

Discussion 0

C, x86

09/03/24

Announcements

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
				Homework 0 Release (8/29)		
	Labor Day	Project 0 Release (9/3)		C Review Session (9/5)	Homework 0 Due Early Drop Deadline (9/6)	
	Midterm Conflict Form Due Homework 1 Release (9/9)					Group Formation Form Due

Important Policies

Contact staff through Ed.

- Use <u>cs162@eecs.berkelev.edu</u> to reach head TAs and professor if Ed is insufficient.
- Only use individual emails for private matters.

3 midterms (check website for date and time).

No conflicts allowed. Alternates are not guaranteed to happen.

Slip day policy:

- We provide 6 HW slip days and 7 Project slip days which are meant for emergencies
 - <u>Extension form</u> available for DSP and those with extenuating circumstances
 - Form is also accessible from the course website

Discussions 0 and 1 are unassigned.

- Feel free to attend different TAs' sections to find a teaching style that suits you best.
- Attendance for discussion 2 and onward are mandatory. Preference forms will be available soon.

Follow office hours policies of filling out a detailed ticket and being present in the OH room when your ticket is taken.

- All OH will be in person.
- Please check the <u>Course calendar</u> to find your OH room.
- Take advantage of empty office hours by starting assignments early.

Post your questions on Ed in the appropriate threads.

No private debugging posts allowed.

C

Resources

- C Review Session
- CS 162 Ladder
 - Overview of C and 61C topics
- CS 61C Resources Page
 - C staff notes, GDB reference card
- Python Tutor C
 - Just like CS 61A's Python Tutor, but for C

Types

C is **statically typed** (i.e. types are known at compile time).

C is **weakly typed** (i.e. can cast between any types).

Primitive types are char, short, int, long, float.

Arrays are contiguous pieces of memory of a homogenous type

- Denoted with [] (e.g. int [] for an array of integers).
- String is an array of chars with last element being null

Build compound types using structs

Also contiguous in memory

Pointers are references that hold the address of an object in memory.

- Essentially just unsigned integers.
- Prefix a pointer with * to return the value at the memory address that the pointer is holding.
- Prefix a variable with & to return the memory address of the variable.

Little Endian	+3	+2	+1	+0	
0x7FFFFFC	00	00	00	08	int a = 8
0x7FFFFFF8	FF	FF	FF	FF	int b = −1
0x7FFFFFF4	7F	FF	FF	FC	int *p = &a
0x7FFFFFF0	27	CE	00	'0'	27, CE are garbage
0x7FFFFEC	'T'	'N'	'I'	'P'	char s[] = "PINTO"
0x7FFFFE8	00	00	00	02	<pre>struct point { int x; int y;</pre>
0x7FFFFFE4	00	00	00	10	<pre>struct point pt = {16, 2}</pre>

Memory

Typical C program is divided into five segments.

- **Text** contains machine code of the compiled program.
- (Un)initialized data contains (un)initialized global/static memory.
- **Heap** contains dynamically allocated memory.
- Stack contains local variables and arguments.
- Initialized strings and global constants may be stored in read-only segments (.rodata).

Think of memory as a giant array with elements of one byte.

• Memory addresses = indices of array.

Heap memory needs to be explicitly managed by the user.

- Allocate memory using malloc, calloc, realloc which return a pointer to a chunk of memory.
- Release memory using free.

High Address

Command Line Arguments			
Stack			
\			
↑			
Неар			
Uninitialized Data			
Initialized Data			
Text			

Low Address

GNU Debugger (GDB)

Need to learn how to use it for 162 even if you skidded by 61C without it.

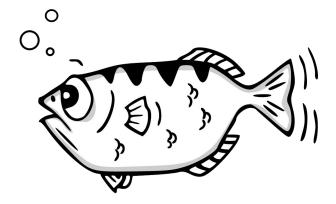
Staff will not help during OH unless you are able to use GDB.

General workflow of using GDB.

- 1. Compile program using –g flag.
- 2. Start GDB using gdb <executable name>.
- 3. Set breakpoints using break line number>. Can also break at functions using break <function name>.
- 4. Run program with run. If the program takes in arguments, pass in those after run (i.e. run arg1 arg2 ...).
- 5. Once breakpoint is hit, examine variables using print. Other commands like display, watch, and set are also useful.

Use GDB frequently to become familiar with the commands.

Check out GDB manual for more details.



Consider a valid double pointer char** dbl_char in a 32-bit system. What is returned by sizeof(*dbl_char)?
 Bonus question: Does sizeof(*dbl_char) error if dbl_char == NULL?

2. Consider strings char * a = "162 is the best" and char b[] = "162 is the best". Are a and b different?

3. Consider the following struct declaration:

```
struct point {
  int x;
  int y;
};
```

Point out a few differences between struct point p; and struct point* p; printf("%d", p.x = 1); printf("%d", p->x = 1);

1. Consider a valid double pointer char** dbl_char in a 32-bit system. What is returned by sizeof(*dbl_char)?
Dereferencing a double pointer gives a single pointer. 32-bit systems have 32-bit = 4 byte memory addresses.
Bonus question: Does sizeof(*dbl_char) error if dbl_char == NULL?

2. Consider strings char * a = "162 is the best" and char b[] = "162 is the best". Are a and b different?

 Consider the following struct declaration: struct point { int x; int y; }:

Point out a few differences between struct point p; and struct point* p; printf("%d", p.x = 1); printf("%d", p->x = 1);

1. Consider a valid double pointer char** dbl_char in a 32-bit system. What is returned by sizeof(*dbl_char)?
Dereferencing a double pointer gives a single pointer. 32-bit systems have 32-bit = 4 byte memory addresses.
Bonus question: Does sizeof(*dbl_char) error if dbl_char == NULL?
No, since type sizes are known at compile time.

2. Consider strings char* a = "162 is the best" and char b[] = "162 is the best". Are a and b different?

 Consider the following struct declaration: struct point { int x; int y;

Point out a few differences between struct point p; and struct point* p; printf("%d", p.x = 1); printf("%d", p->x = 1);

Consider a valid double pointer char** dbl_char in a 32-bit system. What is returned by sizeof(*dbl_char)?
 Dereferencing a double pointer gives a single pointer. 32-bit systems have 32-bit = 4 byte memory addresses.
 Bonus question: Does sizeof(*dbl_char) error if dbl_char == NULL?
 No, since type sizes are known at compile time.

Consider strings char* a = "162 is the best" and char b[] = "162 is the best". Are a and b different?
 Yes. a points to a string literal in the read-only segment (.rodata) while b resides on the stack.

3. Consider the following struct declaration:

Consider a valid double pointer char** dbl_char in a 32-bit system. What is returned by sizeof(*dbl_char)?
 Dereferencing a double pointer gives a single pointer. 32-bit systems have 32-bit = 4 byte memory addresses.
 Bonus question: Does sizeof(*dbl_char) error if dbl_char == NULL?
 No, since type sizes are known at compile time.

Consider strings char* a = "162 is the best" and char b[] = "162 is the best". Are a and b different?
 Yes. a points to a string literal in the read-only segment (.rodata) while b resides on the stack.

3. Consider the following struct declaration:

struct point {

```
int x;
int y;
};

Point out a few differences between
struct point p; and struct point* p;
printf("%d", p.x = 1); printf("%d", p->x = 1);
```

The first one allocates struct point that is the size of two ints. On the other hand, the second one puts a single pointer to a struct point on the stack, which is the size of a single int. Also, the second one will likely segfault, since the pointer is uninitialized.

```
#include <stdio.h>
                                                 typedef struct helper_args {
                                                                                                   #include "lib.h"
#include "lib.h"
                                                 #ifdef ABC
                                                                                                   char* helper_func(helper_args_t* args) {
                                                  char* aux:
int main(int argc, char** argv) {
                                                 #endif
                                                                                                     int i:
  helper_args_t helper_args;
                                                  char* string;
                                                                                                     for (i = 0; args->string[i] != '\0'; i++)
                                                                                                       if (args->string[i] == args->target)
  helper_args.string = argv[0];
                                                  char target;
  helper_args.target = '/';
                                                 } helper_args_t;
                                                                                                         return &args->string[i + 1];
  char* result = helper_func(&helper_args);
                                                 char* helper_func(helper_args_t* args);
                                                                                                     return args->string;
  printf("%s\n", result);
  return 0:
                      App.c
                                                                      lib.h
                                                                                                                       lib.c
```

You build the program on a 64-bit machine as follows.

```
gcc -c app.c -o app.ogcc -c lib.c -o lib.ogcc app.o lib.o -o app
```

What is the size of a helper_args_t struct?

Suppose you add a #define ABC at the top of lib.h. What is the size of a helper_args_t struct?

```
#include <stdio.h>
#include "lib.h"

int main(int argc, char** argv) {
  helper_args_t helper_args;
  helper_args.string = argv[0];
  helper_args.target = '/';
  char* result = helper_func(&helper_args);
  printf("%s\n", result);
  return 0;
}
```

App.c

What is the size of a helper_args_t struct?

16 bytes. Only char* string and char target meaning 9 bytes but GCC pads structs.

```
typedef struct helper_args {
#ifdef ABC
  char* aux;
#endif
  char* string;
  char target;
} helper_args_t;
char* helper_func(helper_args_t* args);
```

lib.h

2. Suppose you add a #define ABC at the top of lib.h. What is the size of a helper_args_t struct?

```
#include "lib.h"

char* helper_func(helper_args_t* args) {
   int i;
   for (i = 0; args->string[i] != '\0'; i++)
      if (args->string[i] == args->target)
        return &args->string[i + 1];
   return args->string;
}
```

```
#include <stdio.h>
#include "lib.h"

int main(int argc, char** argv) {
  helper_args_t helper_args;
  helper_args.string = argv[0];
  helper_args.target = '/';
  char* result = helper_func(&helper_args);
  printf("%s\n", result);
  return 0;
}
App.c
```

1. What is the size of a helper_args_t struct?

16 bytes. Only char* string and char target meaning 9 bytes but GCC pads structs.

```
typedef struct helper_args {
#ifdef ABC
  char* aux;
#endif
  char* string;
  char target;
} helper_args_t;
char* helper_func(helper_args_t* args);
```

lib.h

Suppose you add a #define ABC at the top of lib.h. What is the size of a helper_args_t struct?
 24 bytes. Additional 8 bytes from char* aux

24 bytes. Additional 8 bytes from char* aux since ABC is defined.

```
#include "lib.h"

char* helper_func(helper_args_t* args) {
  int i;
  for (i = 0; args->string[i] != '\0'; i++)
    if (args->string[i] == args->target)
      return &args->string[i + 1];
  return args->string;
}
```

```
#include <stdio.h>
#include "lib.h"

int main(int argc, char** argv) {
  helper_args_t helper_args;
  helper_args.string = argv[0];
  helper_args.target = '/';
  char* result = helper_func(&helper_args);
  printf("%s\n", result);
  return 0;
}
```

App.c

Suppose you build the program in a different way with the original files (i.e. none of the changes from previous

> gcc -DABC -c app.c -o app.o
> gcc -c lib.c -o lib.o
> gcc app.o lib.o -o app

questions apply).

The program will now exhibit undefined behavior. Why?

```
typedef struct helper_args {
#ifdef ABC
  char* aux;
#endif
  char* string;
  char target;
} helper_args_t;
char* helper_func(helper_args_t* args);
```

lib.h

```
#include "lib.h"

char* helper_func(helper_args_t* args) {
  int i;
  for (i = 0; args->string[i] != '\0'; i++)
    if (args->string[i] == args->target)
      return &args->string[i + 1];
  return args->string;
}
```

```
#include <stdio.h>
#include "lib.h"

int main(int argc, char** argv) {
  helper_args_t helper_args;
  helper_args.string = argv[0];
  helper_args.target = '/';
  char* result = helper_func(&helper_args);
  printf("%s\n", result);
  return 0;
}
```

App.c

 Suppose you build the program in a different way with the original files (i.e. none of the changes from previous questions apply).

```
> gcc -DABC -c app.c -o app.o
> gcc -c lib.c -o lib.o
> gcc app.o lib.o -o app
The grace and will approach in the second of the sec
```

The program will now exhibit undefined behavior. Why?

```
typedef struct helper_args {
#ifdef ABC
  char* aux;
#endif
  char* string;
  char target;
} helper_args_t;
char* helper_func(helper_args_t* args);
```

lib.h

app.cis compiled with ABC defined but lib.cis not

- main stored argv[0] at address of helper_args +8
- helper_func access address of args when accessing args->string
- First 8 bytes of args which helper_func is accessing is garbage

```
#include "lib.h"

char* helper_func(helper_args_t* args) {
  int i;
  for (i = 0; args->string[i] != '\0'; i++)
    if (args->string[i] == args->target)
      return &args->string[i + 1];
  return args->string;
}
```

This is a GDB exercise. Here's the Repl if you want to try it out on your own machine:



```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++j:
    while (a[i] <= pivot && i <= r);
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
   sort(a, l, j-1);
   sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL;
 if (argc > 1)
   a = argv[1];
  else
   a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeyeon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted: "!Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

.. We want to debug the program using GDB. How should we compile the program?

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
     ++j:
    while (a[i] <= pivot && i <= r);
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
   sort(a, l, j-1);
   sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
 if (argc > 1)
   a = argv[1];
  else
   a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeyeon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted: "!Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

We want to debug the program using GDB. How should we compile the program?

gcc -g singer.c -o singer Needa-gflag for debugging.

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++j:
    while (a[i] <= pivot && i <= r);
    do
      --i;
    while (a[i] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
    sort(a, l, j-1);
   sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
 if (argc > 1)
   a = argv[1];
  else
   a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeyeon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted: "!Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

2. When running the program without any arguments, what line does the segfault happen? Describe the memory operations happening in that line.

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++i:
    while (a[i] <= pivot && i <= r);
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break:
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
    sort(a, l, j-1);
    sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
 if (argc > 1)
   a = argv[1];
  else
   a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeveon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted : " !Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

When running the program without any arguments, what line does the segfault happen? Describe the memory operations happening in that line.

Find segfaulting line by letting the program run until it encounters the fault.

```
> gcc -g singer.c -o singer
> gdb singer
(gdb) run
Starting program: /home/runner/intro/singer
Unsorted: "IU is the best char!"

Program received signal SIGSEGV, Segmentation
fault.
0x00005646308006c8 in swap (a=0x564630800904
"IU is the best singer!", i=1, j=21)
at singer.c:6
```

Ignore "warning: Error disabling address space randomization: Operation not permitted" if using Replit.

a[i] = a[j];

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l]:
  int i = l, j = r+1;
  while (1) {
    do
      ++i:
    while (a[i] <= pivot && i <= r);</pre>
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
  if (l < r){
   int j = partition(a, l, r);
    sort(a, l, j-1);
    sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
  if (argc > 1)
   a = argv[1];
  else
    a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeveon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted : " !Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

When running the program without any arguments, what line does the segfault happen? Describe the memory operations happening in that line.

Use backtrace for more comprehensive breakdown.

```
(gdb) backtrace
#0 0x00005646308006c8 in swap
(a=0x564630800904 "IU is the best singer!",
i=1, j=21)
    at singer.c:6
#1 0x0000564630800773 in partition
(a=0x564630800904 "IU is the best singer!",
l=0, r=21)
    at singer.c:26
#2 0x00005646308007bd in sort
(a=0x564630800904 "IU is the best singer!",
l=0, r=21)
    at singer.c:36
#3 0x0000564630800861 in main (argc=1,
argv=0x7ffd04ac7098) at singer.c:51
```

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++j:
    while (a[i] <= pivot && i <= r);
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
  if (l < r){
   int j = partition(a, l, r);
    sort(a, l, j-1);
    sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
  if (argc > 1)
   a = argv[1];
  else
    a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeveon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted: "!Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

 When running the program without any arguments, what line does the segfault happen? Describe the memory operations happening in that line.

Two memory operations

- Read from a [j].
- 2. Write to a [i].

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++j:
    while (a[i] <= pivot && i <= r);
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
    sort(a, l, j-1);
   sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
 if (argc > 1)
   a = argv[1];
  else
   a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeyeon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted : " !Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

3. Run the program with and without an argument and observe the memory addresses of a in the segfaulting line. Why are the memory addresses so different?

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++j:
    while (a[i] <= pivot && i <= r);
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
    sort(a, l, j-1);
    sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
 if (argc > 1)
   a = argv[1];
  else
   a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeveon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted : " !Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

Run the program with and without an argument and observe the memory addresses of a in the segfaulting line. Why are the memory addresses so different?

Break at line 6 using GDB.

```
> gdb singer
(gdb) break 6
Breakpoint 1 at 0x6ab: file singer.c, line 6.
(gdb) run
Starting program: /home/runner/intro/singer
Unsorted: "IU is the best singer!"

Breakpoint 1, swap (
    a=0x5624e4600904 "IU is the best singer!",
i=1, j=21)
    at singer.c:6
6    a[i] = a[j];
(gdb) print a
$1 = 0x5624e4600904 "IU is the best singer!"
```

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++i:
    while (a[i] <= pivot && i <= r);</pre>
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break:
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
    sort(a, l, j-1);
    sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
 if (argc > 1)
   a = argv[1];
  else
   a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeveon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted : " !Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

3. Run the program with and without an argument and observe the memory addresses of a in the segfaulting line. Why are the memory addresses so different?

Break at line 6 using GDB.

```
(gdb) run "Taeyeon is the best singer!"
The program being debugged has been started
already.
Start it from the beginning? (y or n) y
Starting program: /home/runner/intro/asuna
"Taeyeon is the best singer!"
Unsorted: "Taeyeon is the best singer!"

Breakpoint 1, swap (
    a=0x7ffcce01bfe5 "Taeyeon is the best
singer!", i=1, j=26)
    at asuna.c:6
(gdb) print a
$2 = 0x7ffcce01bfe5 "Taeyeon is the best
singer!"
```

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++i:
    while (a[i] <= pivot && i <= r);</pre>
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
    sort(a, l, j-1);
    sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
 if (argc > 1)
   a = argv[1];
  else
   a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeveon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted: "!Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

Run the program with and without an argument and observe the memory addresses of a in the segfaulting line. Why are the memory addresses so different?

No argument: 0x5624e4600904

 Statically defined strings stored in read-only segment (.rodata).

With argument: 0x7ffcce01bfe5

Arguments are passed in through the stack.

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++j:
    while (a[i] <= pivot && i <= r);
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
   sort(a, l, j-1);
   sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL;
 if (argc > 1)
   a = argv[1];
  else
   a = "IU is the best singer!";
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeyeon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted: "!Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

How should the code be changed to fix the segfault?

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++j:
    while (a[i] <= pivot && i <= r);</pre>
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
    sort(a, l, j-1);
    sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
 if (argc > 1)
   a = argv[1];
  else
   a = "IU is the best singer!":
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeyeon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted: "!Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

4. How should the code be changed to fix the segfault?

Write to a[i] is the problem since it's in read-only data \rightarrow need to put a in writable memory.

Allocate memory on the heap and put the default string on there.

- Accomplish with malloc followed by strcpy.
- Equivalently, use <u>strdup</u>.

Replace line 48.

```
void swap(char* a, int i, int j) {
  char t = a[i];
  a[i] = a[i];
  a[i] = t;
int partition(char* a, int l, int r){
  int pivot = a[l];
  int i = l, j = r+1;
  while (1) {
    do
      ++j:
    while (a[i] <= pivot && i <= r);
    do
      --i;
    while (a[j] > pivot);
    if (i >= j)
     break;
    swap(a, i, j);
  swap(a, l, j);
  return i:
```

```
void sort(char* a, int l, int r){
 if (l < r){
   int j = partition(a, l, r);
    sort(a, l, j-1);
    sort(a, j+1, r);
int main(int argc, char** argv){
  char* a = NULL:
 if (argc > 1)
   a = argv[1];
  else
    a = strdup("IU is the best singer!");
  printf("Unsorted: \"%s\"\n", a);
  sort(a, 0, strlen(a) - 1);
  printf("Sorted : \"%s\"\n", a);
> ./singer "Taeyeon is the best singer!"
Unsorted: "Taeyeon is the best singer!"
Sorted: "!Tabeeeeeghiinnorssstty"
> ./singer
Unsorted: "IU is the best singer!"
Segmentation fault (core dumped)
```

4. How should the code be changed to fix the segfault?

Write to a[i] is the problem since it's in read-only data \rightarrow need to put a in writable memory.

Allocate memory on the heap and put the default string on there.

- Accomplish with malloc followed by strcpy.
- Equivalently, use <u>strdup</u>.

Replace line 48.

x86

Registers

Registers are small storage spaces directly on the processor.

• Allows for fast memory access.

General purpose registers (GPR) store both data and addresses.

- x86 has 8, RISC-V has 32 (x0-x31).
- Started as 16-bits, extend to 32-bit using e prefix (e.g. EAX for AX).
- Access 8-bit LSB by replacing last letter with I (e.g. AL for AX).
- Access 8-bit MSB by replacing last letter with h (e.g. AH for AX) only for AX, BX, CX, DX.

Instruction pointer register holds address of next instruction to execute.

- Called ip which can be extended with e prefix like a GPR.
- Can't be read/modified like a GPR using regular memory instructions.

	Name	Purpose
AX	Accumulator	I/O port access, arithmetic, interrupt calls
ВХ	Base	Base pointer for memory access
CX	Counter	Loop counting, bit shifts
DX	Data	I/O port access, arithmetic, interrupt calls
SP	Stack Pointer	Top address of stack
BP	Base Pointer	Base address of stack
SI	Source Index	Source for stream operations (e.g. string)
DI	Destination Index	Destination for stream operations (e.g. string)

Syntax

Use AT&T Syntax not Intel which is used by GCC.

Prefix registers with % (e.g. %eax), constants with \$ (e.g. \$4).

General structure is **inst src, dest**.

Address memory with offset(base, index, scale).

- base, index = registers, offset = any integer, scale = 1, 2, 4, or 8.
- Accesses data at address base + index * scale + offset.
- All parameters optional but will see base and offset usually.
- Using lea instruction will operate on the address itself not contents.

Suffix instructions to signify operand size.

• Not always necessary but should use them regardless.

mov 8(%ebx), %eax	Move contents from address EBX + 8 into EAX
mov %ecx, -4(%esi, %ebx, 8)	Move contents in ECX into address ESI + 8 * EBX - 4
lea 8(%ebx), %eax	Puts EBX + 8 into EAX

movb \$0, (%esp)	Zero out a single byte from ESP
movw \$0, (%esp)	Zero out two bytes from ESP
movl \$0, (%esp)	Zero out four bytes from ESP

Calling Convention

Calling convention is a procedure for how to call and return from functions.

- Specifies stack management, argument passing, register saving, etc.
- One set of rule each for the caller and callee

Many different calling conventions, will use the one defined in i386 System V ABI in this class.

EBP Existing Caller Stack Frame ESP

Caller

Before calling the function (i.e. prologue),

Save caller-saved GPRs (EAX, ECX, EDX) onto the stack if needed after the function call.

EBP

Existing Caller Stack Frame
Caller-Saved GPRs

Caller

Before calling the function (i.e. prologue),

- Save caller-saved GPRs (EAX, ECX, EDX) onto the stack if needed after the function call.
- 2. Push parameters onto the stack in reverse order. Add necessary padding before the parameters to ensure a 16-byte alignment.

EBP Existing Caller Stack Frame Caller-Saved GPRs **Padding Function Parameters** ESP (16 Byte aligned)

Caller

Before calling the function (i.e. prologue),

- 1. Save caller-saved GPRs (EAX, ECX, EDX) onto the stack if needed after the function call.
- 2. Push parameters onto the stack in reverse order. Add necessary padding *before the parameters* to ensure a 16-byte alignment.

Call function by pushing the return address onto the stack and jumping to the function.

EBP Existing Caller Stack Frame Caller-Saved GPRs **Padding Function Parameters** Return Address **ESP**

Callee

Before executing any function logic (i.e. prologue),

1. Push EBP onto the stack and set EBP to be the new ESP.

Existing Caller Stack Frame Caller-Saved GPRs **Padding Function Parameters** Return Address **EBP of Caller Stack Frame**

EBP, ESP

Callee

Before executing any function logic (i.e. prologue),

- 1. Push EBP onto the stack and set EBP to be the new ESP.
- 2. Allocate stack space for local variables.

Existing Caller Stack Frame
Caller-Saved GPRs
Padding
Function Parameters
Return Address
EBP of Caller Stack Frame
Local Variables

EBP

Callee

Before executing any function logic (i.e. prologue),

- 1. Push EBP onto the stack and set EBP to be the new ESP.
- 2. Allocate stack space for local variables.
- 3. Save callee-saved GPRs (EBX, EDI, ESI) onto the stack if used during the function logic.

Existing Caller Stack Frame
Caller-Saved GPRs
Padding
Function Parameters
Return Address
EBP of Caller Stack Frame
Local Variables
Callee-Saved GPRs

EBP

Callee

Before executing any function logic (i.e. prologue),

- 1. Push EBP onto the stack and set EBP to be the new ESP.
- 2. Allocate stack space for local variables.
- 3. Save callee-saved GPRs (EBX, EDI, ESI) onto the stack if used during the function logic.

Perform function logic.

Existing Caller Stack Frame
Caller-Saved GPRs
Padding
Function Parameters
Return Address
EBP of Caller Stack Frame
Local Variables
Callee-Saved GPRs

EBP

Callee

Before executing any function logic (i.e. prologue),

- 1. Push EBP onto the stack and set EBP to be the new ESP.
- 2. Allocate stack space for local variables.
- 3. Save callee-saved GPRs (EBX, EDI, ESI) onto the stack if used during the function logic.

Perform function logic.

Before returning (i.e. epilogue),

1. Store return value in EAX.

Existing Caller Stack Frame
Caller-Saved GPRs
Padding
Function Parameters
Return Address
EBP of Caller Stack Frame
Local Variables
Callee-Saved GPRs

EBP

Callee

Before executing any function logic (i.e. prologue),

- 1. Push EBP onto the stack and set EBP to be the new ESP.
- 2. Allocate stack space for local variables.
- 3. Save callee-saved GPRs (EBX, EDI, ESI) onto the stack if used during the function logic.

Perform function logic.

Before returning (i.e. epilogue),

- 1. Store return value in EAX.
- 2. Restore callee-saved GPRs if any from the prologue.

Existing Caller Stack Frame
Caller-Saved GPRs
Padding
Function Parameters
Return Address
EBP of Caller Stack Frame
Local Variables
Callee Saved GPRs

EBP

Callee

Before executing any function logic (i.e. prologue),

- 1. Push EBP onto the stack and set EBP to be the new ESP.
- 2. Allocate stack space for local variables.
- 3. Save callee-saved GPRs (EBX, EDI, ESI) onto the stack if used during the function logic.

Perform function logic.

Before returning (i.e. epilogue),

- 1. Store return value in FAX.
- 2. Restore callee-saved GPRs if any from the prologue.
- 3. Deallocate local variables.

Existing Caller Stack Frame Caller-Saved GPRs **Padding Function Parameters** Return Address FBP of Caller Stack Frame

EBP, ESP

Callee

Before executing any function logic (i.e. prologue),

- 1. Push EBP onto the stack and set EBP to be the new ESP.
- 2. Allocate stack space for local variables.
- 3. Save callee-saved GPRs (EBX, EDI, ESI) onto the stack if used during the function logic.

Perform function logic.

Before returning (i.e. epilogue),

- 1. Store return value in EAX.
- 2. Restore callee-saved GPRs if any from the prologue.
- Deallocate local variables.
- 4. Restore caller's EBP from stack.

EBP

Existing Caller Stack Frame Caller-Saved GPRs **Padding Function Parameters** Return Address FBP of Caller Stack Frame

Callee

Before executing any function logic (i.e. prologue),

- 1. Push EBP onto the stack and set EBP to be the new ESP.
- 2. Allocate stack space for local variables.
- 3. Save callee-saved GPRs (EBX, EDI, ESI) onto the stack if used during the function logic.

Perform function logic.

Before returning (i.e. epilogue),

- 1. Store return value in EAX.
- 2. Restore callee-saved GPRs if any from the prologue.
- Deallocate local variables.
- Restore caller's FBP from stack.
- Return from function call by popping the return address pushed by the caller in its prologue and jumping to it.

EBP

FSP

Existing Caller Stack Frame Caller-Saved GPRs **Padding Function Parameters** Return Address

Caller

Before calling the function (i.e. prologue),

- 1. Save caller-saved GPRs (EAX, ECX, EDX) onto the stack if needed after the function call.
- 2. Push parameters onto the stack in reverse order. Add necessary padding *before the parameters* to ensure a 16-byte alignment.

Call function by pushing the return address onto the stack and jumping to the function.

Once function call returns (i.e. epilogue),

1. Remove parameters from the stack.

EBP

ESP

Existing Caller Stack Frame Caller-Saved GPRs

Caller

Before calling the function (i.e. prologue),

- 1. Save caller-saved GPRs (EAX, ECX, EDX) onto the stack if needed after the function call.
- 2. Push parameters onto the stack in reverse order. Add necessary padding *before the parameters* to ensure a 16-byte alignment.

Call function by pushing the return address onto the stack and jumping to the function.

Once function call returns (i.e. epilogue),

- 1. Remove parameters from the stack.
- 2. Restore caller-saved GPRs if any from the prologue.

EBP

Existing Caller Stack Frame
Caller Saved GPRs
Padding
Function Parameters
Return Address
EBP of Caller Stack Frame
Local Variables
Callee Saved GPRs

Caller

Before calling the function (i.e. **prologue**),

- 1. Save caller-saved GPRs (EAX, ECX, EDX) onto the stack if needed after the function call.
- 2. Push parameters onto the stack in reverse order. Add necessary padding before the parameters to ensure a 16-byte alignment.

Call function by pushing the return address onto the stack and jumping to the function.

Once function call returns (i.e. epilogue),

- 1. Remove parameters from the stack.
- 2. Restore caller-saved GPRs if any from the prologue.

Callee

Before executing any function logic (i.e. prologue),

- Push EBP onto the stack and set EBP to be the new ESP.
- 2. Allocate stack space for local variables.
- 3. Save callee-saved GPRs (EBX, EDI, ESI) onto the stack if used during the function logic.

Perform function logic.

Before returning (i.e. epilogue),

- Store return value in EAX.
- 2. Restore callee-saved GPRs if any from the prologue.
- 3. Deallocate local variables.
- 4. Restore caller's EBP from stack.
- Return from function call by popping the return address pushed by the caller in its prologue and jumping to it.

Instruction	Purpose	Effective
pushl src	Push src onto stack	subl \$4, %esp movl src, (%esp)
popl dest	Pop from stack into dest	movl (%esp), dest addl \$4, %esp
call addr	Push return address onto stack and jump to addr	pushl %eip jump addr
leave	Restore EBP and ESP to previous stack frame	movl %ebp, %esp popl %ebp
ret	Pop return address from stack and jump to it	popl %eip

 Between SP and BP, which has a higher memory addr 	ess?
---	------

2. Write three different ways to clear the EAX register (i.e. store a 0).

3. True or False: Right before the caller jumps to the desired function, the stack must be 16-byte aligned.

Between SP and BP, which has a higher memory address?
 BP. Stack grows downwards → top of the stack (SP) moves towards lower addresses

2. Write three different ways to clear the EAX register (i.e. store a 0).

3. True or False: Right before the caller jumps to the desired function, the stack must be 16-byte aligned.

Between SP and BP, which has a higher memory address?
 BP. Stack grows downwards → top of the stack (SP) moves towards lower addresses

2. Write three different ways to clear the EAX register (i.e. store a 0).

```
movl $0, %eax subl %eax, %eax xorl %eax, %eax
```

3. True or False: Right before the caller jumps to the desired function, the stack must be 16-byte aligned.

Between SP and BP, which has a higher memory address?

BP. Stack grows downwards → top of the stack (SP) moves towards lower addresses

2. Write three different ways to clear the EAX register (i.e. store a 0).

movl \$0, %eax

subl %eax, %eax

xorl %eax, %eax

3. True or False: Right before the caller jumps to the desired function, the stack must be 16-byte aligned.

False. Stack needs to be 16-byte aligned after parameters have been pushed onto the stack. Return address is pushed right before jumping.

```
call.s
                                         27
                                                    pushl
                                                             $3
                                         28
                                                    call
                                                             bar
1 p:
                                                    addl
                                                             $16, %esp
2
          .zero
                   4
                                         29
                                                    addl
3 bar:
                                         30
                                                             %ebx, %eax
          pushl
                  %ebp
                                         31
                                                    movl
                                                             %eax, p
4
          movl
                  %esp, %ebp
5
                                         32
                                                    nop
6
          subl
                  $16, %esp
                                         33
                                                    movl
                                                             -4(%ebp), %ebx
                  8(%ebp), %edx
          movl
                                         34
                                                    leave
                  12(%ebp), %eax
          movl
                                         35
                                                    ret
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
                                                   Which lines of code correspond to a
10
          movl
                  %eax, -4(%ebp)
11
                                                   caller/callee prologue?
                  -4(%ebp), %eax
          movl
12
          addl
13
                  $1, %eax
          leave
14
15
          ret
16 foo:
          pushl
                  %ebp
17
          movl
                  %esp, %ebp
18
          pushl
                  %ebx
19
          subl
20
                  $4, %esp
21
          movl
                  8(%ebp), %edx
                  12(%ebp), %eax
22
          movl
                  (%edx,%eax), %ebx
23
          leal
          subl
                  $4, %esp
24
25
          pushl
                  $5
          pushl
                   $4
26
```

```
call.s
                                                        pushl
                                            27
                                                                 $3
                                            28
                                                        call
                                                                 bar
1 p:
                                                                 $16, %esp
                                                        addl
2
           .zero
                    4
                                            29
                                                        addl
3 bar:
                                                                 %ebx, %eax
                                            30
                    %ebp
           pushl
                                            31
                                                        movl
                                                                 %eax, p
4
                   %esp, %ebp
5
          movl
                                            32
                                                        nop
           subl
                    $16, %esp
                                            33
                                                        movl
                                                                 -4(%ebp), %ebx
6
                    8(%ebp), %edx
          movl
                                            34
                                                        leave
                   12(%ebp), %eax
                                                        ret
          movl
                                            35
8
           addl
                   %edx, %eax
9
           subl
                   16(%ebp), %eax
                                                      Which lines of code correspond to a
10
          movl
                   %eax, -4(%ebp)
11
                                                       caller/callee prologue?
                   -4(%ebp), %eax
          movl
12
                                                       foo/bar as caller/callee
           addl
                    $1, %eax
13
          leave
14
                                                        24-27 = caller prologue
15
          ret
16 foo:
                                                        29 = caller epilogue
                    %ebp
           pushl
17
                                                        4-6 = callee prologue
                    %esp, %ebp
          movl
18
          pushl
                    %ebx
                                                        13-15 = callee epilogue
19
20
           subl
                    $4, %esp
                                                       foo as callee
           movl
                    8(%ebp), %edx
21
                                                        17-20 = callee prologue
          movl
                   12(%ebp), %eax
22
                   (%edx,%eax), %ebx
23
           leal
                                                        33-35 = callee epilogue
           subl
                    $4, %esp
24
25
           pushl
                    $5
          pushl
                    $4
26
```

```
call.s
                                         27
                                                    pushl
                                                            $3
                                         28
                                                    call
                                                            bar
1 p:
                                                    addl
                                                            $16, %esp
2
          .zero
                  4
                                         29
                                                    addl
3 bar:
                                         30
                                                            %ebx, %eax
          pushl
                  %ebp
                                         31
                                                    movl
                                                            %eax, p
4
          movl
                  %esp, %ebp
5
                                         32
                                                    nop
6
          subl
                  $16, %esp
                                         33
                                                    movl
                                                            -4(%ebp), %ebx
                  8(%ebp), %edx
7
          movl
                                         34
                                                    leave
                  12(%ebp), %eax
          movl
                                         35
                                                    ret
8
          addl
                  %edx, %eax
9
          subl
                 16(%ebp), %eax
                                                   What does line 19 do in call.s? Why is
10
          movl
                 %eax, -4(%ebp)
11
                                                   it necessary?
                  -4(%ebp), %eax
          movl
12
13
          addl
                  $1, %eax
          leave
14
15
          ret
16 foo:
          pushl
                  %ebp
17
          movl
                  %esp, %ebp
18
          pushl
                  %ebx
19
          subl
20
                  $4, %esp
          movl
                  8(%ebp), %edx
21
                  12(%ebp), %eax
22
          movl
                  (%edx,%eax), %ebx
23
          leal
          subl
                  $4, %esp
24
25
          pushl
                  $5
          pushl
                   $4
26
```

```
call.s
                                          27
                                                     pushl
                                                              $3
                                          28
                                                     call
                                                              bar
1 p:
                                                     addl
                                                              $16, %esp
2
          .zero
                   4
                                          29
                                                     addl
3 bar:
                                          30
                                                              %ebx, %eax
          pushl
                   %ebp
                                          31
                                                     movl
                                                              %eax, p
4
          movl
                   %esp, %ebp
5
                                          32
                                                     nop
6
          subl
                   $16, %esp
                                          33
                                                     movl
                                                              -4(%ebp), %ebx
                   8(%ebp), %edx
          movl
                                          34
                                                     leave
                  12(%ebp), %eax
          movl
                                          35
                                                      ret
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
                                                    What does line 19 do in call.s? Why is
10
          movl
                 %eax, -4(%ebp)
11
                                                     it necessary?
                  -4(%ebp), %eax
          movl
12
                                                     Saves EBX register since it's callee-saved
13
          addl
                   $1, %eax
          leave
14
                                                     and foo uses it. Doesn't matter that bar
15
          ret
16 foo:
                                                     never uses EBX.
          pushl
                   %ebp
17
          movl
                   %esp, %ebp
18
          pushl
                   %ebx
19
20
          subl
                   $4, %esp
          movl
                   8(%ebp), %edx
21
22
          movl
                   12(%ebp), %eax
                   (%edx,%eax), %ebx
23
          leal
                   $4, %esp
          subl
24
25
          pushl
                   $5
          pushl
                   $4
26
```

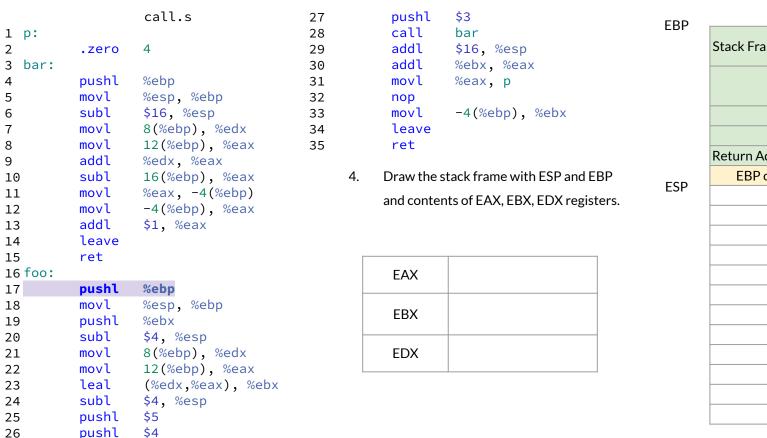
```
call.s
                                         27
                                                    pushl
                                                             $3
                                         28
                                                    call
                                                             bar
1 p:
                                                    addl
                                                             $16, %esp
2
          .zero
                   4
                                         29
                                                    addl
3 bar:
                                         30
                                                             %ebx, %eax
          pushl
                  %ebp
                                         31
                                                    movl
                                                             %eax, p
4
          movl
                  %esp, %ebp
5
                                         32
                                                    nop
6
          subl
                  $16, %esp
                                         33
                                                    movl
                                                             -4(%ebp), %ebx
                  8(%ebp), %edx
          movl
                                         34
                                                    leave
                  12(%ebp), %eax
          movl
                                         35
                                                     ret
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
                                                   Why is EDX not saved by foo before
10
          movl
                  %eax, -4(%ebp)
11
                                                   calling bar despite the register being
                  -4(%ebp), %eax
          movl
12
                                                   overwritten in bar?
13
          addl
                  $1, %eax
          leave
14
15
          ret
16 foo:
          pushl
                  %ebp
17
          movl
                  %esp, %ebp
18
          pushl
                  %ebx
19
          subl
20
                  $4, %esp
21
          movl
                  8(%ebp), %edx
22
          movl
                  12(%ebp), %eax
                  (%edx,%eax), %ebx
23
          leal
          subl
                  $4, %esp
24
25
          pushl
                  $5
          pushl
                   $4
26
```

```
call.s
                                          27
                                                     pushl
                                                              $3
                                          28
                                                     call
                                                              bar
1 p:
                                                     addl
                                                              $16, %esp
2
          .zero
                   4
                                          29
                                                     addl
3 bar:
                                          30
                                                              %ebx, %eax
          pushl
                   %ebp
                                          31
                                                     movl
                                                              %eax, p
4
          movl
                  %esp, %ebp
5
                                          32
                                                     nop
6
          subl
                  $16, %esp
                                          33
                                                     movl
                                                              -4(%ebp), %ebx
                  8(%ebp), %edx
          movl
                                          34
                                                     leave
                  12(%ebp), %eax
          movl
                                          35
                                                     ret
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
                                                    Why is EDX not saved by foo before
10
                 %eax, -4(%ebp)
          movl
11
                                                    calling bar despite the register being
                  -4(%ebp), %eax
          movl
12
                                                    overwritten in bar?
13
          addl
                   $1, %eax
          leave
14
                                                    EDX does not need to persist after the
15
          ret
16 foo:
                                                    function call.
          pushl
                   %ebp
17
          movl
                  %esp, %ebp
18
          pushl
                  %ebx
19
          subl
20
                   $4, %esp
21
          movl
                  8(%ebp), %edx
22
          movl
                  12(%ebp), %eax
          leal
                   (%edx,%eax), %ebx
23
                   $4, %esp
          subl
24
25
          pushl
                   $5
          pushl
                   $4
26
```

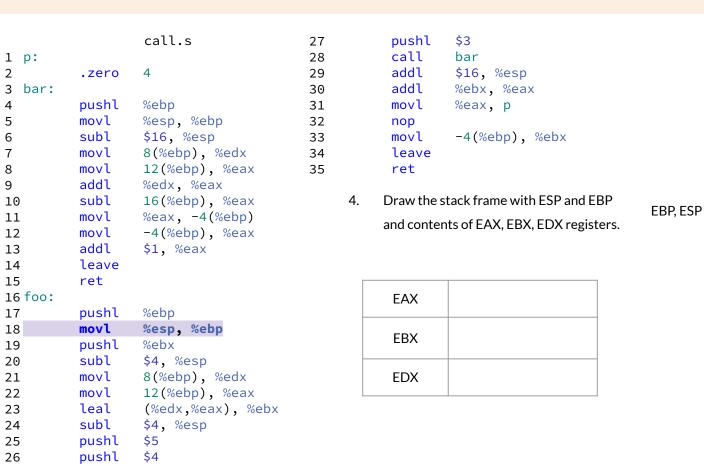
						1 -			
		call.s	27		pushl	\$3		EBP	
1 p:			28		call	bar		25.	C. I
2	.zero	4	29		addl	\$16, %esp			Stack
3 bar:			30		addl	<pre>%ebx, %eax</pre>		ESP	
4	pushl	%ebp	31		movl	%eax, p			
5	movl	%esp, %ebp	32		nop				
6	subl	\$16, %esp	33		movl	-4(%ebp), %ebx			
7	movl	8(%ebp), %edx	34		leave				
8	movl	12(%ebp), %eax	35		ret				
9	addl	%edx, %eax							
10	subl	16(%ebp), %eax		4.	Draw the s	stack frame with ESP and	EBP		
11	movl	<pre>%eax, -4(%ebp)</pre>			and conton	ata of EAV EDV EDV root	at a 4 a		
12	movl	-4(%ebp), %eax			and conter	nts of EAX, EBX, EDX regi	sters.		
13	addl	\$1, %eax							
14	leave								
15	ret						\neg		
16 foo:					EAX				
17	pushl	%ebp							
18	movl	%esp, %ebp							
19	pushl	%ebx			EBX				
20	subl	\$4, %esp					_		
21	movl	8(%ebp), %edx			EDX				
22	movl	12(%ebp), %eax							
23	leal	(%edx,%eax), %ebx							
24	subl	\$4, %esp							
25	pushl	\$5							
26	pushl	\$4							
20	pusiic	YT							

ck Frame of foo's Caller

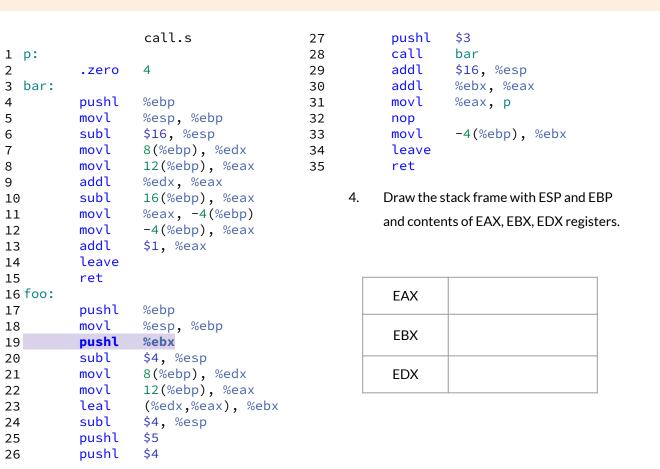
		call.s	27	pushl	\$3	ED C	
1 p:			28	call	bar	EBP	
2	.zero	4	29	addl	\$16, %esp		Stack Frame of foo's Caller
3 bar:			30	addl	%ebx, %eax		
4	pushl	%ebp	31	movl	%eax, p		Padding
5	movl	%esp, %ebp	32	nop			i adding
6	subl	\$16, %esp	33	movl	-4(%ebp), %ebx		b
7	movl	8(%ebp), %edx	34	leave			a
8	movl	12(%ebp), %eax	35	ret			
9	addl	<pre>%edx, %eax</pre>				ESP	Return Addr of foo's Caller
10	subl	16(%ebp), %eax	2	4. Draw the s	tack frame with ESP and EBP		
11	movl	%eax, -4(%ebp)		and conter	its of EAX, EBX, EDX registers	i.	
12	movl	-4(%ebp), %eax		GG. 5555.		•	
13	addl	\$1, %eax					
14	leave						
15	ret						
16 foo:		0.1		EAX			
17	pushl	%ebp					
18	movl	%esp, %ebp		EBX			
19	pushl	%ebx					
20	subl	\$4, %esp		EDV			
21	movl	8(%ebp), %edx		EDX			
22	movl	12(%ebp), %eax					
23	leal	(%edx,%eax), %ebx					
24	subl	\$4, %esp					
25	pushl	\$5					
26	pushl	\$4					



Stack Frame of foo's Caller **Padding** b а Return Addr of foo's Caller EBP of foo's Caller

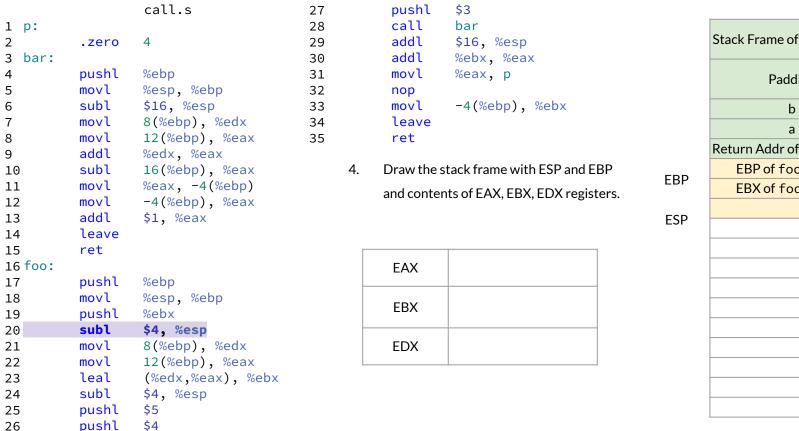


Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller

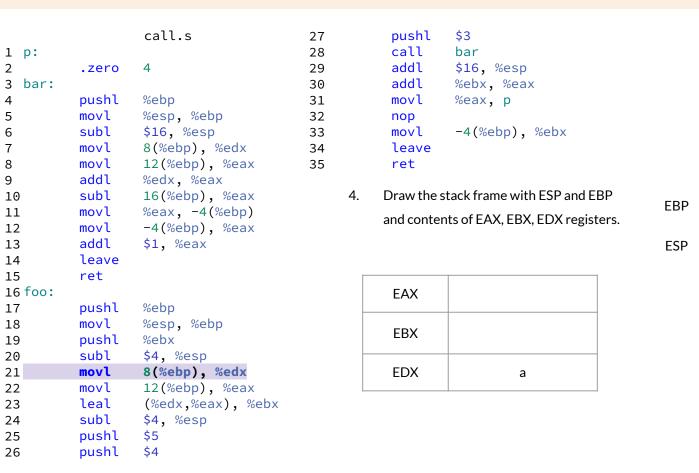


Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller

EBP



Stack Frame of foo's Caller **Padding** Return Addr of foo's Caller EBP of foo's Caller EBX of foo's Caller



Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller

```
call.s
                                          27
                                                     pushl
                                                              $3
                                          28
                                                     call
                                                              bar
1 p:
                                                     addl
                                                              $16, %esp
2
          .zero
                   4
                                          29
                                                     addl
                                                              %ebx, %eax
3 bar:
                                          30
          pushl
                   %ebp
                                          31
                                                     movl
                                                              %eax, p
4
                  %esp, %ebp
          movl
5
                                          32
                                                     nop
6
          subl
                  $16, %esp
                                          33
                                                     movl
                                                              -4(%ebp), %ebx
                  8(%ebp), %edx
                                                     leave
          movl
                                          34
          movl
                  12(%ebp), %eax
                                          35
                                                     ret
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
                                                    Draw the stack frame with FSP and FBP
10
          movl
                  %eax, -4(%ebp)
11
                                                    and contents of EAX, EBX, EDX registers.
          movl
                  -4(%ebp), %eax
12
          addl
                   $1, %eax
13
          leave
14
15
          ret
16 foo:
                                                     EAX
                                                                       b
          pushl
17
                   %ebp
          movl
                   %esp, %ebp
18
                                                     FBX
          pushl
                   %ebx
19
          subl
                   $4, %esp
20
21
          movl
                   8(%ebp), %edx
                                                     EDX
                                                                       а
                   12(%ebp), %eax
22
          movl
          leal
                   (%edx,%eax), %ebx
23
24
          subl
                   $4, %esp
25
          pushl
                   $5
          pushl
                   $4
26
```

Stack Frame of foo's Caller **Padding** b а Return Addr of foo's Caller EBP of foo's Caller EBX of foo's Caller

EBP

```
call.s
                                          27
                                                     pushl
                                                              $3
                                          28
                                                     call
                                                              bar
1 p:
                                                     addl
                                                              $16, %esp
2
          .zero
                   4
                                          29
                                                     addl
3 bar:
                                          30
                                                              %ebx, %eax
          pushl
                   %ebp
                                          31
                                                     movl
                                                              %eax, p
4
          movl
                   %esp, %ebp
5
                                          32
                                                     nop
6
          subl
                   $16, %esp
                                          33
                                                     movl
                                                              -4(%ebp), %ebx
                   8(%ebp), %edx
                                                     leave
          movl
                                          34
          movl
                   12(%ebp), %eax
                                          35
                                                     ret
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
                                                    Draw the stack frame with FSP and FBP
10
          movl
                  %eax, -4(%ebp)
11
                                                    and contents of EAX, EBX, EDX registers.
                   -4(%ebp), %eax
          movl
12
          addl
13
                   $1, %eax
          leave
14
15
          ret
16 foo:
                                                     EAX
                                                                       b
          pushl
17
                   %ebp
          movl
                   %esp, %ebp
18
                                                     EBX
                                                                     a + b
          pushl
                   %ebx
19
          subl
20
                   $4, %esp
21
          movl
                   8(%ebp), %edx
                                                     EDX
                                                                       а
                   12(%ebp), %eax
22
          movl
          leal
                   (%edx,%eax), %ebx
23
24
          subl
                   $4, %esp
25
          pushl
                   $5
          pushl
                   $4
26
```

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller

EBP

```
call.s
                                          27
                                                     pushl
                                                              $3
                                          28
                                                     call
                                                              bar
1 p:
                                                     addl
                                                              $16, %esp
2
          .zero
                   4
                                          29
                                                     addl
3 bar:
                                          30
                                                              %ebx, %eax
          pushl
                   %ebp
                                          31
                                                     movl
                                                              %eax, p
4
          movl
                  %esp, %ebp
5
                                          32
                                                     nop
6
          subl
                  $16, %esp
                                          33
                                                     movl
                                                              -4(%ebp), %ebx
                  8(%ebp), %edx
                                                     leave
          movl
                                          34
          movl
                  12(%ebp), %eax
                                          35
                                                     ret
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
                                                    Draw the stack frame with FSP and FBP
10
                  %eax, -4(%ebp)
          movl
11
                                                    and contents of EAX, EBX, EDX registers.
                  -4(%ebp), %eax
          movl
12
          addl
13
                   $1, %eax
          leave
14
15
          ret
16 foo:
                                                     EAX
                                                                       b
          pushl
17
                   %ebp
                  %esp, %ebp
          movl
18
                                                     EBX
                                                                     a + b
          pushl
                   %ebx
19
          subl
20
                   $4, %esp
21
          movl
                  8(%ebp), %edx
                                                     EDX
                                                                       а
                  12(%ebp), %eax
22
          movl
                  (%edx,%eax), %ebx
          leal
23
                   $4, %esp
24
          subl
25
          pushl
                   $5
          pushl
                   $4
26
```

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller

EBP

```
call.s
1 p:
2
          .zero
                  4
3 bar:
          pushl
                  %ebp
4
          movl
                  %esp, %ebp
5
6
          subl
                  $16, %esp
                  8(%ebp), %edx
          movl
          movl
                  12(%ebp), %eax
8
          addl
                 %edx, %eax
9
          subl
                 16(%ebp), %eax
10
          movl
                 %eax, -4(%ebp)
11
                  -4(%ebp), %eax
          movl
12
          addl
13
                  $1, %eax
          leave
14
15
          ret
16 foo:
          pushl
17
                  %ebp
          movl
                  %esp, %ebp
18
          pushl
                  %ebx
19
          subl
                  $4, %esp
20
21
          movl
                  8(%ebp), %edx
                  12(%ebp), %eax
22
          movl
          leal
                 (%edx,%eax), %ebx
23
                  $4, %esp
24
          subl
                  $5
25
          pushl
          pushl
                  $4
26
```

pushl call addl addl movl nop	\$3 bar \$16, %esp %ebx, %eax %eax, p
movl leave ret	-4(%ebp), %ebx

27 28

29

30

31

32

33

34

35

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	b
EBX	a + b
EDX	a

Stack Frame of foo's Caller						
Padding						
b						
a						
Return Addr of foo's Caller						
EBP of foo's Caller						
EBX of foo's Caller						
5						

EBP

```
call.s
                                        27
                                        28
1 p:
2
          .zero
                  4
                                        29
3 bar:
                                        30
          pushl
                  %ebp
                                        31
4
          movl
                  %esp, %ebp
5
                                        32
6
          subl
                  $16, %esp
                                        33
                  8(%ebp), %edx
          movl
                                        34
          movl
                  12(%ebp), %eax
                                        35
8
          addl
                 %edx, %eax
9
          subl
                 16(%ebp), %eax
10
          movl
                 %eax, -4(%ebp)
11
          movl
                  -4(%ebp), %eax
12
          addl
                  $1, %eax
13
          leave
14
15
          ret
16 foo:
          pushl
17
                  %ebp
          movl
                  %esp, %ebp
18
          pushl
                  %ebx
19
          subl
                  $4, %esp
20
21
          movl
                  8(%ebp), %edx
                  12(%ebp), %eax
22
          movl
          leal
                  (%edx,%eax), %ebx
23
                  $4, %esp
24
          subl
25
          pushl
                  $5
26
          pushl
```

pushl	\$3
call	bar
addl	\$16, %esp
addl	%ebx, %eax
movl	%eax, p
nop movl leave ret	-4(%ebp), %ebx

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	b
EBX	a + b
EDX	a

Stack Frame of foo's Caller						
Padding						
b						
a						
Return Addr of foo's Caller						
EBP of foo's Caller						
EBX of foo's Caller						
5						
4						

EBP

		call.s	27		pushl	
1 p:			28		call	
2	.zero	4	29		addl	
3 bar:			30		addl	
4	pushl	%ebp	31		movl	
5	movl	%esp, %ebp	32		nop	
6	subl	\$16, %esp	33		movl	
7	movl	8(%ebp), %edx	34		leave	
8	movl	12(%ebp), %eax	35		ret	
9	addl	<pre>%edx, %eax</pre>				
10	subl	16(%ebp), %eax		4.	Draw the st	į
11	movl	<pre>%eax, -4(%ebp)</pre>			and conten	+
12	movl	-4(%ebp), %eax			and conten	ι
13	addl	\$1, %eax				
14	leave					
15	ret					Γ
16 foo:					EAX	
17	pushl	%ebp				L
18	movl	%esp, %ebp			EDV	
19	pushl	%ebx			EBX	
20	subl	\$4, %esp		-		r
21	movl	8(%ebp), %edx			EDX	
22	movl	12(%ebp), %eax		L		L
23	leal	(%edx,%eax), %ebx				
24	subl	\$4, %esp				
25	pushl	\$5				
26	pushl	\$4				

27	pushl	\$3
28	call	bar
29	addl	\$16, %esp
30	addl	<pre>%ebx, %eax</pre>
31	movl	%eax, p
32	nop	
33	movl	-4(%ebp), %ebx
34	leave	
35	ret	

tack frame with ESP and EBP nts of EAX, EBX, EDX registers.

EAX	b
EBX	a + b
EDX	а

Stack Frame of foo's Caller						
Padding						
b						
a						
Return Addr of foo's Caller						
EBP of foo's Caller						
EBX of foo's Caller						
5						
4						
3						

EBP

1 p:		call.s	27 28		pushl call	\$3 bar
2	.zero	4	29		addl	\$16, %esp
3 bar:			30		addl	%ebx, %eax
4	pushl	%ebp	31		movl	%eax, p
5	movl	%esp, %ebp	32		nop_	
6	subl	\$16, %esp	33		movl	-4(%ebp), %ebx
7	movl		34		leave	
8	movl	12(%ebp), %eax	35		ret	
9	addl	%edx, %eax				
10	subl	16(%ebp), %eax		4.	Draw the st	tack frame with ESP and EBP
11		%eax, -4(%ebp)			and conten	ts of EAX, EBX, EDX registers.
12		-4(%ebp), %eax				, , ,
13		\$1, %eax				
14	leave					
15	ret					
16 foo:					EAX	b b
17	pushl	%ebp				
18	movl	%esp, %ebp			EBX	a+b
19	pushl	%ebx			LDA	
20	subl	\$4, %esp				
21		8(%ebp), %edx			EDX	a
22	movl	12(%ebp), %eax				
23		(%edx,%eax), %ebx				
24	subl	\$4, %esp				
25	pushl	\$5				
26	pushl	\$4				

Stack Frame of foo's Caller **Padding** b а Return Addr of foo's Caller EBP of foo's Caller EBX of foo's Caller 5 4 3 Return Addr of foo

EBP

```
call.s
                                          27
                                                     pushl
                                                              $3
                                          28
                                                     call
                                                              bar
1 p:
                                                             $16, %esp
                                                     addl
2
          .zero
                   4
                                          29
                                                     addl
3 bar:
                                          30
                                                             %ebx, %eax
                   %ebp
          pushl
                                          31
                                                     movl
                                                              %eax, p
4
          movl
                   %esp, %ebp
5
                                          32
                                                     nop
6
          subl
                   $16, %esp
                                          33
                                                     movl
                                                              -4(%ebp), %ebx
                   8(%ebp), %edx
                                                     leave
          movl
                                          34
          movl
                  12(%ebp), %eax
                                          35
                                                     ret
8
          addl
                  %edx, %eax
9
                                                    Draw the stack frame with FSP and FBP
          subl
                  16(%ebp), %eax
10
          movl
                  %eax, -4(%ebp)
11
                                                    and contents of EAX, EBX, EDX registers.
                  -4(%ebp), %eax
          movl
12
13
          addl
                   $1, %eax
          leave
14
15
          ret
16 foo:
                                                     EAX
                                                                       b
          pushl
                   %ebp
17
          movl
                   %esp, %ebp
18
                                                     FBX
                                                                     a + b
          pushl
                   %ebx
19
          subl
                   $4, %esp
20
21
          movl
                  8(%ebp), %edx
                                                     EDX
                                                                       а
                  12(%ebp), %eax
22
          movl
          leal
                  (%edx,%eax), %ebx
23
          subl
                   $4, %esp
24
25
          pushl
                   $5
          pushl
                   $4
26
```

Stack Frame of foo's Caller **Padding** b а Return Addr of foo's Caller EBP of foo's Caller EBX of foo's Caller 5 4 3 Return Addr of foo EBP of foo

EBP

```
call.s
1 p:
2
          .zero
                  4
3 bar:
         pushl
                  %ebp
4
         movl
                  %esp, %ebp
5
6
          subl
                  $16, %esp
                  8(%ebp), %edx
         movl
                  12(%ebp), %eax
         movl
8
          addl
                 %edx, %eax
9
          subl
                 16(%ebp), %eax
10
         movl
                %eax, -4(%ebp)
11
                 -4(%ebp), %eax
         movl
12
13
          addl
                  $1, %eax
         leave
14
15
          ret
16 foo:
          pushl
                  %ebp
17
         movl
                  %esp, %ebp
18
         pushl
                  %ebx
19
          subl
20
                  $4, %esp
21
         movl
                  8(%ebp), %edx
                 12(%ebp), %eax
22
         movl
         leal
                 (%edx,%eax), %ebx
23
                  $4, %esp
          subl
24
25
          pushl
                  $5
          pushl
                  $4
26
```

```
pushl $3
call bar
addl $16, %esp
addl %ebx, %eax
movl %eax, p
nop
movl -4(%ebp), %ebx
leave
ret
```

27

28

29

30

31

32

33

34

35

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	b
EBX	a + b
EDX	a

Stack Frame of foo's Caller **Padding** b а Return Addr of foo's Caller EBP of foo's Caller EBX of foo's Caller 5 4 3 Return Addr of foo EBP of foo

EBP, ESP

```
call.s
                                        27
                                        28
1 p:
2
          .zero
                  4
                                        29
3 bar:
                                        30
         pushl
                  %ebp
                                        31
4
         movl
                  %esp, %ebp
5
                                        32
6
         subl
                  $16, %esp
                                        33
                  8(%ebp), %edx
         movl
                                        34
                  12(%ebp), %eax
         movl
                                        35
8
          addl
                 %edx, %eax
9
          subl
                 16(%ebp), %eax
10
         movl
                 %eax, -4(%ebp)
11
                 -4(%ebp), %eax
         movl
12
13
          addl
                  $1, %eax
         leave
14
15
          ret
16 foo:
          pushl
17
                  %ebp
         movl
                  %esp, %ebp
18
         pushl
                  %ebx
19
          subl
20
                  $4, %esp
21
         movl
                 8(%ebp), %edx
                 12(%ebp), %eax
22
         movl
                 (%edx,%eax), %ebx
23
         leal
                  $4, %esp
24
          subl
25
          pushl
                  $5
          pushl
                  $4
26
```

```
pushl $3
call bar
addl $16, %esp
addl %ebx, %eax
movl %eax, p
nop
movl -4(%ebp), %ebx
leave
ret
```

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	b
EBX	a + b
EDX	a

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller
5
4
3
Return Addr of foo
EBP of foo

EBP

```
call.s
1 p:
2
          .zero
                  4
3 bar:
          pushl
                  %ebp
4
          movl
                  %esp, %ebp
5
6
          subl
                  $16, %esp
7
                  8(%ebp), %edx
          movl
                  12(%ebp), %eax
          movl
                                        35
8
          addl
                  %edx, %eax
9
          subl
                 16(%ebp), %eax
10
          movl
                 %eax, -4(%ebp)
11
                  -4(%ebp), %eax
          movl
12
13
          addl
                  $1, %eax
          leave
14
15
          ret
16 foo:
          pushl
17
                  %ebp
                  %esp, %ebp
          movl
18
          pushl
                  %ebx
19
          subl
20
                  $4, %esp
21
          movl
                  8(%ebp), %edx
                  12(%ebp), %eax
22
          movl
                  (%edx,%eax), %ebx
23
          leal
                  $4, %esp
24
          subl
25
          pushl
                  $5
          pushl
                  $4
26
```

```
27
           pushl
                   $3
28
           call
                   bar
           addl
                   $16, %esp
29
           addl
30
                   %ebx, %eax
31
           movl
                   %eax, p
32
           nop
33
           movl
                   -4(%ebp), %ebx
34
           leave
           ret
```

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	b
EBX	a + b
EDX	3

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller
5
4
3
Return Addr of foo
EBP of foo

EBP

```
call.s
1 p:
2
          .zero
                  4
3 bar:
         pushl
                  %ebp
4
         movl
                  %esp, %ebp
5
6
          subl
                  $16, %esp
                  8(%ebp), %edx
         movl
                  12(%ebp), %eax
8
         movl
          addl
                  %edx, %eax
9
          subl
                 16(%ebp), %eax
10
         movl
                 %eax, -4(%ebp)
11
                  -4(%ebp), %eax
         movl
12
13
          addl
                  $1, %eax
         leave
14
15
          ret
16 foo:
          pushl
17
                  %ebp
                  %esp, %ebp
         movl
18
         pushl
                  %ebx
19
          subl
20
                  $4, %esp
21
         movl
                  8(%ebp), %edx
                 12(%ebp), %eax
22
         movl
                 (%edx,%eax), %ebx
23
         leal
                  $4, %esp
24
          subl
25
          pushl
                  $5
          pushl
                  $4
26
```

```
pushl $3
call bar
addl $16, %esp
addl %ebx, %eax
movl %eax, p
nop
movl -4(%ebp), %ebx
leave
ret
```

27

28

29

30

31

32

33

34

35

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	4
EBX	a + b
EDX	3

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller
5
4
3
Return Addr of foo
EBP of foo

EBP

```
call.s
1 p:
2
          .zero
                  4
3 bar:
         pushl
                  %ebp
4
         movl
                  %esp, %ebp
5
6
          subl
                  $16, %esp
                  8(%ebp), %edx
         movl
                  12(%ebp), %eax
         movl
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
10
         movl
                 %eax, -4(%ebp)
11
                  -4(%ebp), %eax
         movl
12
13
          addl
                  $1, %eax
         leave
14
15
          ret
16 foo:
          pushl
17
                  %ebp
                  %esp, %ebp
         movl
18
         pushl
                  %ebx
19
          subl
20
                  $4, %esp
21
         movl
                 8(%ebp), %edx
                 12(%ebp), %eax
22
         movl
                 (%edx,%eax), %ebx
23
         leal
          subl
                  $4, %esp
24
25
          pushl
                  $5
          pushl
                  $4
26
```

```
pushl $3
call bar
addl $16, %esp
addl %ebx, %eax
movl %eax, p
nop
movl -4(%ebp), %ebx
leave
ret
```

27

28

29

30

31

32

33

34

35

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	7
EBX	a + b
EDX	3

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller
5
4
3
Return Addr of foo
EBP of foo

EBP

```
call.s
1 p:
2
          .zero
3 bar:
          pushl
                  %ebp
4
          movl
                  %esp, %ebp
5
6
          subl
                  $16, %esp
                  8(%ebp), %edx
          movl
          movl
                 12(%ebp), %eax
8
          addl
                 %edx, %eax
9
          subl
                 16(%ebp), %eax
10
          movl
                 %eax, -4(%ebp)
11
          movl
                  -4(%ebp), %eax
12
          addl
                  $1, %eax
13
          leave
14
15
          ret
16 foo:
          pushl
                  %ebp
17
          movl
                  %esp, %ebp
18
          pushl
                  %ebx
19
          subl
                  $4, %esp
20
21
          movl
                 8(%ebp), %edx
                 12(%ebp), %eax
22
          movl
          leal
                 (%edx,%eax), %ebx
23
                  $4, %esp
24
          subl
25
          pushl
                  $5
          pushl
                  $4
26
```

pushl call addl addl movl nop	\$3 bar \$16, %esp %ebx, %eax %eax, p
movl leave ret	-4(%ebp), %ebx

27 28

29

30

31

32

33

34

35

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	2
EBX	a + b
EDX	3

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller
5
4
3
Return Addr of foo
EBP of foo

EBP

```
call.s
                                        27
                                        28
1 p:
2
          .zero
                                        29
3 bar:
                                        30
          pushl
                  %ebp
4
                                        31
          movl
                  %esp, %ebp
5
                                        32
6
          subl
                  $16, %esp
                                        33
                  8(%ebp), %edx
          movl
                                        34
          movl
                  12(%ebp), %eax
                                        35
8
          addl
                 %edx, %eax
9
          subl
                 16(%ebp), %eax
10
          movl
                  %eax, -4(%ebp)
11
          movl
                  -4(%ebp), %eax
12
          addl
13
                  $1, %eax
          leave
14
15
          ret
16 foo:
          pushl
                  %ebp
17
          movl
                  %esp, %ebp
18
          pushl
                  %ebx
19
          subl
                  $4, %esp
20
21
          movl
                  8(%ebp), %edx
                  12(%ebp), %eax
22
          movl
          leal
                 (%edx,%eax), %ebx
23
                  $4, %esp
24
          subl
25
          pushl
                  $5
          pushl
                  $4
26
```

pushl	\$3
call	bar
addl	\$16, %esp
addl	<pre>%ebx, %eax</pre>
movl	%eax, p
nop	
movl	-4(%ebp), %ebx
leave	
ret	

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	2
EBX	a + b
EDX	3

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller
5
4
3
Return Addr of foo
EBP of foo
2

ESP

EBP

```
call.s
                                         27
                                                     pushl
                                                             $3
                                          28
                                                     call
                                                             bar
1 p:
                                                     addl
                                                             $16, %esp
2
                                         29
          .zero
                   4
3 bar:
                                                     addl
                                          30
                                                             %ebx, %eax
          pushl
                   %ebp
                                          31
                                                     movl
                                                             %eax, p
4
          movl
                  %esp, %ebp
5
                                          32
                                                     nop
6
          subl
                  $16, %esp
                                          33
                                                     movl
                                                             -4(%ebp), %ebx
                  8(%ebp), %edx
                                                     leave
          movl
                                         34
          movl
                  12(%ebp), %eax
                                         35
                                                     ret
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
                                                    Draw the stack frame with FSP and FBP
10
          movl
                  %eax, -4(%ebp)
11
                                                    and contents of EAX, EBX, EDX registers.
                   -4(%ebp), %eax
12
          movl
13
          addl
                   $1, %eax
          leave
14
15
          ret
16 foo:
                                                     EAX
          pushl
                  %ebp
17
          movl
                  %esp, %ebp
18
                                                     EBX
                                                                     a + b
          pushl
                  %ebx
19
          subl
                   $4, %esp
20
21
          movl
                  8(%ebp), %edx
                                                     EDX
                  12(%ebp), %eax
22
          movl
          leal
                  (%edx,%eax), %ebx
23
                   $4, %esp
24
          subl
25
          pushl
                   $5
          pushl
                   $4
26
```

2

3

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller
5
4
3
Return Addr of foo
EBP of foo
2

EBP

```
call.s
                                         27
                                                     pushl
                                                             $3
                                         28
                                                     call
                                                             bar
1 p:
                                                     addl
                                                             $16, %esp
2
          .zero
                   4
                                         29
3 bar:
                                                     addl
                                          30
                                                             %ebx, %eax
          pushl
                   %ebp
                                          31
                                                     movl
                                                             %eax, p
4
          movl
                  %esp, %ebp
5
                                          32
                                                     nop
6
          subl
                  $16, %esp
                                         33
                                                     movl
                                                             -4(%ebp), %ebx
                  8(%ebp), %edx
                                                     leave
          movl
                                         34
          movl
                  12(%ebp), %eax
                                         35
                                                     ret
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
                                                    Draw the stack frame with FSP and FBP
10
          movl
                  %eax, -4(%ebp)
11
                                                    and contents of EAX, EBX, EDX registers.
          movl
                  -4(%ebp), %eax
12
13
          addl
                   $1, %eax
          leave
14
15
          ret
16 foo:
                                                                      3
                                                     EAX
          pushl
                  %ebp
17
          movl
                  %esp, %ebp
18
                                                     EBX
                                                                     a + b
          pushl
                  %ebx
19
          subl
                   $4, %esp
20
21
          movl
                  8(%ebp), %edx
                                                     EDX
                                                                      3
                  12(%ebp), %eax
22
          movl
          leal
                  (%edx,%eax), %ebx
23
                   $4, %esp
24
          subl
25
          pushl
                   $5
          pushl
                   $4
26
```

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller
5
4
3
Return Addr of foo
EBP of foo
2

EBP

```
call.s
                                          27
                                                     pushl
                                                              $3
                                          28
                                                     call
                                                              bar
1 p:
                                                             $16, %esp
                                                     addl
2
          .zero
                   4
                                          29
                                                     addl
3 bar:
                                          30
                                                             %ebx, %eax
          pushl
                   %ebp
                                          31
                                                     movl
                                                              %eax, p
4
          movl
                  %esp, %ebp
5
                                          32
                                                     nop
6
          subl
                  $16, %esp
                                          33
                                                     movl
                                                              -4(%ebp), %ebx
                  8(%ebp), %edx
                                                     leave
          movl
                                          34
          movl
                  12(%ebp), %eax
                                          35
                                                     ret
8
          addl
                  %edx, %eax
9
                                                    Draw the stack frame with FSP and FBP
          subl
                  16(%ebp), %eax
10
                                                                                          EBP
          movl
                  %eax, -4(%ebp)
11
                                                    and contents of EAX, EBX, EDX registers.
                  -4(%ebp), %eax
          movl
12
13
          addl
                   $1, %eax
          leave
14
15
          ret
16 foo:
                                                                      3
                                                     EAX
          pushl
                   %ebp
17
          movl
                   %esp, %ebp
18
                                                     FBX
                                                                     a + b
          pushl
                   %ebx
19
                                                                                           ESP
          subl
                   $4, %esp
20
21
          movl
                  8(%ebp), %edx
                                                     EDX
                                                                      3
                  12(%ebp), %eax
22
          movl
          leal
                  (%edx,%eax), %ebx
23
24
          subl
                   $4, %esp
25
          pushl
                   $5
          pushl
                   $4
26
```

Stack Frame of foo's Caller **Padding** b а Return Addr of foo's Caller EBP of foo's Caller EBX of foo's Caller 5 4 3 Return Addr of foo EBP of foo 2

```
call.s
                                           27
                                                      pushl
                                                               $3
                                           28
                                                      call
                                                               bar
1 p:
                                                                                                    Stack Frame of foo's Caller
                                                               $16, %esp
2
                                                      addl
          .zero
                   4
                                           29
                                                      addl
3 bar:
                                           30
                                                               %ebx, %eax
          pushl
                   %ebp
                                           31
                                                      movl
                                                               %eax, p
4
          movl
                   %esp, %ebp
5
                                           32
                                                      nop
6
          subl
                   $16, %esp
                                           33
                                                      movl
                                                               -4(%ebp), %ebx
                   8(%ebp), %edx
                                                      leave
          movl
                                           34
          movl
                   12(%ebp), %eax
                                           35
                                                      ret
8
                                                                                                    Return Addr of foo's Caller
          addl
                   %edx, %eax
9
                                                     Draw the stack frame with FSP and FBP
          subl
                  16(%ebp), %eax
                                                                                                       EBP of foo's Caller
10
                                                                                             EBP
          movl
                  %eax, -4(%ebp)
11
                                                                                                       EBX of foo's Caller
                                                     and contents of EAX, EBX, EDX registers.
          movl
                   -4(%ebp), %eax
12
                   $1, %eax
13
          addl
          leave
14
15
          ret
16 foo:
                                                                        3
                                                       EAX
          pushl
17
                   %ebp
          movl
                   %esp, %ebp
18
                                                                                             ESP
                                                       FBX
                                                                       a + b
                                                                                                       Return Addr of foo
          pushl
                   %ebx
19
          subl
                   $4, %esp
20
21
          movl
                   8(%ebp), %edx
                                                      EDX
                                                                        3
                   12(%ebp), %eax
22
          movl
          leal
                   (%edx,%eax), %ebx
23
24
          subl
                   $4, %esp
25
          pushl
                   $5
          pushl
                   $4
26
```

Padding

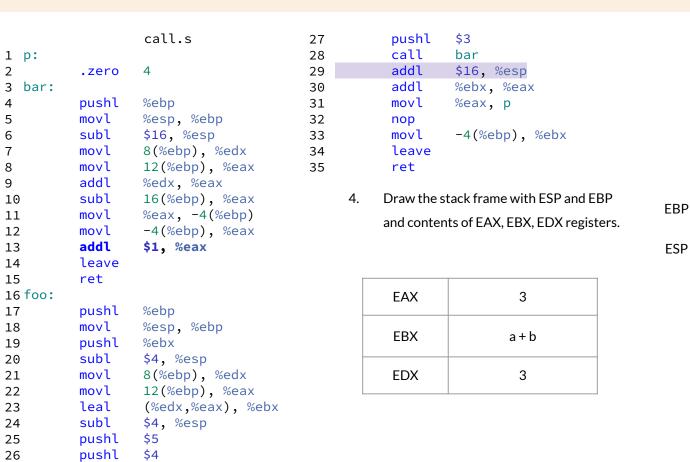
b

а

5

4

3



Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller
5
4
3

```
call.s
                                          27
                                                     pushl
                                                              $3
                                          28
                                                     call
                                                              bar
1 p:
                                                     addl
                                                              $16, %esp
2
          .zero
                   4
                                          29
                                                     addl
                                                              %ebx, %eax
3 bar:
                                          30
          pushl
                   %ebp
                                          31
                                                     movl
                                                              %eax, p
4
          movl
                  %esp, %ebp
5
                                          32
                                                     nop
6
          subl
                  $16, %esp
                                          33
                                                     movl
                                                              -4(%ebp), %ebx
                  8(%ebp), %edx
                                                     leave
          movl
                                          34
          movl
                  12(%ebp), %eax
                                          35
                                                     ret
8
          addl
                  %edx, %eax
9
                                                    Draw the stack frame with FSP and FBP
          subl
                  16(%ebp), %eax
10
          movl
                  %eax, -4(%ebp)
11
                                                    and contents of EAX, EBX, EDX registers.
          movl
                  -4(%ebp), %eax
12
                   $1, %eax
13
          addl
          leave
14
15
          ret
16 foo:
                                                     EAX
                                                                    3+a+b
          pushl
17
                   %ebp
          movl
                   %esp, %ebp
18
                                                     FBX
                                                                     a + b
          pushl
                   %ebx
19
          subl
                   $4, %esp
20
21
          movl
                  8(%ebp), %edx
                                                     EDX
                                                                       3
                  12(%ebp), %eax
22
          movl
          leal
                  (%edx,%eax), %ebx
23
                   $4, %esp
24
          subl
25
          pushl
                   $5
          pushl
                   $4
26
```

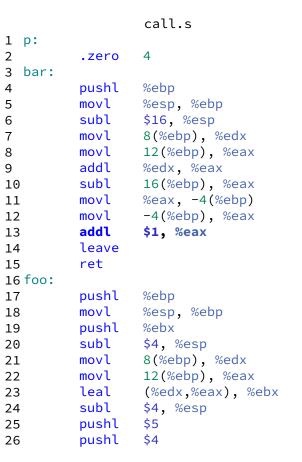
Stack Frame of foo's Caller **Padding** b а Return Addr of foo's Caller EBP of foo's Caller EBX of foo's Caller

EBP

```
call.s
                                          27
                                                     pushl
                                                              $3
                                          28
                                                     call
                                                              bar
1 p:
                                                     addl
                                                              $16, %esp
2
          .zero
                   4
                                          29
                                                     addl
3 bar:
                                          30
                                                              %ebx, %eax
          pushl
                   %ebp
                                          31
                                                     movl
                                                              %eax, p
4
          movl
                  %esp, %ebp
5
                                          32
                                                     nop
6
          subl
                  $16, %esp
                                          33
                                                     movl
                                                              -4(%ebp), %ebx
                  8(%ebp), %edx
                                                     leave
          movl
                                          34
                  12(%ebp), %eax
          movl
                                          35
                                                     ret
8
          addl
                  %edx, %eax
9
          subl
                  16(%ebp), %eax
                                                    Draw the stack frame with FSP and FBP
10
          movl
                  %eax, -4(%ebp)
11
                                                    and contents of EAX, EBX, EDX registers.
                  -4(%ebp), %eax
          movl
12
13
          addl
                   $1, %eax
          leave
14
15
          ret
16 foo:
                                                     EAX
                                                                    3+a+b
          pushl
17
                   %ebp
                  %esp, %ebp
          movl
18
                                                     EBX
                                                                EBX of foo's Caller
          pushl
                   %ebx
19
          subl
20
                   $4, %esp
21
          movl
                  8(%ebp), %edx
                                                     EDX
                                                                       3
                  12(%ebp), %eax
22
          movl
          leal
                  (%edx,%eax), %ebx
23
                   $4, %esp
24
          subl
25
          pushl
                   $5
          pushl
                   $4
26
```

Stack Frame of foo's Caller
Padding
b
a
Return Addr of foo's Caller
EBP of foo's Caller
EBX of foo's Caller

EBP



```
27
           pushl
                   $3
28
           call
                   bar
                   $16, %esp
           addl
29
           addl
30
                   %ebx, %eax
31
           movl
                   %eax, p
32
           nop
33
           movl
                   -4(%ebp), %ebx
           leave
34
35
           ret
```

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	3+a+b
EBX	EBX of foo's Caller
EDX	3

EBP

ESP

Stack Frame of foo's Caller **Padding** b а Return Addr of foo's Caller EBP of foo's Caller EBX of foo's Caller

```
1 p:
2
          .zero
3 bar:
          pushl
                  %ebp
4
          movl
                  %esp, %ebp
5
6
          subl
                  $16, %esp
                  8(%ebp), %edx
          movl
                 12(%ebp), %eax
          movl
8
          addl
                 %edx, %eax
9
          subl
                 16(%ebp), %eax
10
          movl
                 %eax, -4(%ebp)
11
                  -4(%ebp), %eax
          movl
12
13
          addl
                  $1, %eax
          leave
14
15
          ret
16 foo:
          pushl
17
                  %ebp
          movl
                  %esp, %ebp
18
          pushl
                  %ebx
19
          subl
20
                  $4, %esp
21
          movl
                  8(%ebp), %edx
                 12(%ebp), %eax
22
          movl
          leal
                 (%edx,%eax), %ebx
23
24
          subl
                  $4, %esp
25
          pushl
                  $5
          pushl
                  $4
26
```

call.s

```
27
           pushl
                   $3
28
           call
                   bar
           addl
                   $16, %esp
29
           addl
30
                   %ebx, %eax
31
           movl
                   %eax, p
32
           nop
33
           movl
                   -4(%ebp), %ebx
           leave
34
           ret
35
```

 Draw the stack frame with ESP and EBP and contents of EAX, EBX, EDX registers.

EAX	3+a+b
EBX	EBX of foo's Caller
EDX	3

EBP

Stack Frame of foo's Calle
Padding
b
a
Return Addr of foo's Calle