x86-64 + C

logistics notes

quizzes — hopefully you found them

- post-quiz: released Thursday night, due Sat
- pre-quiz: released Sat, due Tuesday morning

note on comments:

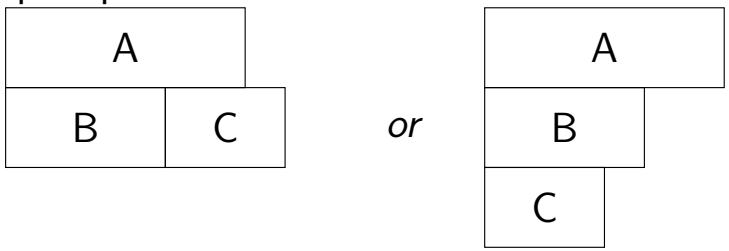
- we will eventually have TAs look at them (for answers autograded as wrong)

question order is *randomized* —

- avoid referring to answer letters in your comments

anonymous feedback (1)

quiz question: unclear:



or

agreed — multiple accepted answers

anonymous feedback (2)

also: "lecture slides did not prepare us properly to answer that question"

could not just apply formula — was intentional

- not just a case of something being sped-up/split in parallel pieces by a simple factor

intended idea: draw pictures like those shown

anonymous feedback (3)

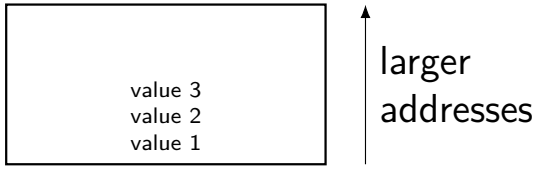
on endianness question

"do not know if ...bottom-up or top-down"

traditional memory drawing: starts at bottom

lowest address is 'first' — don't think the question was unclear

but I should've pointed out "wrong" direction



anonymous feedback (4)

why so many little assignments? "more appropriate for High School students than University students"

Sorry, but...

I don't believe most students will prep for lectures

I don't believe most students will review except before exam

on the C/C++ question

didn't cover "standards-conformant"

- yes, mistake to include in question
- question presently dropped (0 points) for this
- standard defines what C and C++ are
- compilers may accept more than standard

slide said 'C: almost a subset of C++'

- a lot of C is valid C++
- there is some C that is not valid C++

example: C that is not C++

valid C and invalid C++:

```
char *str = malloc(100);
```

valid C and valid C++:

```
char *str = (char *) malloc(100);
```

valid C and invalid C++:

```
int class = 1;
```

anonymous feedback (5)

questions on quiz not covered in lecture/reading?

not sure what you're referring to

agree emphasis on quiz 00 was a bit off

if today's pre-quiz, reading was:

Chapter 1 and
Figures 3.2, 3.3, and 3.28

non-anonymous feedback?

office hours and such

office hours are posted on the website

there is a Piazza for asking questions

Layers of Abstraction

```
x += y
```
"Higher-level" language: C

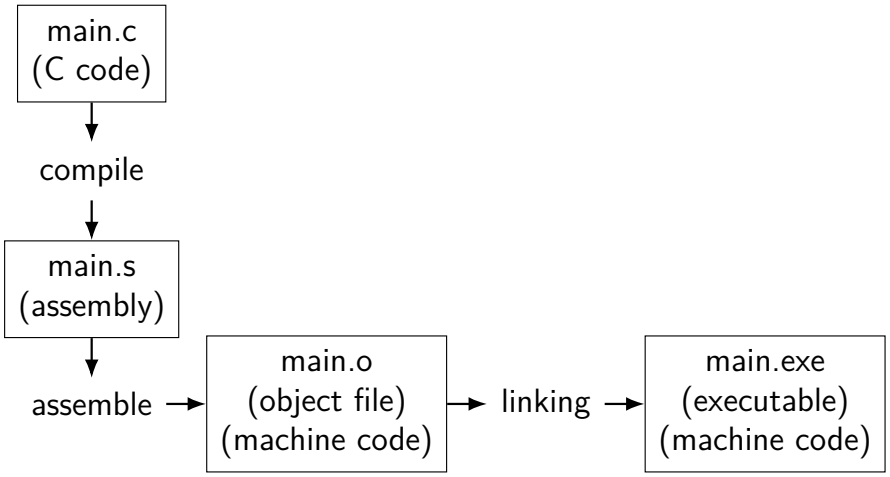
```
add %rbx, %rax
```
Assembly: X86-64

```
60 03
```
Machine code: Y86

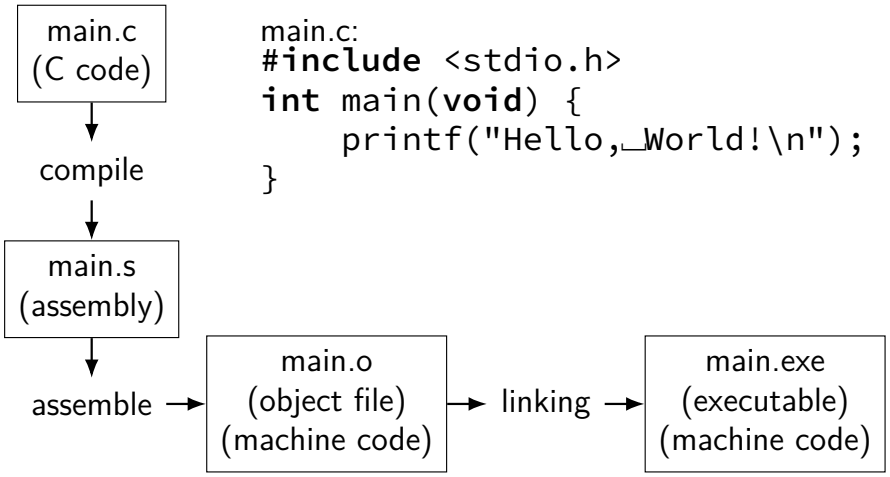
(we'll talk later)

Logic and Registers

compilation pipeline



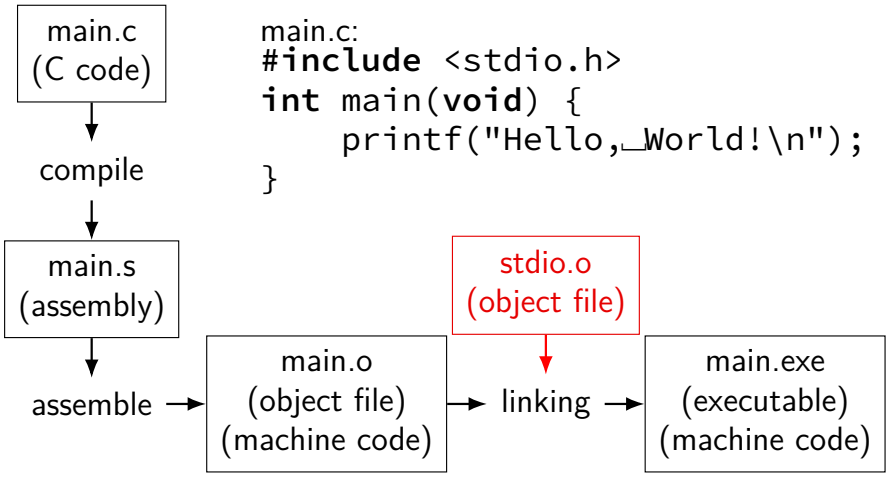
compilation pipeline



main.c:

```
#include <stdio.h>
int main(void) {
    printf("Hello, World!\n");
}
```


compilation pipeline



main.c:

```
#include <stdio.h>
int main(void) {
    printf("Hello, World!\n");
}
```


compilation commands

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?

hello.c

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


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

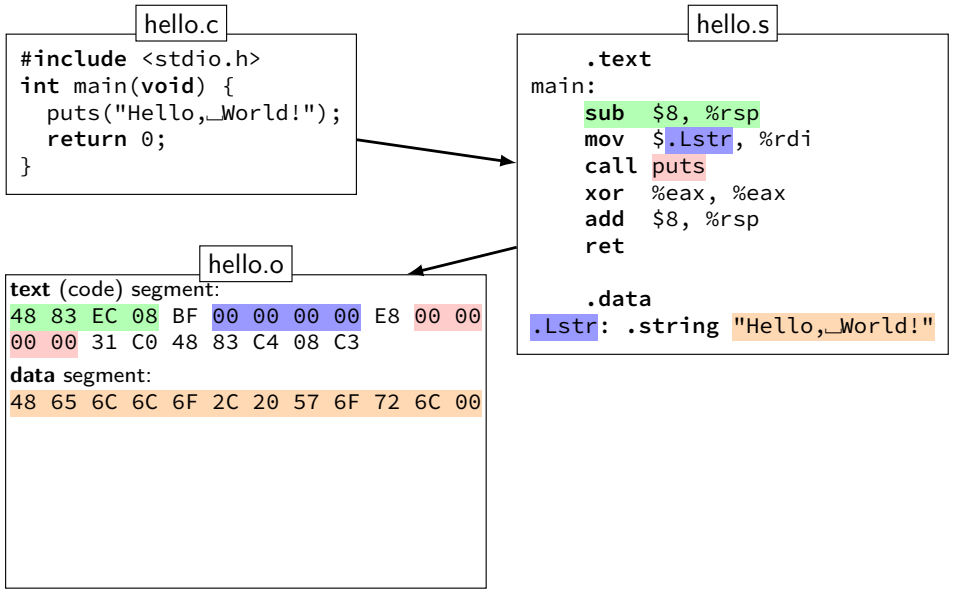
text (code) segment:

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

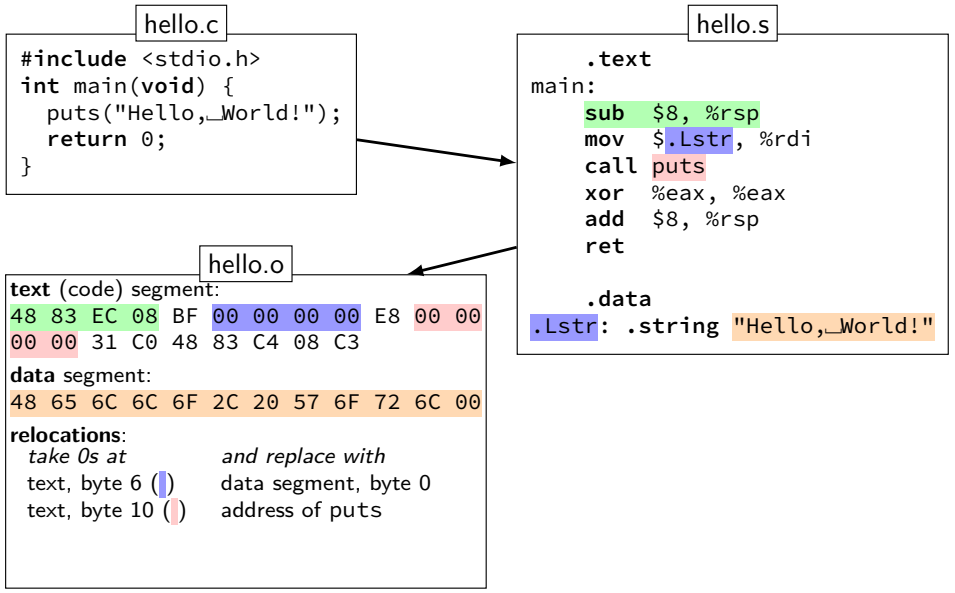
data segment:

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

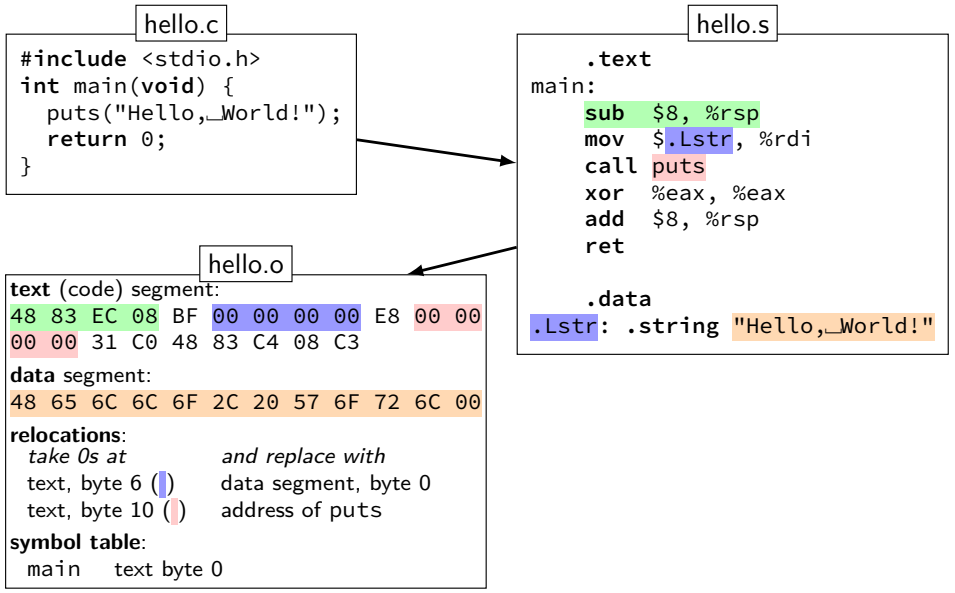
What's in those files?



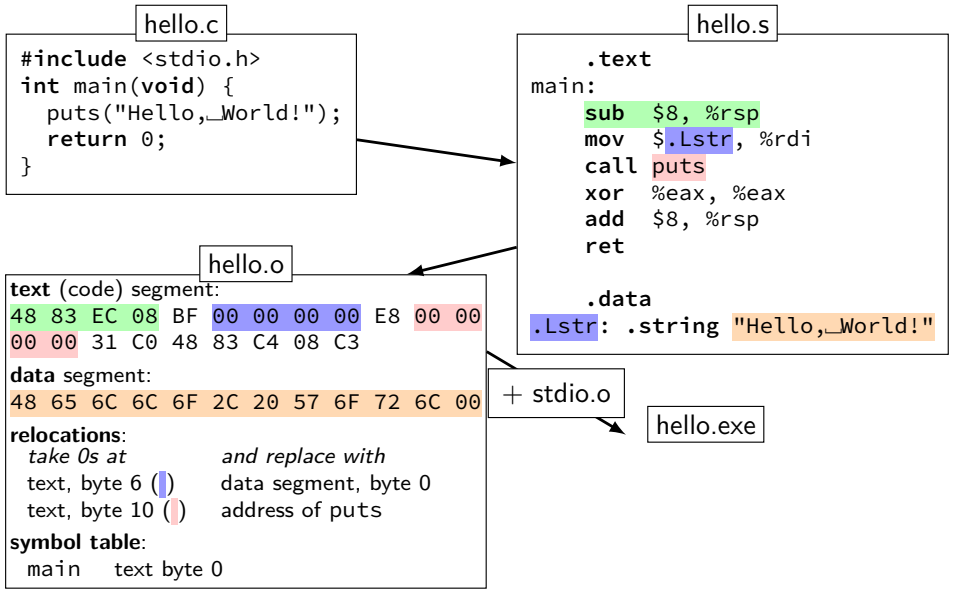
What's in those files?



What's in those files?



What's in those files?



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:

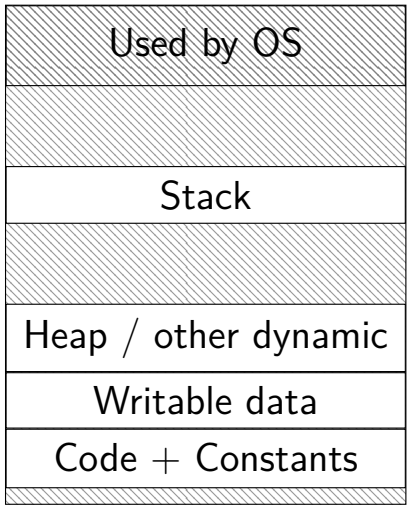
main	text byte 0

+ stdio.o

hello.exe

```
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) …
```


Program Memory (x86-64 Linux)



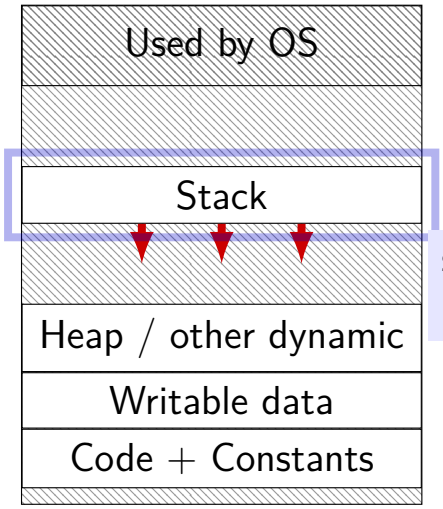
0xFFFF FFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F…

0x0000 0000 0040 0000

Program Memory (x86-64 Linux)



0xFFFF FFFF FFFF FFFF

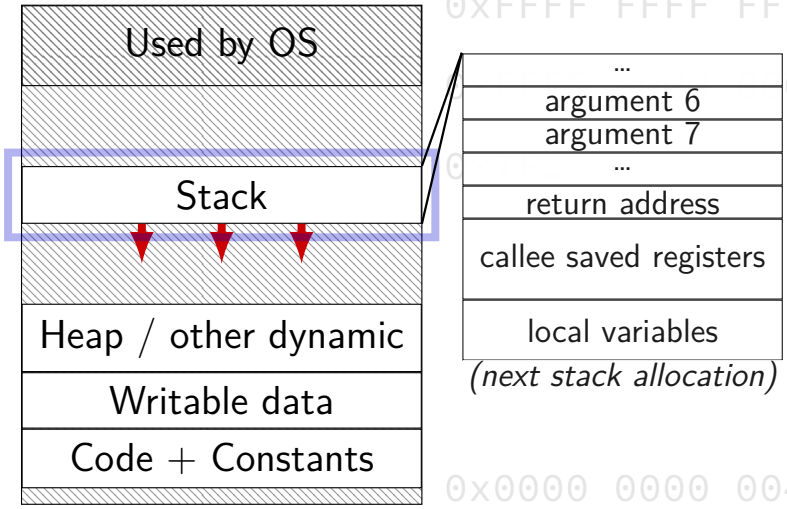
0xFFFF 8000 0000 0000

0x7F…

stack grows down
"top" has smallest address

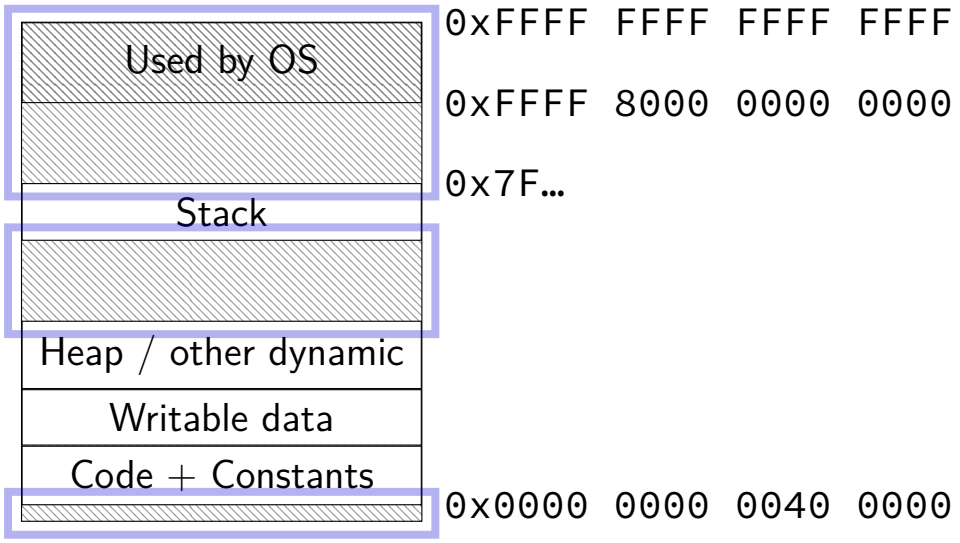
0x0000 0000 0040 0000

Program Memory (x86-64 Linux)



(next stack allocation)

Program Memory (x86-64 Linux)



x86-64 refresher

upcoming homework/lab requires reading x86-64

you can do this, but…

some surprises

AT&T versus Intel syntax (1)

AT&T syntax:
```
movq $42, (%rbx)
```

Intel syntax:
```
mov QWORD PTR [rbx], 42
```

effect (pseudo-C):
```
memory[rbx] <- 42
```


AT&T syntax example (1)

```
movq $42, (%rbx)
// memory[rbx] ← 42
```

destination last

()s represent value in memory

constants start with $

registers start with %

q indicates length (8 bytes)

l: 4; w: 2; b: 1

AT&T syntax example (1)

```
movq $42, (%rbx)
// memory[rbx] ← 42
```

destination last

( )s represent value in memory

constants start with $

registers start with %

q indicates length (8 bytes)

- l: 4; w: 2; b: 1

AT&T syntax example (1)

```
movq $42, (%rbx)
// memory[rbx] ← 42
```

destination last

( )s represent value in memory

constants start with $

registers start with %

q indicates length (8 bytes)

- l: 4; w: 2; b: 1

19

AT&T syntax example (1)

```
movq $42, (%rbx)
// memory[rbx] ← 42
```

destination last

( )s represent value in memory

constants start with $

registers start with %

q indicates length (8 bytes)

- l: 4; w: 2; b: 1

19

AT&T syntax example (1)

```
movq $42, (%rbx)
// memory[rbx] ← 42
```

destination last

( )s represent value in memory

constants start with $

registers start with %

q indicates length (8 bytes)

- l: 4; w: 2; b: 1

19

AT&T versus Intel syntax (2)

AT&T syntax:

```
movq $42, 100(%rbx,%rcx,4)
```

Intel syntax:

```
mov QWORD PTR [rbx+rcx*4+100], 42
```

effect (pseudo-C):

```
memory[rbx + rcx * 4 + 100] <- 42
```


AT&T versus Intel syntax (2)

AT&T syntax:

```
movq $42, 100(%rbx,%rcx,4)
```

Intel syntax:

```
mov QWORD PTR [rbx+rcx*4+100], 42
```

effect (pseudo-C):

```
memory[rbx + rcx * 4 + 100] <- 42
```


AT&T versus Intel syntax (2)

AT&T syntax:

```
movq $42, 100(%rbx,%rcx,4)
```

Intel syntax:

```
mov QWORD PTR [rbx+rcx*4+100], 42
```

effect (pseudo-C):

```
memory[rbx + rcx * 4 + 100] <- 42
```


AT&T versus Intel syntax (2)

AT&T syntax:

```
movq $42, 100(%rbx,%rcx,4)
```

Intel syntax:

```
mov QWORD PTR [rbx+rcx*4+100], 42
```

effect (pseudo-C):

```
memory[rbx + rcx * 4 + 100] <- 42
```

AT&T syntax: addressing

`100(%rbx)`: `memory[rbx + 100]`

`100(%rbx,8)`: `memory[rbx * 8 + 100]`

`100(%rcx,%rbx,8)`:
`memory[rcx + rbx * 8 + 100]`

AT&T versus Intel syntax (3)

`rbx ← rbx - rax`

Intel syntax:	`sub rbx, rax`
AT&T syntax:	`subq %rax, %rbx`
(or:	`sub %rax, %rbx`)

AT&T syntax: addresses

```
addq 0x1000, %rax
// rax ← rax + memory[0x1000]
addq $0x1000, %rax
// rax ← rax + 0x1000
```

no `$` — probably memory address

x86-64 calling convention

registers for first 6 arguments:

- `%rdi` (or `%edi` or `%di`, etc.), then
- `%rsi` (or `%esi` or `%si`, etc.), then
- `%rdx` (or `%edx` or `%dx`, etc.), then
- `%rcx` (or `%ecx` or `%cx`, etc.), then
- `%r8` (or `%r8d` or `%r8w`, etc.), then
- `%r9` (or `%r9d` or `%r9w`, etc.)

rest on stack

return value in `%rax`

don't memorize: Figure 3.28 in book

x86-64 calling convention example

```
// foo(1, 2, 3)
movl $1, %edi
movl $2, %esi
movl $3, %edx
call foo
// any return value in %rax
```


AT&T syntax in one slide

destination last

`()` means value in memory

`disp(base, index, scale)` same as `memory[disp + base + index * scale]`

- omit disp (defaults to `0`)
- and/or omit base (defaults to `0`)
- and/or scale (defualts to `1`)

`$` means constant

plain number/label means value in memory

question

```
pushq $0x1
pushq $0x2
addq $0x3, 8(%rsp)
popq %rax
popq %rbx
```

What is value of `%rax` and `%rbx` after this?

a. `%rax` = `0x2`, `%rbx` = `0x4`
b. `%rax` = `0x5`, `%rbx` = `0x1`
c. `%rax` = `0x2`, `%rbx` = `0x1`
d. the snippet has invalid syntax or will crash
e. more information is needed
f. something else?

more assembly later

we'll review assembly more later

including reviewing

- branching and loops
- labels
- condition codes

...since you're going to implement a CPU

C Data Types

Varies between machines(!). For this course:

type	size (bytes)
`char`	1
`short`	2
`int`	4
`long`	8

C Data Types

Varies between machines(!). For this course:

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

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`~~

Truth

~~`bool`~~

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

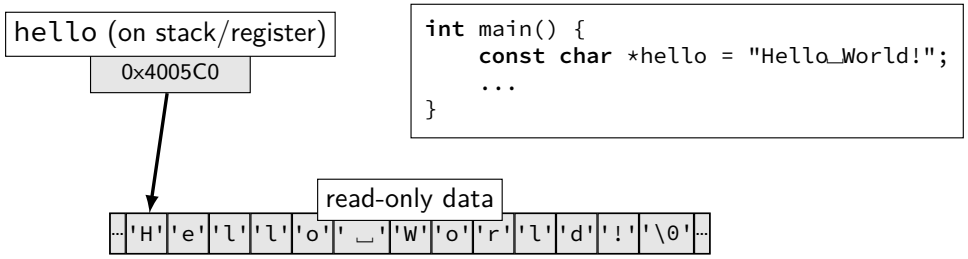
`1` if true; `0` if false

False values in C

`0`

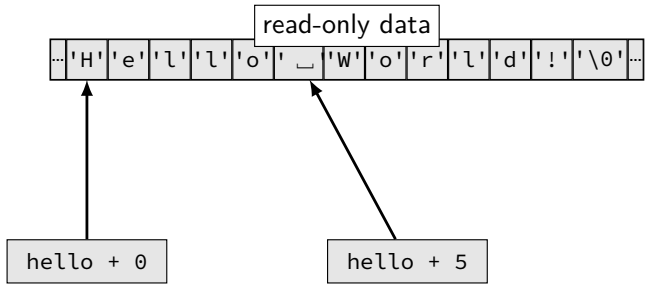
including null pointers — `0` cast to a pointer

Strings in C

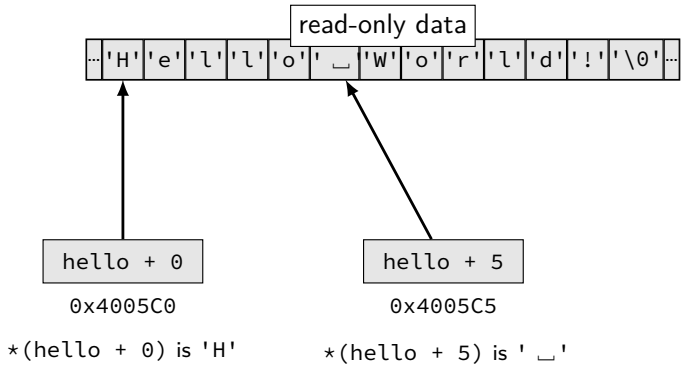



```
int main() {
    const char *hello = "Hello World!";
    ...
}
```

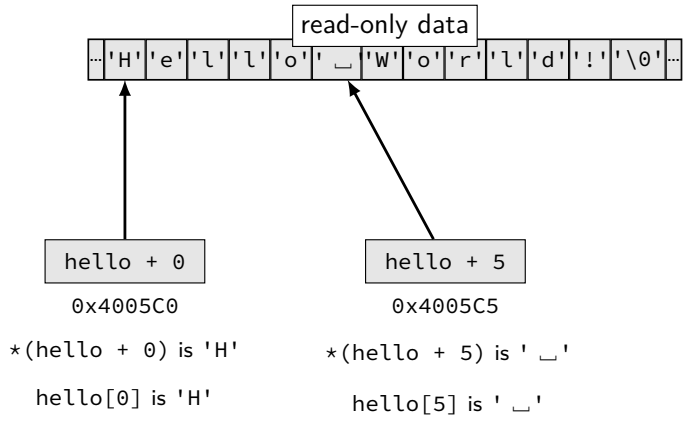

Pointer Arithmetic



Pointer Arithmetic



Pointer Arithmetic



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

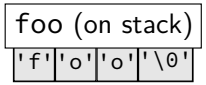
```
1 char foo[4] = "foo";
2     // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';
8 *(foo + 1) = 'a';
```

Final value of foo?

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

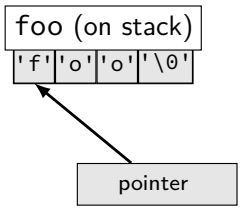
Exercise explanation

```
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';
8 *(foo + 1) = 'a';
```



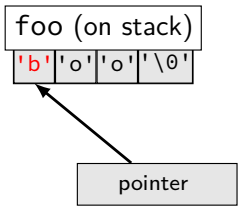
Exercise explanation

```
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';
8 *(foo + 1) = 'a';
```



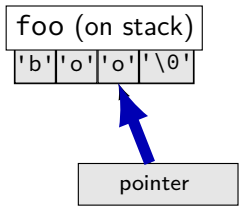
Exercise explanation

```
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';
8 *(foo + 1) = 'a';
```



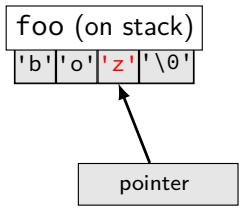
Exercise explanation

```
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';
8 *(foo + 1) = 'a';
```



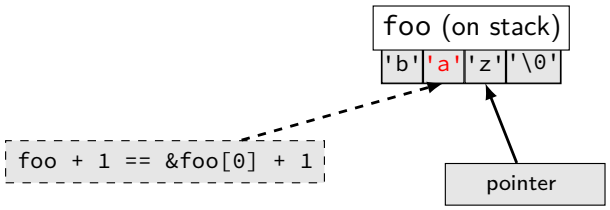
Exercise explanation

```
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';
```



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 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]);`

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

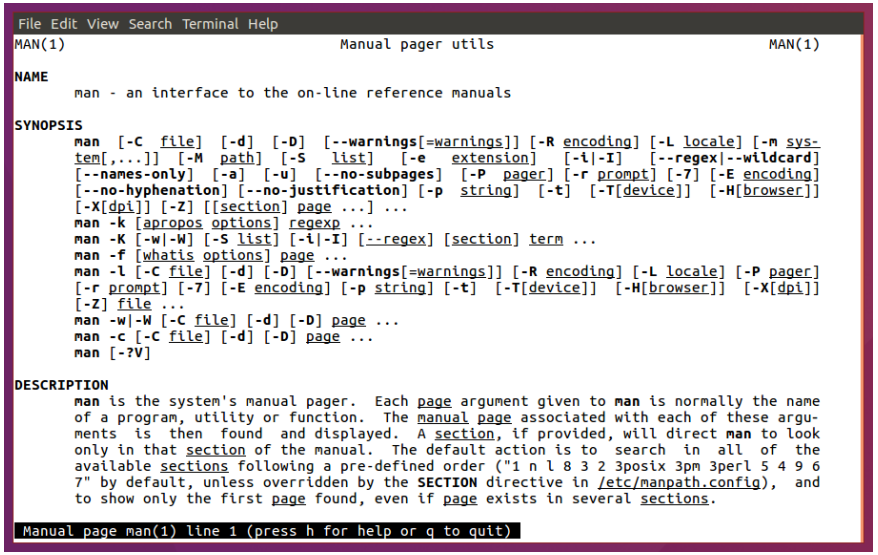
`sizeof(&array[0]) == ???`

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

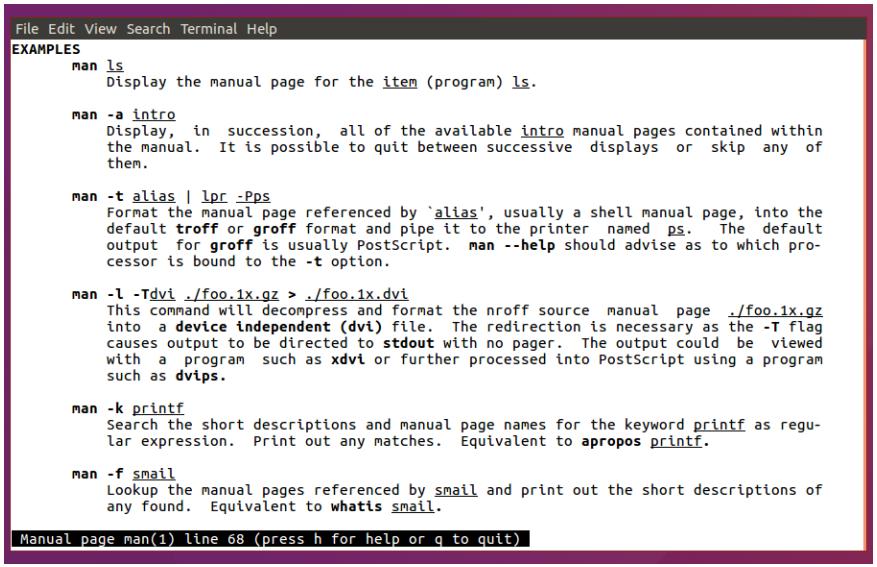
Interlude: Command Line Tips

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

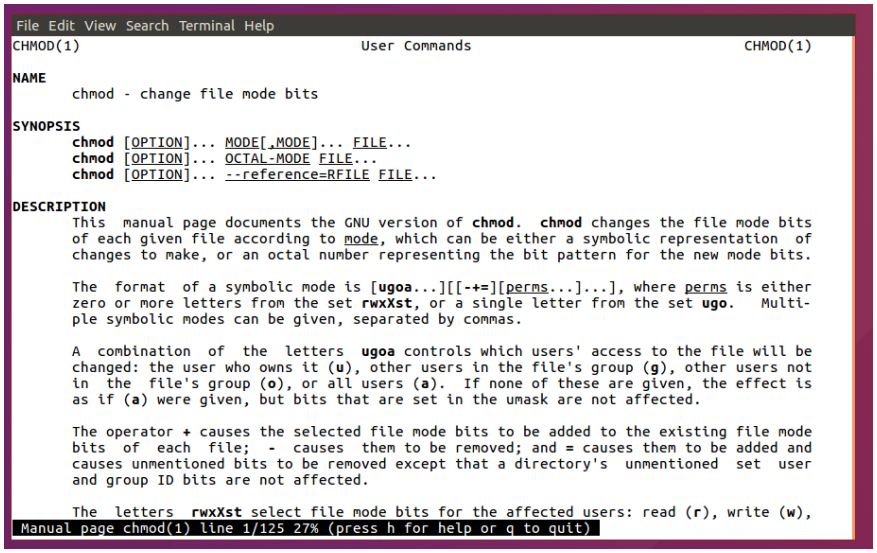

man man



man man



man chmod



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;
```


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		
%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;
```

56

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;
// ...
```

57

linked lists / dynamic allocation

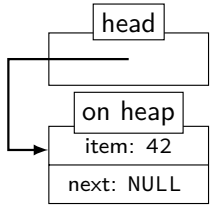
```
typedef struct list_t {
    int item;
    struct list_t *next;
} list;
// ...

  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 */
```

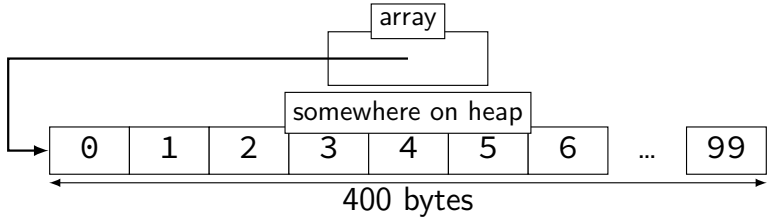


dynamic arrays

```
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
```



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	$2^{64}-1$