more C / assembly intro

last time

program memory layout

- stack versus heap versus code/globals
- at fixed locations (by convention only)

compile/assemble/link

- object file
  - machine code + data (from assembly)
  - placeholders for addresses (labels in assembly)
- linking
  - decide where in memory object files go
  - fix placeholders

pointer arithmetic — how arrays in C work

- treat pointer as a number, add/etc.

anonymous feedback (1)

(paraphrased) I like learning from videos; can you suggest some for this course?

- other comp. arch. courses (e.g. CMU, Georgia Tech)? — but not same topics/instruction set
- for particular topics (pipelining, virtual memory) can probably find some

anonymous feedback (2)

“In the slides, what is said in class, and the questions asked on the quiz, I have seen/heard the content presented pretty ambiguously. As a native English speaker, I’ve had trouble interpreting what is said or being asked, so I can’t imagine the trouble a non-native speaker could be having. This may be the reason so many clarification questions are being asked in class. Can you try to be a little more clear and precise about what you intend to portray?”

- I believe you!
- ...but this isn’t specific enough to help me

anonymous feedback (3)

(paraphrased) can quizzes be due on Tuesday?

intent is when we have reading quizzes:
- quiz on lecture/lab material done Thurs-Sat
- quiz on reading done Sun-Tues

currently: no reading quiz

but want to get you used to when lecture/lab quizzes are

on the quiz (1)

“In addition to the machine code itself, the object file contains information about where in the machine code memory addresses will be when the program runs”
- relocations (placeholders) say “linker, put a memory address here”
- so object file must say where memory address will eventually be

“the object file contains the names of labels from the corresponding assembly file”
- symbol table
- object files refer to things from other object files by label name

on the quiz (2)

“in addition to the machine code itself, the object file contains information about where each instruction starts in the machine code”
- can tell from machine code itself, but...
- only care about labels/placeholders — not every instruction
- I should have bold+italiced ***each***

endianness question — yes, would’ve been better if I said “in memory” (referring to how arrays work in memory)

on the quiz (3)

in comments, please don’t refer to the randomized answer letters

lab this week

you will download an 64-bit Linux executable

use debugger, other tools to figure out what input it expects

note: tools output AT&T syntax assembly by default

- lab writeup mentions options to get Intel syntax assembly instead

in theory: just 2150 stuff?

in practice: we'll be reviewing assembly

arrays and pointers

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

arrays 'decay' into pointers

arrays of non-bytes

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

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

arrays of non-bytes

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

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

arrays: not quite pointers (1)

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

Legal: `pointer = array;`

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

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

arrays: not quite pointers (2)

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

`sizeof(array) == 400`

size of all elements

`sizeof(pointer) == 8`

size of address

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])`)

a note on precedence

`&foo[1]` is the same as `&(foo[1])` (*not* `(&foo)[1]`)

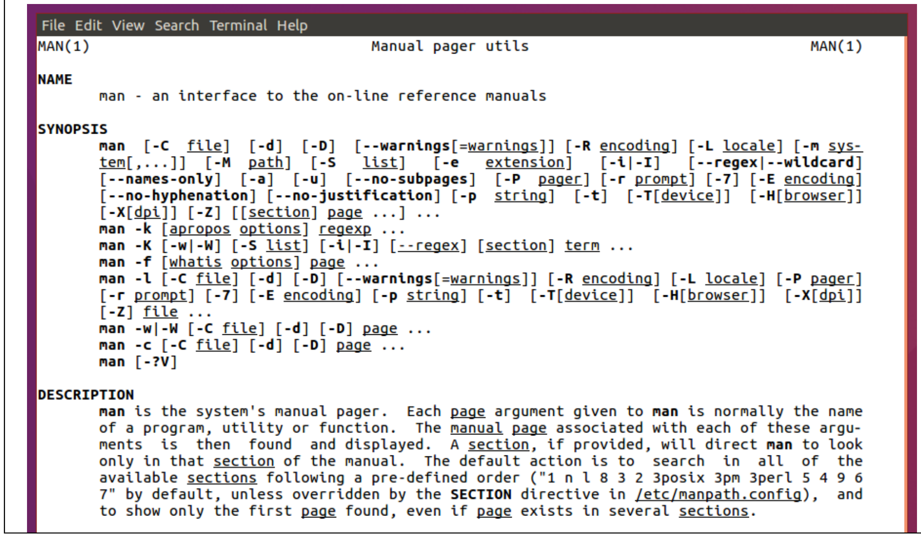
`*foo[0]` is the same as `*(foo[0])` (*not* `(*foo)[0]`)

`*foo++` is the same as `*(foo++)` (*not* `(*foo)++`)

interlude: command line tips

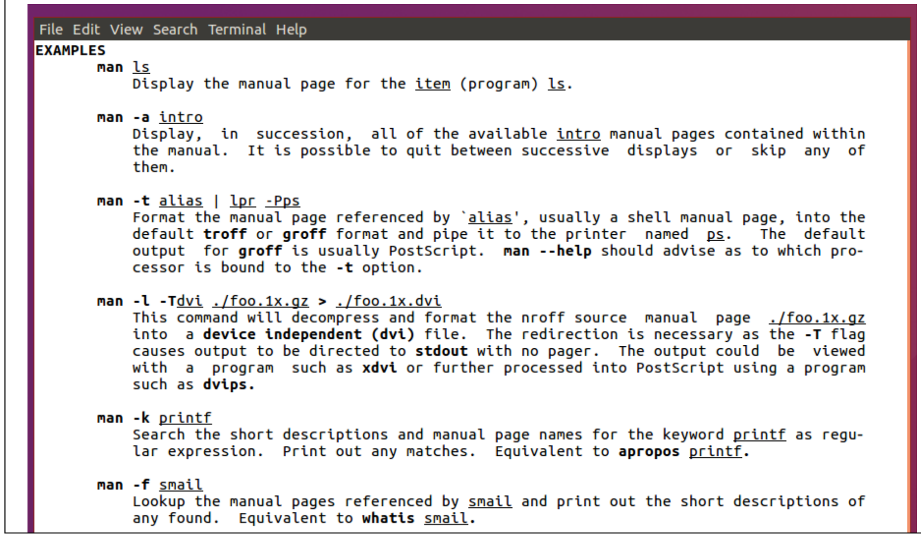
```
cr4bd@reiss-lenovo:~$ man man
```

man man



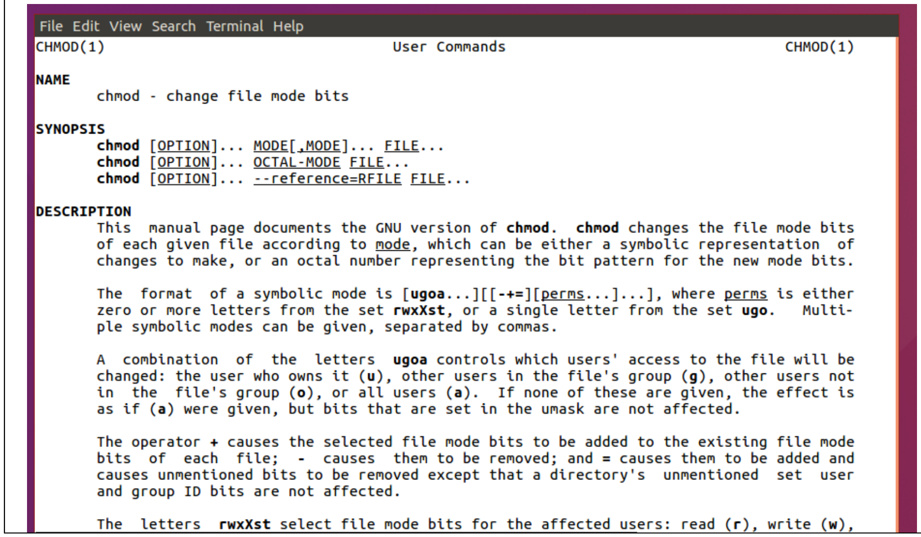
```
MAN(1)                        Manual pager utils                       MAN(1)

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.
```

man man



```
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.
```

man chmod



```
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 rwxXst, 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 umask 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  rwxXst select file mode bits for the affected users: read (r), write (w),
```

chmod

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

chmod

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

others and group (student)

– remove

read

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;
```

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;
```

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)	`%`

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		
%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`

goto

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

goto

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

assembly:
`jmp found`

assembly:
`found:`

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);
```

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);
```

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);
```

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);
```

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;
```

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;
```

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;
```

linked lists / dynamic allocation

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

linked lists / dynamic allocation

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

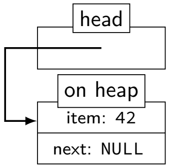
linked lists / dynamic allocation

```
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 */
```

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 */
```



head

on heap

item: 42

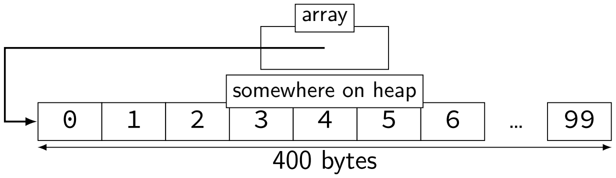
next: NULL

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
```

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
```



array

somewhere on heap

0 1 2 3 4 5 6 … 99

400 bytes

miss vector? (1)

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

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);
```

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;
```

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`?

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	$2^{64}-1$

⋮

unsigned/signed comparison trap (1)

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

unsigned/signed comparison trap (1)

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

result is `0`

unsigned/signed comparison trap (1)

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

result is `0`

short solution: don't compare signed to unsigned:

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

unsigned/sign comparison trap (2)

```c
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`

C evolution and standards

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

- very different from modern C

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

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

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

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));
}
```

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`

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

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 (pad with zeros)
    ret
```

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)

undefined behavior

why undefined behavior?

different architectures work differently

- allow compilers to expose whatever processor does “naturally”
- don’t encode any particular machine in the standard

flexibility for optimizations

more C later

bitwise operators

- after we talk about assembly a bit
- we’ll maybe have more of a use-case

x86-64 manuals

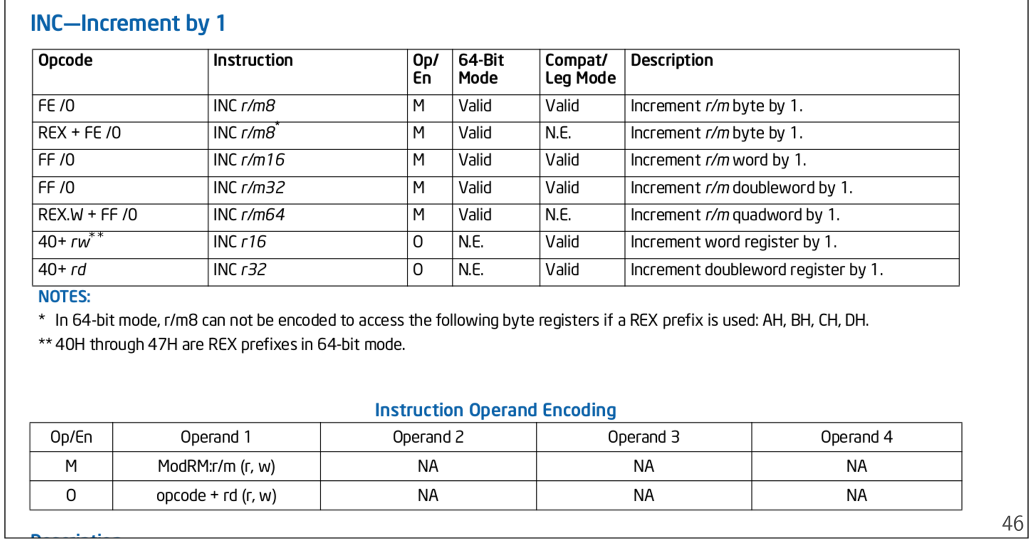
Intel manuals:

- `https://software.intel.com/en-us/articles/intel-sdm`
- 24 MB, 4684 pages
- Volume 2: instruction set reference (2190 pages)

AMD manuals:

- `https://support.amd.com/en-us/search/tech-docs`
- “AMD64 Architecture Programmer’s Manual”

example manual page



INC—Increment by 1

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
FE /0	INC *r/m8*	M	Valid	Valid	Increment *r/m* byte by 1.
REX + FE /0	INC *r/m8**	M	Valid	N.E.	Increment *r/m* byte by 1.
FF /0	INC *r/m16*	M	Valid	Valid	Increment *r/m* word by 1.
FF /0	INC *r/m32*	M	Valid	Valid	Increment *r/m* doubleword by 1.
REX.W + FF /0	INC *r/m64*	M	Valid	N.E.	Increment *r/m* quadword by 1.
40+ *rw***	INC *r16*	O	N.E.	Valid	Increment word register by 1.
40+ *rd*	INC *r32*	O	N.E.	Valid	Increment doubleword register by 1.

NOTES:

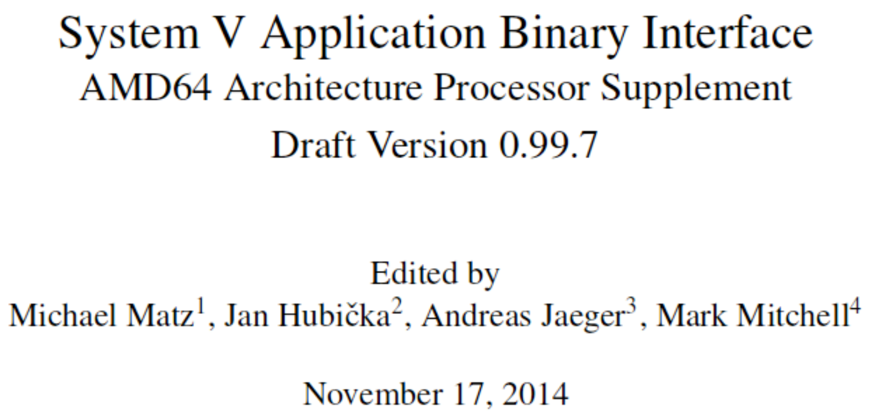
* In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

** 40H through 47H are REX prefixes in 64-bit mode.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r, w)	NA	NA	NA
O	opcode + rd (r, w)	NA	NA	NA

Linux x86-64 calling convention



System V Application Binary Interface

AMD64 Architecture Processor Supplement

Draft Version 0.99.7

Edited by

Michael Matz[1], Jan Hubička[2], Andreas Jaeger[3], Mark Mitchell[4]

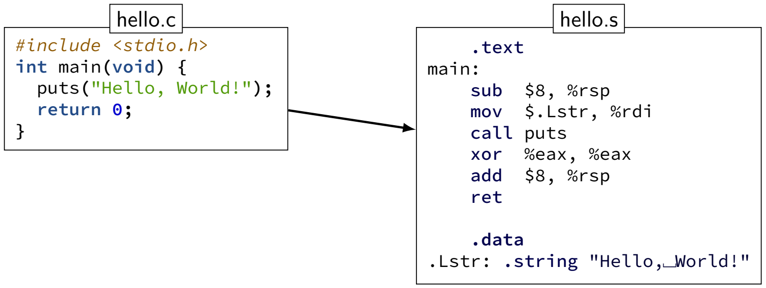
November 17, 2014

what's in those files?

hello.c

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

what's in those files?



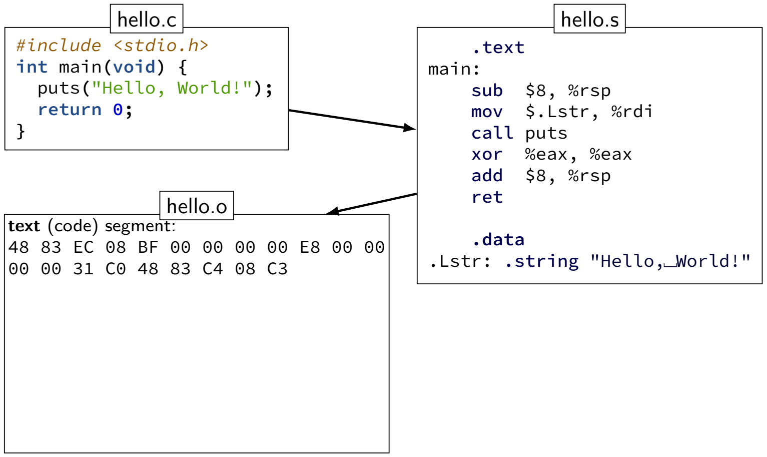
hello.c

```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!"
```

what's in those files?



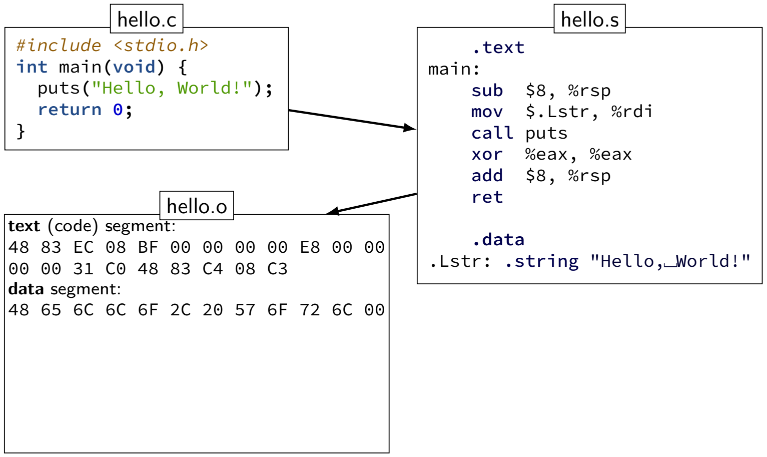
hello.c
```
#include <stdio.h>
int main(void) {
  puts("Hello, World!");
  return 0;
}
```

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

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

hello.o

text (code) segment:
```
48 83 EC 08 BF 00 00 00 00 E8 00 00
00 00 31 C0 48 83 C4 08 C3
```

what's in those files?



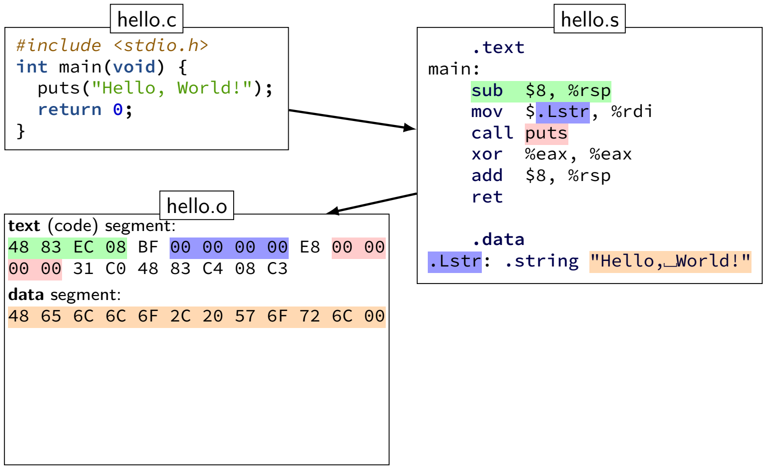
hello.c
```
#include <stdio.h>
int main(void) {
  puts("Hello, World!");
  return 0;
}
```

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

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

hello.o

text (code) segment:
```
48 83 EC 08 BF 00 00 00 00 E8 00 00
00 00 31 C0 48 83 C4 08 C3
```
data segment:
```
48 65 6C 6C 6F 2C 20 57 6F 72 6C 00
```

what's in those files?



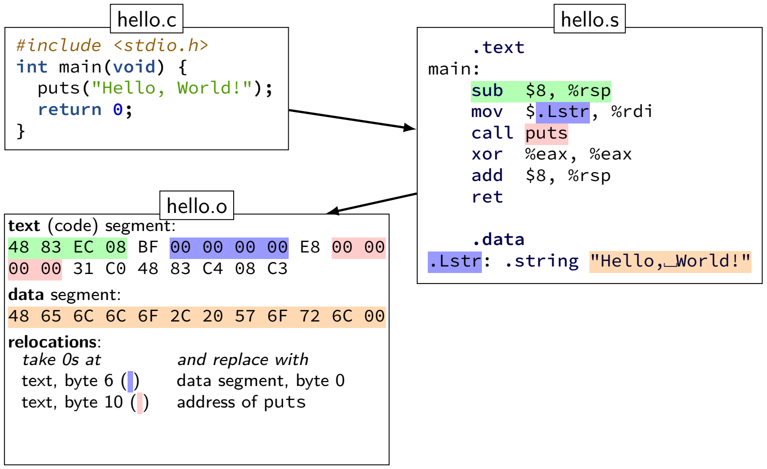
hello.c
```
#include <stdio.h>
int main(void) {
  puts("Hello, World!");
  return 0;
}
```

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

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

hello.o

text (code) segment:
```
48 83 EC 08 BF 00 00 00 00 E8 00 00
00 00 31 C0 48 83 C4 08 C3
```
data segment:
```
48 65 6C 6C 6F 2C 20 57 6F 72 6C 00
```

what's in those files?



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

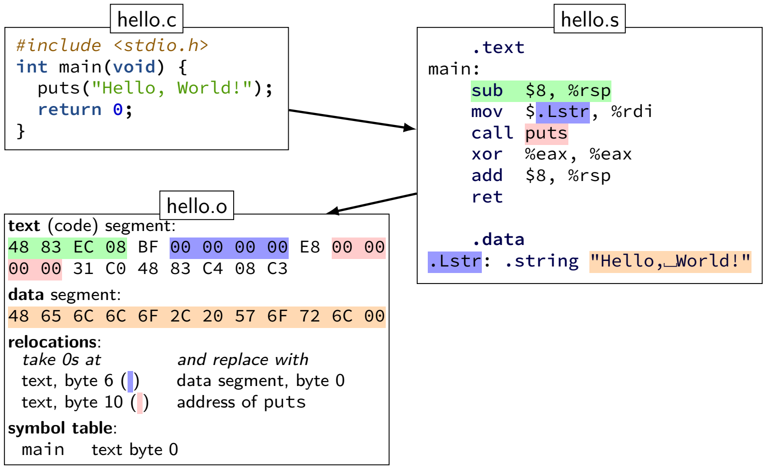
hello.s
```
    .text
main:
    sub  $8, %rsp
    mov  $.Lstr, %rdi
    call puts
    xor  %eax, %eax
    add  $8, %rsp
    ret

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

hello.o

text (code) segment:
```
48 83 EC 08 BF 00 00 00 00 E8 00 00
00 00 31 C0 48 83 C4 08 C3
```
data segment:
```
48 65 6C 6C 6F 2C 20 57 6F 72 6C 00
```
relocations:

take 0s at	*and replace with*
text, byte 6 (🟦)	data segment, byte 0
text, byte 10 (🟥)	address of puts

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!"
```

hello.o

text (code) segment:

```
48 83 EC 08 BF 00 00 00 00 E8 00 00
00 00 31 C0 48 83 C4 08 C3
```

data segment:

```
48 65 6C 6C 6F 2C 20 57 6F 72 6C 00
```

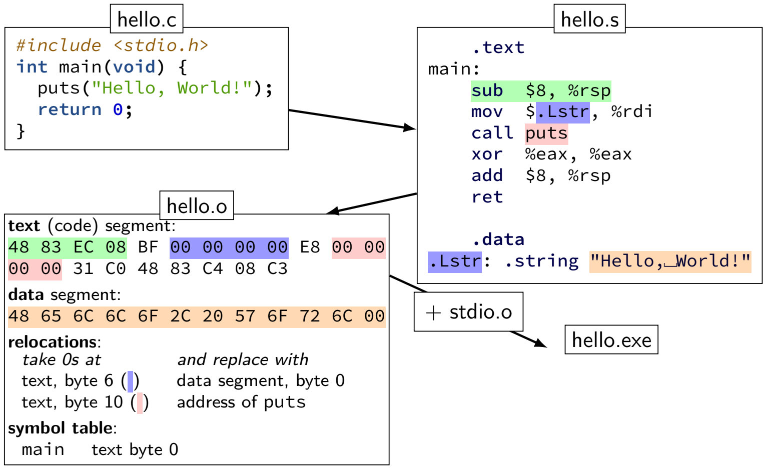
relocations:

take 0s at	*and replace with*
text, byte 6 (🟦)	data segment, byte 0
text, byte 10 (🟥)	address of puts

symbol table:

main text byte 0

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!"
```

hello.o

text (code) segment:

```
48 83 EC 08 BF 00 00 00 00 E8 00 00
00 00 31 C0 48 83 C4 08 C3
```

data segment:

```
48 65 6C 6C 6F 2C 20 57 6F 72 6C 00
```

relocations:

take 0s at	*and replace with*
text, byte 6 (🟦)	data segment, byte 0
text, byte 10 (🟥)	address of puts

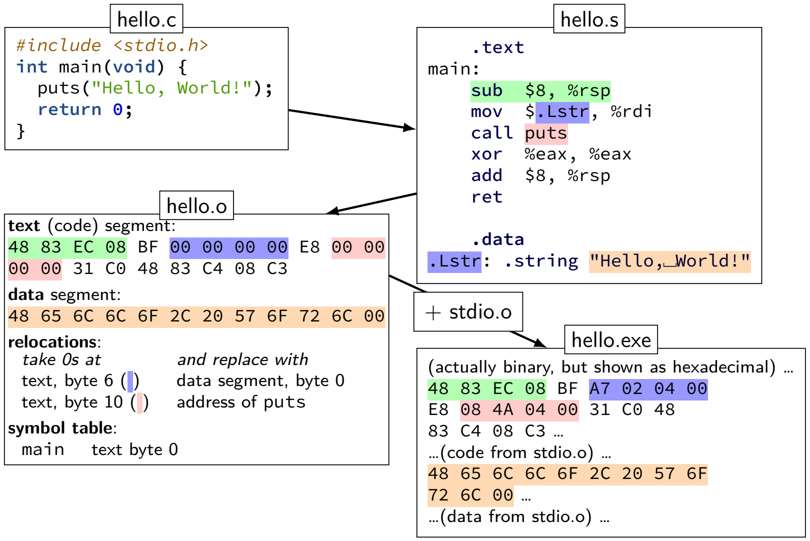
symbol table:

main text byte 0

+ stdio.o

hello.exe

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!"
```

hello.o

text (code) segment:

```
48 83 EC 08 BF 00 00 00 00 E8 00 00
00 00 31 C0 48 83 C4 08 C3
```

data segment:

```
48 65 6C 6C 6F 2C 20 57 6F 72 6C 00
```

relocations:

take 0s at	*and replace with*
text, byte 6 (🟦)	data segment, byte 0
text, byte 10 (🟥)	address of puts

symbol table:

main text byte 0

+ stdio.o

hello.exe

(actually binary, but shown as hexadecimal) …

```
48 83 EC 08 BF A7 02 04 00
E8 08 4A 04 00 31 C0 48
83 C4 08 C3 …
```

…(code from stdio.o) …

```
48 65 6C 6C 6F 2C 20 57 6F
72 6C 00 …
```

…(data from stdio.o) …

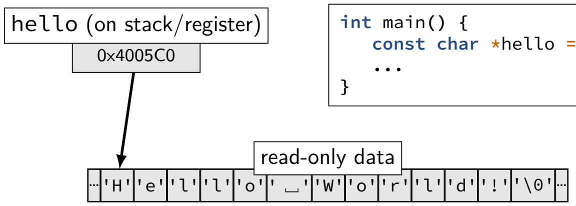
hello.s

```
        .section        .rodata.str1.1,"aMS",@progb
.LC0:
        .string "Hello, World!"
        .text
        .globl  main
main:
        subq    $8, %rsp
        movl    $.LC0, %edi
        call    puts
        movl    $0, %eax
        addq    $8, %rsp
        ret
```

hello.o

```
hello.o:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000000 g     F .text  0000000000000018 mai
0000000000000000         *UND*  0000000000000000 put

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
0000000000000005 R_X86_64_32       .rodata.str1.1
000000000000000a R_X86_64_PC32     puts-0x000000000

Contents of section .text:
 0000 4883ec08 bf000000 00e80000 0000b800  H........
 0010 00000048 83c408c3                    H
```

strings in C

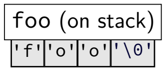


hello (on stack/register)
0x4005C0
read-only data
…'H''e''l''l''o''␣''W''o''r''l''d''!''\0'…

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

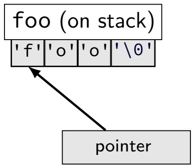
exercise explanation

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



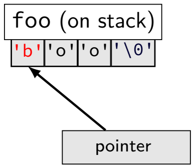
exercise explanation

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



exercise explanation

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



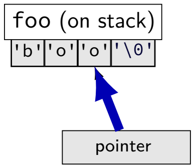
foo (on stack)

'b'	'o'	'o'	'\0'

pointer

exercise explanation

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



foo (on stack)

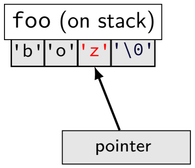
'b'	'o'	'o'	'\0'

pointer

exercise explanation

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

better style: `*pointer = 'z';`



foo (on stack)

'b'	'o'	'z'	'\0'

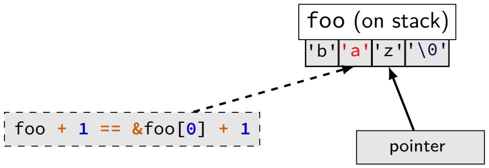
pointer

exercise explanation

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

better style: `*pointer = 'z';`

better style: `foo[1] = 'a';`



foo (on stack)

'b'	'a'	'z'	'\0'

`foo + 1 == &foo[0] + 1`

pointer

middle of blocks?

Examples of things not allowed in 1989 ANSI C:

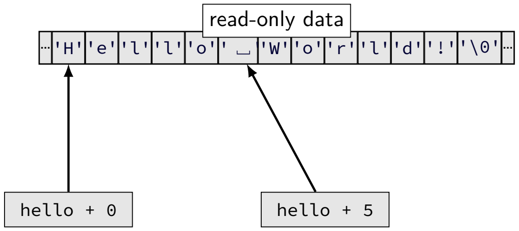
```
printf("Before calling malloc()\n");
int *pointer = malloc(sizeof(int) * 100);
```

`pointer` must be declared earlier

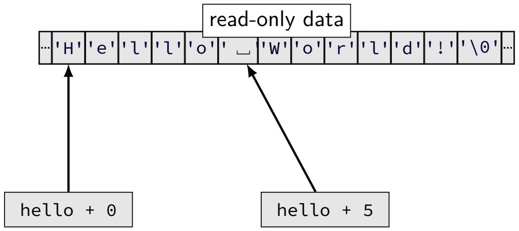
```
for (int x = 0; x < 10; ++x)
```

x must be declared earlier

pointer arithmetic



pointer arithmetic



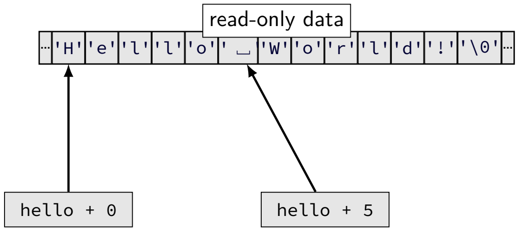
`0x4005C0`

`*(hello + 0)` is `'H'`

`0x4005C5`

`*(hello + 5)` is `'␣'`

pointer arithmetic



`0x4005C0`

`*(hello + 0)` is `'H'`

`hello[0]` is `'H'`

`0x4005C5`

`*(hello + 5)` is `'␣'`

`hello[5]` is `'␣'`