



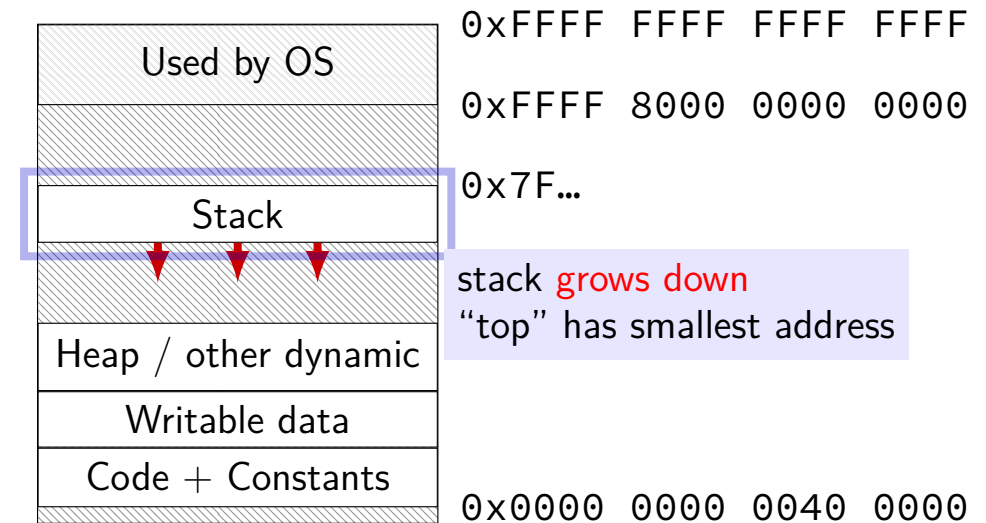
# What's in those files?



# What's in those files?



# Program Memory (x86-64 Linux)





## Arrays: not quite pointers (1)

```
int array[100];  
int *pointer;
```

Legal: `pointer = array;`  
same as `pointer = &(array[0]);`

68

## Arrays: not quite pointers (1)

```
int array[100];  
int *pointer;
```

Legal: `pointer = array;`  
same as `pointer = &(array[0]);`

Illegal: ~~`array = pointer;`~~

68

## Arrays: not quite pointers (2)

```
int array[100];  
int *pointer = array;
```

`sizeof(array) == 400`  
size of all elements

69

## Arrays: not quite pointers (2)

```
int array[100];  
int *pointer = array;
```

`sizeof(array) == 400`  
size of all elements

`sizeof(pointer) == 8`  
size of address

69

## Arrays: not quite pointers (2)

```
int array[100];  
int *pointer = array;
```

```
sizeof(array) == 400  
    size of all elements
```

```
sizeof(pointer) == 8  
    size of address
```

```
sizeof(&array[0]) == ???  
    (&array[0] same as &(array[0]))
```

69

## chmod

```
chmod --recursive og-r /home/USER
```

70

## chmod

```
chmod --recursive og-r /home/USER
```

others and group (student)  
- remove  
read

70

## chmod

```
chmod --recursive og-r /home/USER
```

user (yourself) / group / others  
- remove / + add  
read / write / execute or search

70

## tar

the standard Linux/Unix file archive utility

Table of contents: `tar tf filename.tar`

eXtract: `tar xvf filename.tar`

Create: `tar cvf filename.tar directory`

(v: verbose; f: file — default is tape)