CS 3330: C

25 August 2016

Layers of Abstraction

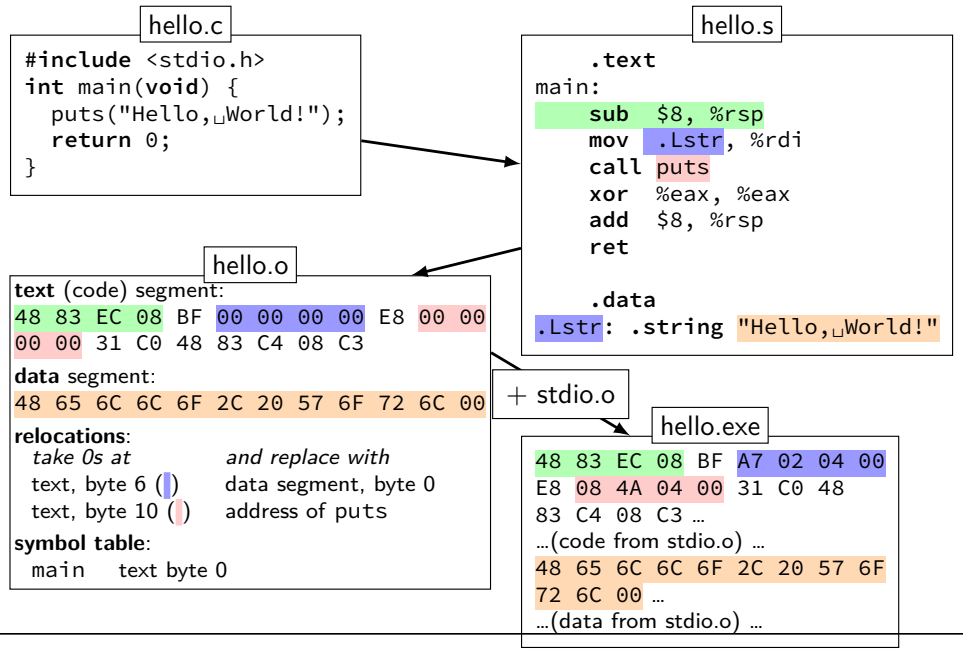
`x += y`	"Higher-level" language: C
`add %rbx, %rax`	Assembly: X86-64
60 03	Machine code: Y86
	(we'll talk later)
	Logic and Registers

Compilation Steps

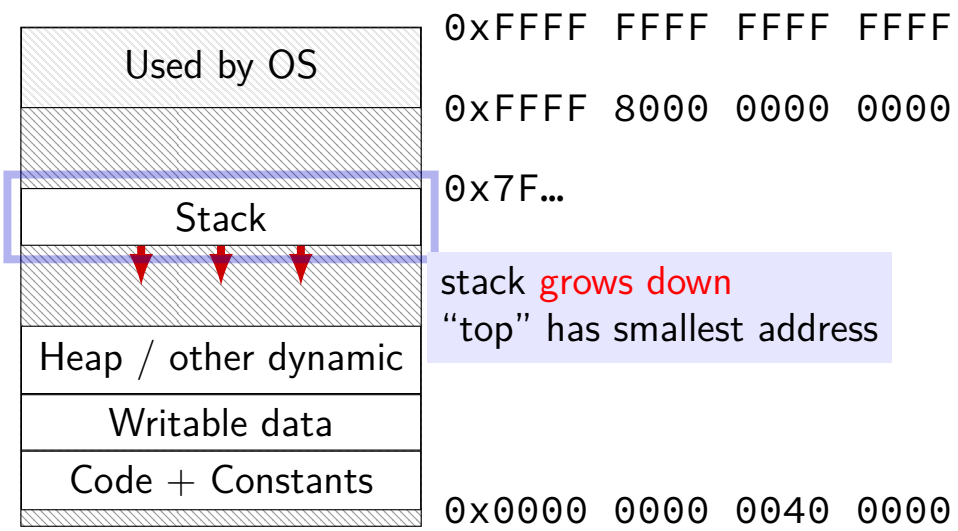
compile:	`gcc -S file.c`	⇒	`file.s`
assemble:	`gcc -c file.s`	⇒	`file.o`
link:	`gcc -o file file.o`	⇒	`file` (exec.)
c+a:	`gcc -c file.c`	⇒	`file.o`
c+a+l:	`gcc -o file file.c`	⇒	`file` (exec.)

...

What's in those files?



Program Memory (x86-64 Linux)



C Data Types

Varies between machines(!). For this course:

type	size (bytes)
`char`	1
`short`	2
`int`	4
`long`	8
`float`	4
`double`	8
`void *`	8
`anything *`	8

Truth

~~`bool`~~

`x == 4` is an **`int`**

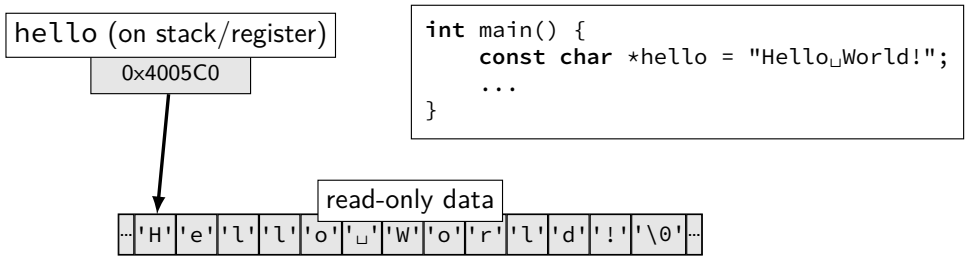
`1` if true; `0` if false

False values in C

`0`

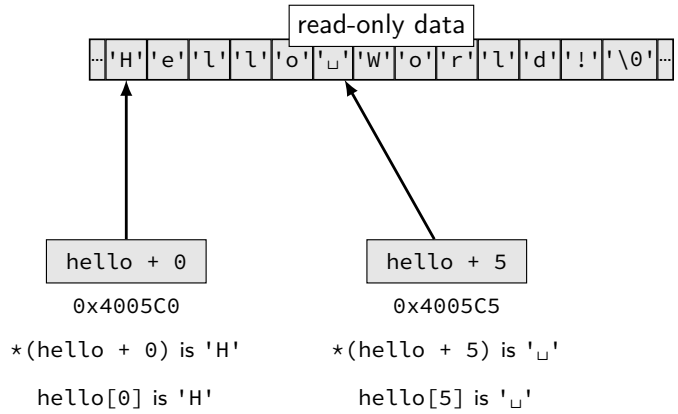
including null pointers — `0` cast to a pointer

Strings in C




```
int main() {
    const char *hello = "Hello␣World!";
    ...
}
```


Pointer Arithmatic



Arrays and Pointers

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

arrays 'decay' into pointers

Exercise

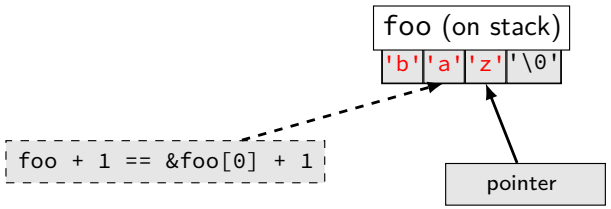
```
1 char foo[4] = "foo";
2     // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';
8 *(foo + 1) = 'a';
```

Final value of `foo`?

A. `"fao"`
B. `"zao"`
C. `"baz"`
D. `"bao"`
E. something else/crash

Exercise explanation

```
1 char foo[4] = "foo";
2     // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';    better style: *pointer = 'z';
8 *(foo + 1) = 'a';    better style: foo[1] = 'a';
```



Arrays of non-bytes

`array[2]` and `*(array + 2)` still the same

```
1 int numbers[4] = {10, 11, 12, 13};
2 int *pointer;
3 pointer = numbers;
4 *pointer = 20;  // numbers[0] = 20;
5 pointer = pointer + 2;
6 /* adds 8 (2 ints) to address */
7 *pointer = 30;  // numbers[2] = 30;
8 // numbers is {20, 11, 30, 13}
```


Arrays: not quite pointers (1)

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

Legal: `pointer = array;`

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

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

Arrays: not quite pointers (2)

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

`sizeof(array) == 400`

size of all elements

`sizeof(pointer) == 8`

size of address

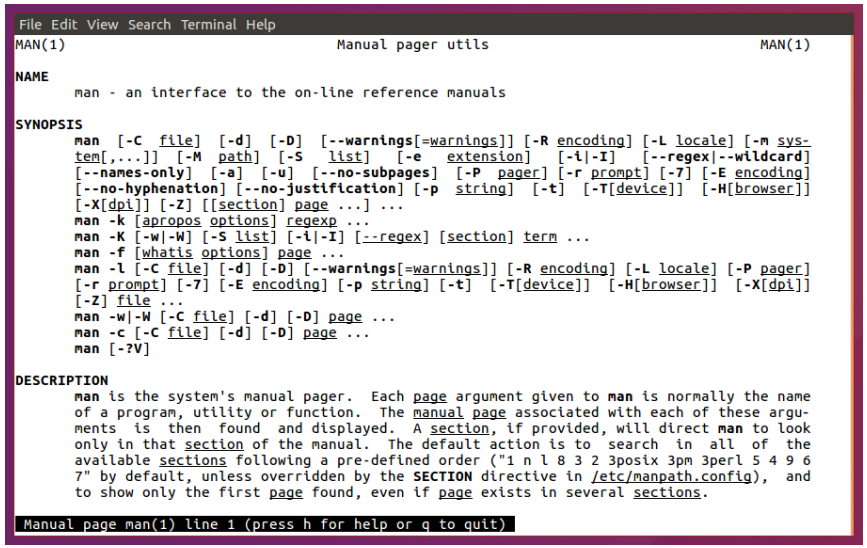
`sizeof(&array[0]) == ???`

(`&array[0]` same as `&(array[0])`)

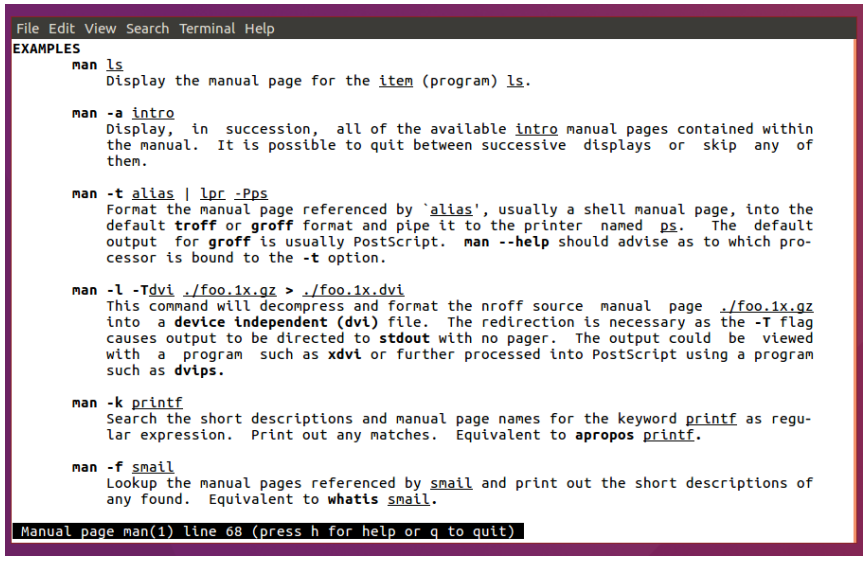
Interlude: Command Line Tips

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

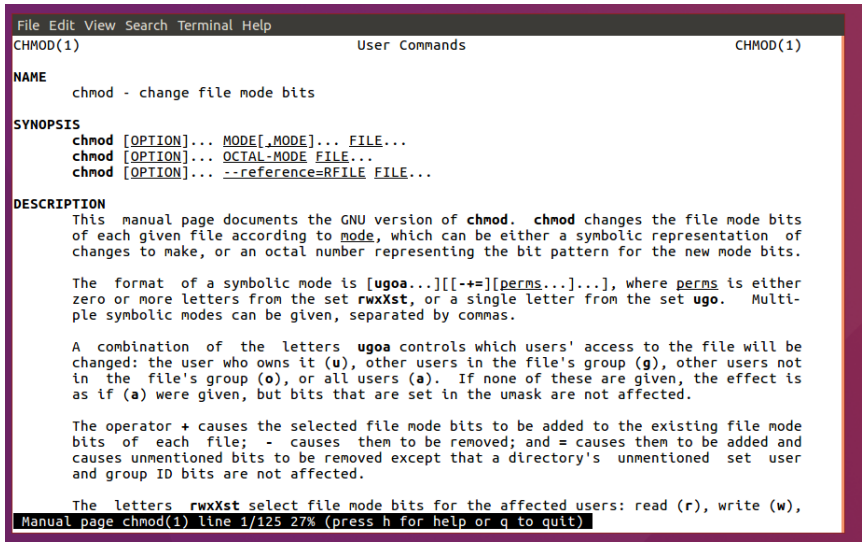

man man



man man



man chmod



chmod

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

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

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)

Tab completion and history

Back To C

stdio.h

C does not have `<iostream>`

Instead `<stdio.h>`

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.
```


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

…
```


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

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)	%

goto

```
for (...) {
  for (...) {
    if (thingAt(i, j
      goto found;
    }
  }
}
printf("not found!\n");
return;
found:
printf("found!\n");
```

assembly:
`jmp found`

assembly:
`found:`