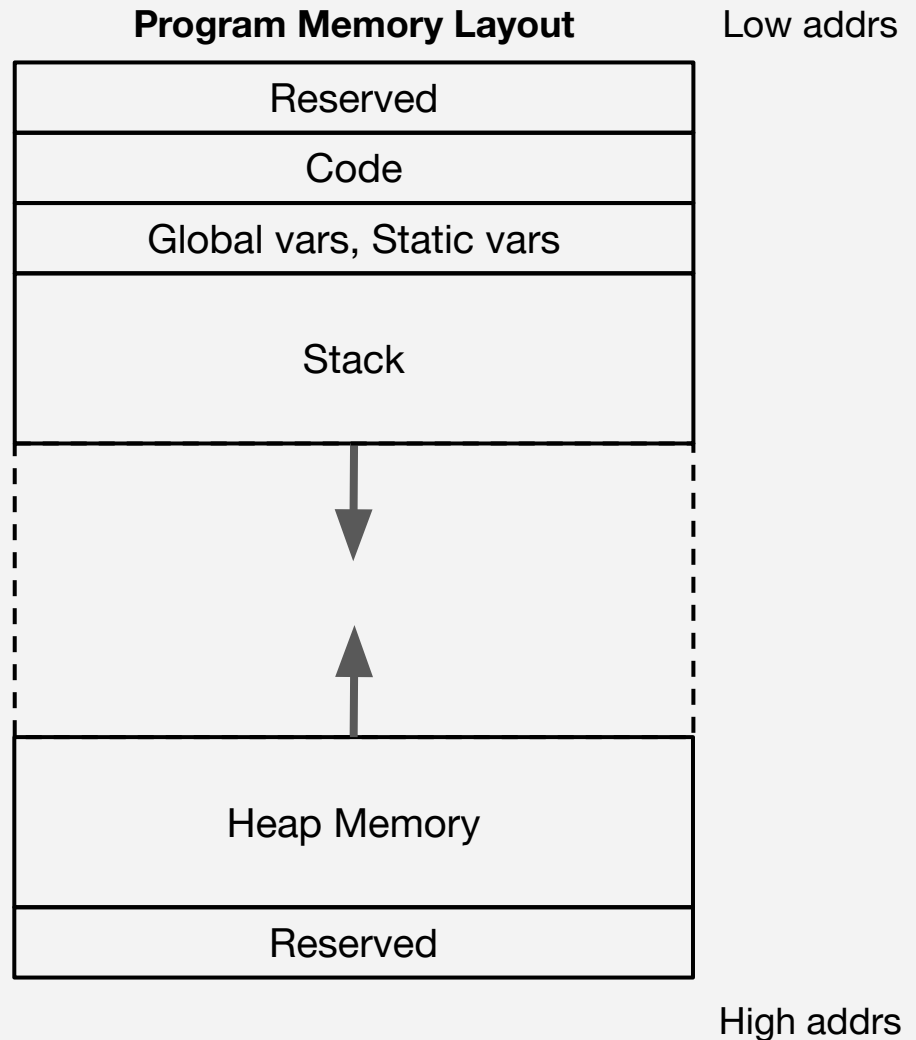# CSc 352

Malloc, Free, Stack, Heap

Benjamin Dicken

Program Layout

When a program is loaded into memory and run, this is the general memory organization and layout

Exact layout could change depending on the specific OS

Program Memory Layout

Reserved
Code
Global vars, Static vars
Stack

Heap Memory

Reserved

What could it print?

```c
#include <stdio.h>

void zeroes() {
  char z[10];
  for (int i = 0; i < 10; i++) {
    z[i] = 'z';
  }
}

void return_char_ptr(char ** r) {
  char z[10] = "abcdefghi";
  *r = z;
}

int main() {
  char * y;
  return_char_ptr(&y);
  printf(">%s<\n", y);
  zeroes();
  printf(">%s<\n", y);
  return 0;
}
```
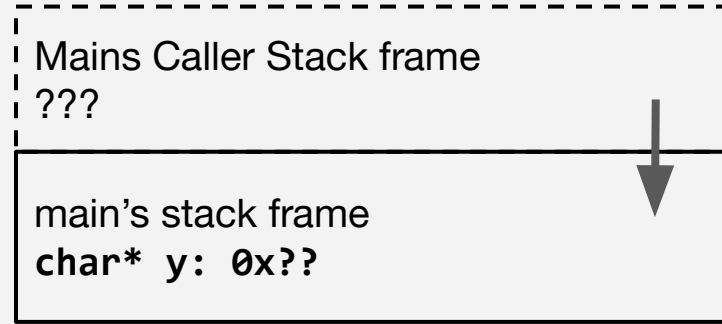
Stack Diagram

```c
#include <stdio.h>

void zeroes() {
  char z[10];
  for (int i = 0; i < 10; i++) {
    z[i] = 'z';
  }
}

void return_char_ptr(char ** r) {
  char z[10] = "abcdefghi";
  *r = z;
}

int main() {
  char * y;
  return_char_ptr(&y);
  printf(">%s<\n", y);
  zeroes();
  printf(">%s<\n", y);
  return 0;
}
```

Addresses

Mains Caller Stack frame
???

0x00..500 y

main's stack frame
`char* y: 0x??`

Stack Diagram

```c
#include <stdio.h>

void zeroes() {
  char z[10];
  for (int i = 0; i < 10; i++) {
    z[i] = 'z';
  }
}

void return_char_ptr(char ** r) {
  char z[10] = "abcdefghi";
  *r = z;
}

int main() {
  char * y;
  return_char_ptr(&y);
  printf(">%s<\n", y);
  zeroes();
  printf(">%s<\n", y);
  return 0;
}
```

Addresses

Mains Caller Stack frame
???

0x00..500 y

main's stack frame
char* y: 0x??

return_char_ptr's stack frame
char z[]: a b c d e f g h i \0
0x00..540 z
0x00..54a r
char** r: 0x00..500

Stack Diagram

```c
#include <stdio.h>

void zeroes() {
  char z[10];
  for (int i = 0; i < 10; i++) {
    z[i] = 'z';
  }
}

void return_char_ptr(char ** r) {
  char z[10] = "abcdefghi";
  *r = z;
}

int main() {
  char * y;
  return_char_ptr(&y);
  printf(">%s<\n", y);
  zeroes();
  printf(">%s<\n", y);
  return 0;
}
```
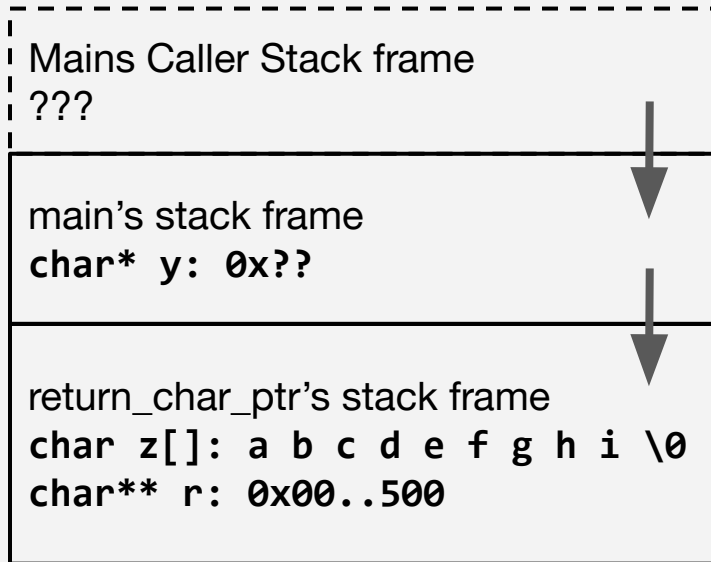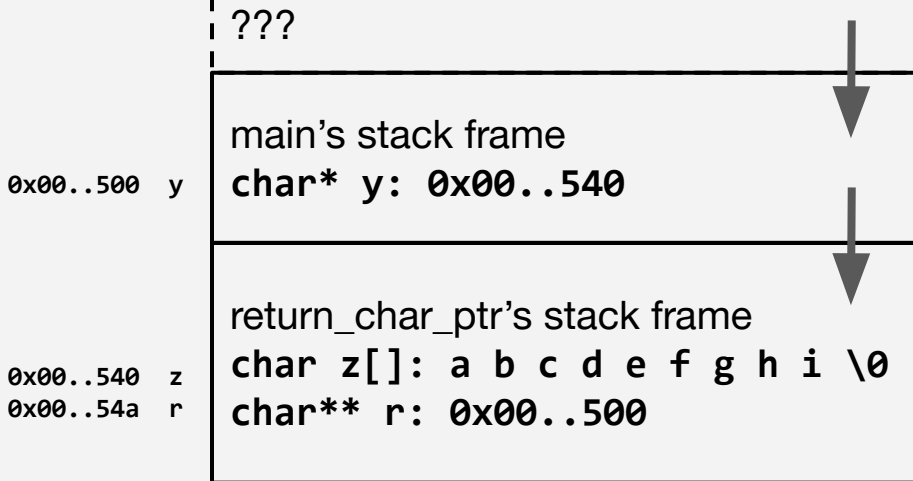
Addresses

Mains Caller Stack frame
???

main's stack frame
0x00..500 y   char* y: 0x00..540

return_char_ptr's stack frame
0x00..540 z   char z[]: a b c d e f g h i \0
0x00..54a r   char** r: 0x00..500

Stack Diagram

```c
#include <stdio.h>

void zeroes() {
  char z[10];
  for (int i = 0; i < 10; i++) {
    z[i] = 'z';
  }
}

void return_char_ptr(char ** r) {
  char z[10] = "abcdefghi";
  *r = z;
}

int main() {
  char * y;
  return_char_ptr(&y);
  printf(">%s<\n", y);
  zeroes();
  printf(">%s<\n", y);
  return 0;
}
```
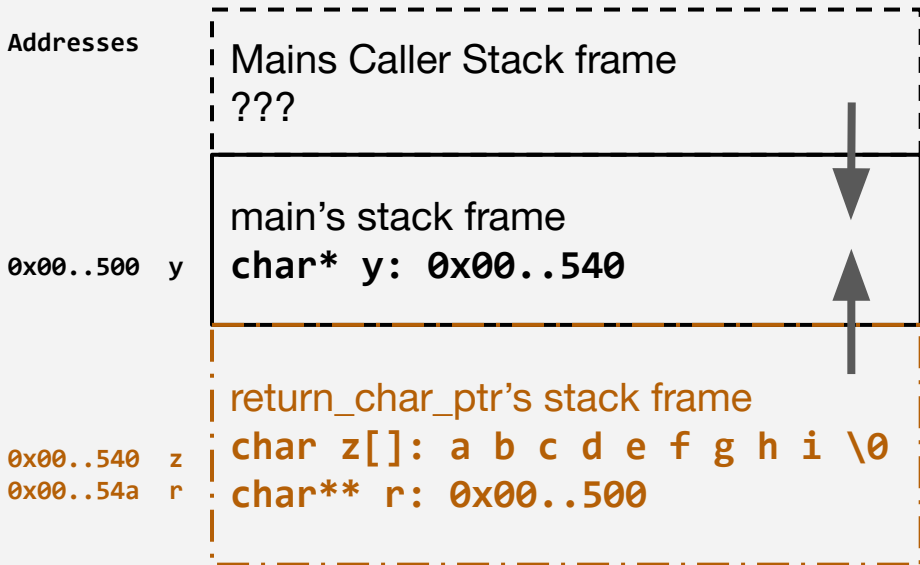
Addresses

Mains Caller Stack frame
???

0x00..500 y

main's stack frame
`char* y: 0x00..540`

return_char_ptr's stack frame

0x00..540 z
0x00..54a r

`char z[]: a b c d e f g h i \0`
`char** r: 0x00..500`

Stack Diagram

```c
#include <stdio.h>

void zeroes() {
  char z[10];
  for (int i = 0; i < 10; i++) {
    z[i] = 'z';
  }
}

void return_char_ptr(char ** r) {
  char z[10] = "abcdefghi";
  *r = z;
}

int main() {
  char * y;
  return_char_ptr(&y);
  printf(">%s<\n", y);
  zeroes();
  printf(">%s<\n", y);
  return 0;
}
```
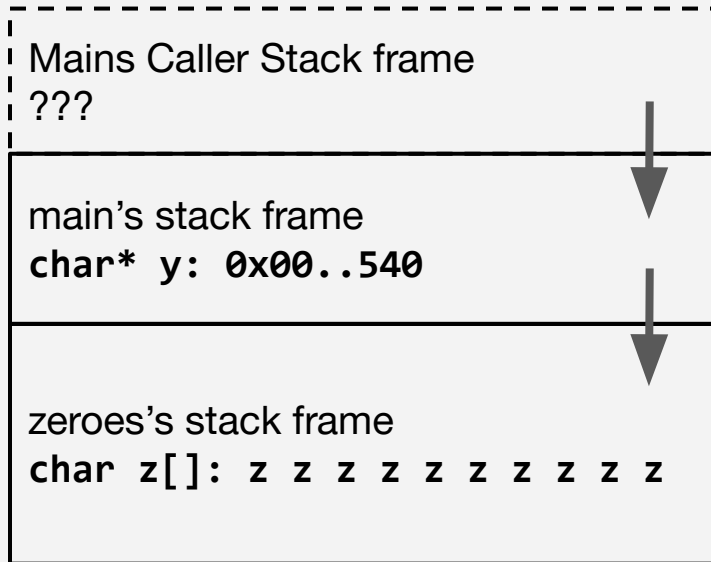
Addresses

Mains Caller Stack frame
???

main's stack frame
char* y: 0x00..540

0x00..500 y

zeroes's stack frame
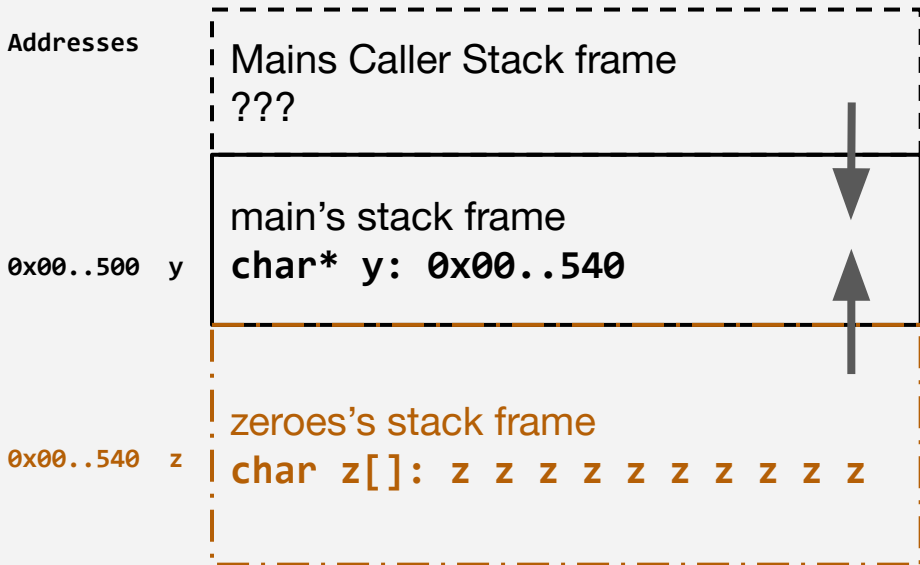**char z[]: z z z z z z z z z z**

0x00..540 z

Stack Diagram

```c
#include <stdio.h>

void zeroes() {
  char z[10];
  for (int i = 0; i < 10; i++) {
    z[i] = 'z';
  }
}

void return_char_ptr(char ** r) {
  char z[10] = "abcdefghi";
  *r = z;
}

int main() {
  char * y;
  return_char_ptr(&y);
  printf(">%s<\n", y);
  zeroes();
  printf(">%s<\n", y);
  return 0;
}
```
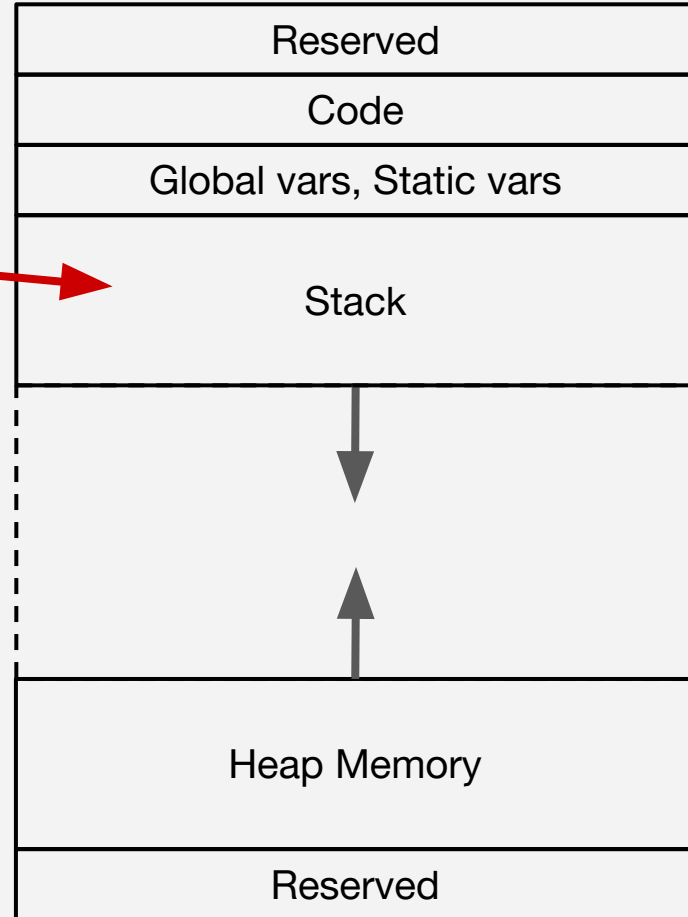
Addresses

Mains Caller Stack frame
???

0x00..500 y

main's stack frame
`char* y: 0x00..540`

0x00..540 z

zeroes's stack frame
`char z[]: z z z z z z z z z z`

Stack Memory

When we create and use local / parameter vars, allocation happens here

Program Memory Layout

Reserved
Code
Global vars, Static vars
Stack

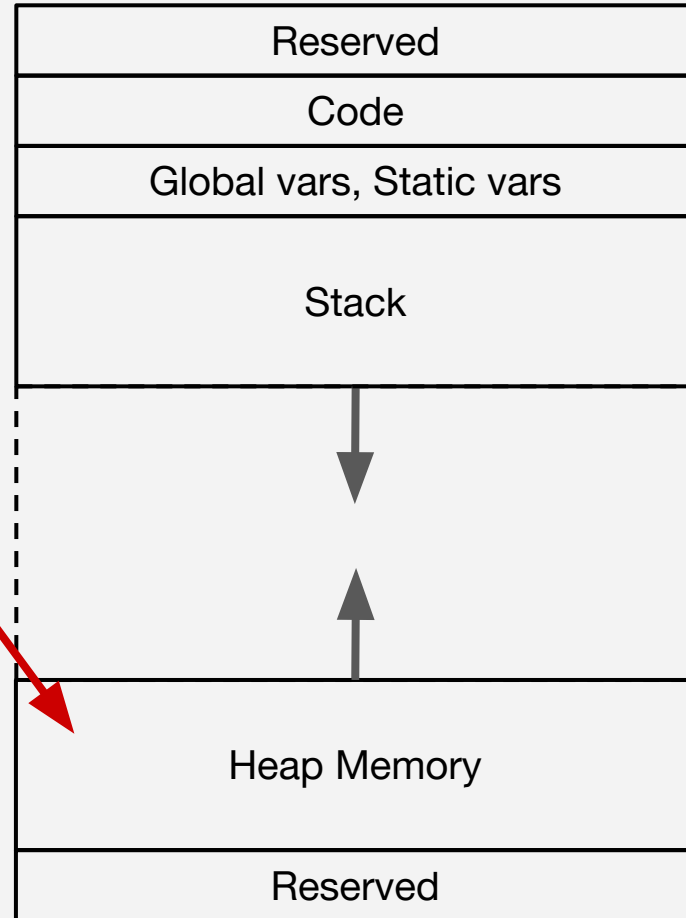| |
|---|
| Heap Memory |
| Reserved |

Malloc

The malloc function lets you allocate memory that is not on the stack! Finally!

```
void* malloc(size_t size);
```

Returns a void * (generic pointer) to a chunk of heap memory of **size** bytes, or NULL if malloc fails

Program Memory Layout

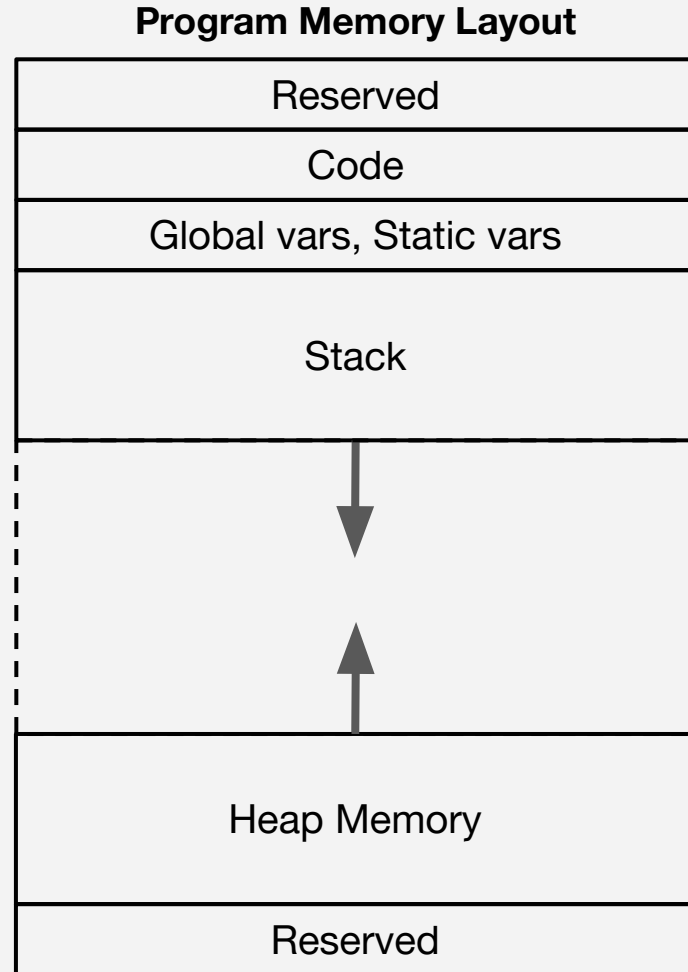| |
|---|
| Reserved |
| Code |
| Global vars, Static vars |
| Stack |
| |
| Heap Memory |
| Reserved |

Malloc

Malloc is useful for:

When you have a string / array whose lifetime extends beyond the function who created it's stack frame

Allocating space for data structures that will get passed around within the code

What else?

Program Memory Layout

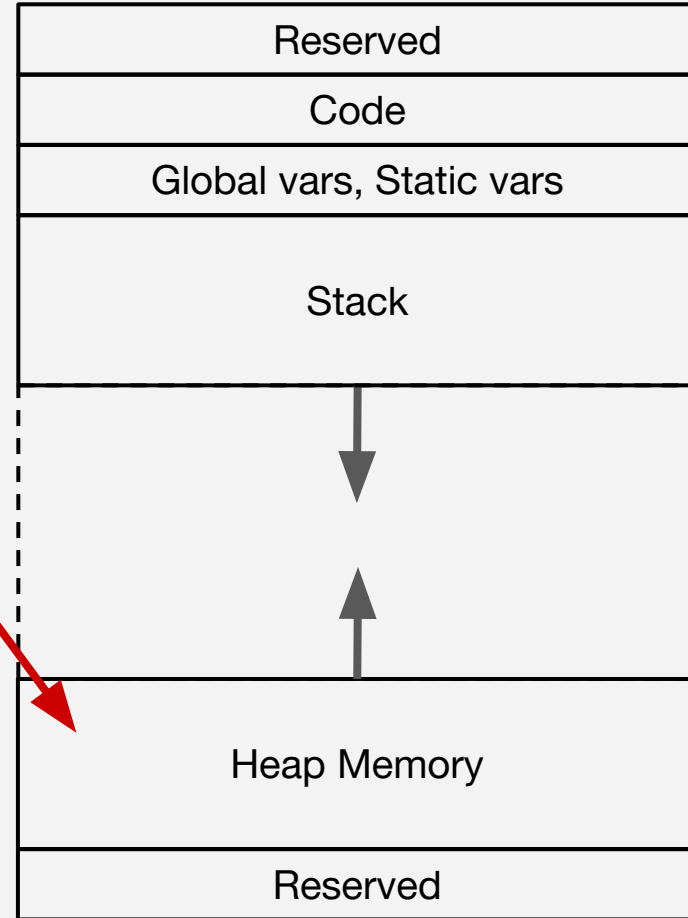| |
|---|
| Reserved |
| Code |
| Global vars, Static vars |
| Stack |
| Heap Memory |
| Reserved |

Free

The free function lets you de-allocate (free up) memory that was previously allocated with malloc

```
void free(void* ptr);
```

- Frees the memory.
- *EVERY* time you are finished will malloc'ed memory, you should call free
- If not, could have memory leak

Program Memory Layout

Reserved
Code
Global vars, Static vars
Stack
Heap Memory
Reserved

calloc

```
void* calloc(size_t n_items, size_t size);
```

allocates **n_items * size bytes**, initializes the data to zeroes

What will it print?

```c
#include <stdio.h>
#include <stdlib.h>
void zeroes() {
  char * z = malloc(10);
  for (int i = 0; i < 10; i++) {
    z[i] = 'z';
  }
}


void return_char_ptr(char ** r) {
  char * z = malloc(10);
  for (int i = 0; i < 10; i++) {
    z[i] = 'w';
  }
  *r = z;
}
```

```c
int main() {
  char * y;
  return_char_ptr(&y);
  printf(">%s<\n", y);
  zeroes();
  printf(">%s<\n", y);
  return 0;
}
```

What will it print?

```c
#include <stdio.h>
#include <stdlib.h>
void zeroes() {
  char * z = malloc(10);
  for (int i = 0; i < 10; i++) {
    z[i] = 'z';
  }
}


void return_char_ptr(char ** r) {
  char * z = malloc(10);
  for (int i = 0; i < 10; i++) {
    z[i] = 'w';
  }
  *r = z;
}
```

```c
int main() {
  char * y;
  return_char_ptr(&y);
  printf(">%s<\n", y);
  zeroes();
  printf(">%s<\n", y);
  return 0;
}
```

What am I
doing wrong?

What will this do?

```c
#include <stdio.h>
#include <stdlib.h>

void zeroes() {
  char * z = malloc(100000);
}

int main() {
  for (int i = 0; i < 10000000; i+=1) {
    zeroes();
  }
  return 0;
}
```

What will this do?

```c
#include <stdio.h>
#include <stdlib.h>

void zeroes() {
  char * z = malloc(100000);
}

int main() {
  for (int i = 0; i < 10000000; i+=1) {
    zeroes();
  }
  return 0;
}
```

Let's inspect
with **top**

Recap:

- **`void* malloc(size_t size);`**
  - Allocates **size** bytes and returns the pointer to it, or NULL if failed to alloc
- **`void* calloc(size_t n_items, size_t size);`**
  - Allocates **(n_items* size)** bytes and returns the pointer to it, or NULL if failed to alloc
- **`void free(void * ptr);`**
  - Frees the memory pointer to by **ptr** so that your program can no longer rely on having access to that memory

Implement the function

- Write a function named **dynamic_strcat**
- Takes two params, **char***s, pointing to two C strings
- Function allocates memory that fits both strings, contacts them, and returns the pointer

Implement the function

- Write a function named **dynamic_split**
- Takes one params, **char*** pointing to a C string
- returns a **char****, a pointer to a sequence of pointers to the words from the parameter split on a space.

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```

What does this code accomplish?

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```

What is WRONG with how this is written?

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```

Input:

abcdefghijk
abcdefghijklmnop
abcdefghijklmnopqrs
abcdefghijklmnopqrstuv

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```

Stack

main's caller's stack frame

main's stack frame
char* longest_line 64 bits   0x???

. . .

Heap

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;          <---
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```

Stack

main's caller's stack frame

main's stack frame
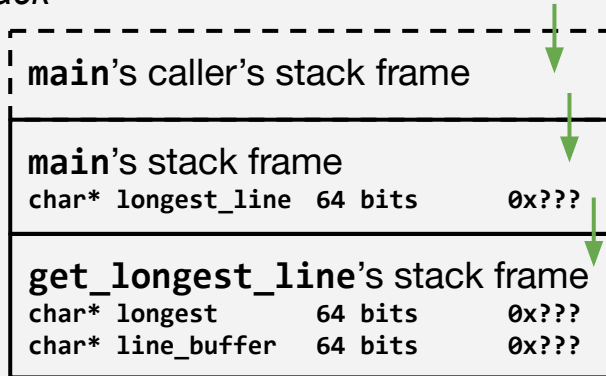char* longest_line   64 bits   0x???

get_longest_line's stack frame
char* longest   64 bits   0x???
char* line_buffer   64 bits   0x???

....

Heap

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```


Stack

main's caller's stack frame

main's stack frame
char* longest_line   64 bits   0x???

get_longest_line's stack frame
char* longest   64 bits   0x???
char* line_buffer   64 bits   0x???

....

Heap
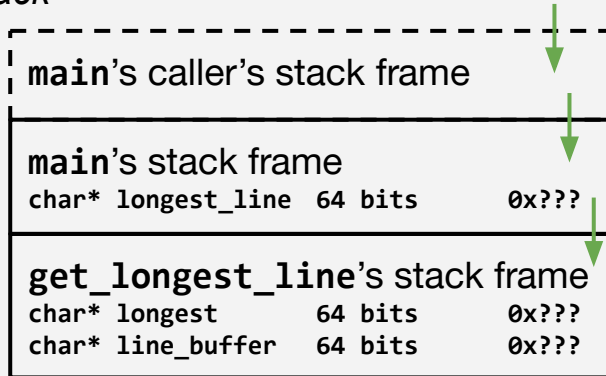
0x00..1981   250 bytes of memory

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```
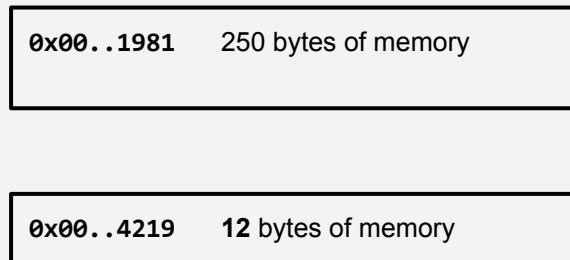
Stack

main's caller's stack frame

main's stack frame
char* longest_line   64 bits   0x???

get_longest_line's stack frame
char* longest   64 bits   0x???
char* line_buffer   64 bits   0x???

....

Heap

0x00..1981   250 bytes of memory

0x00..4219   **12** bytes of memory

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```
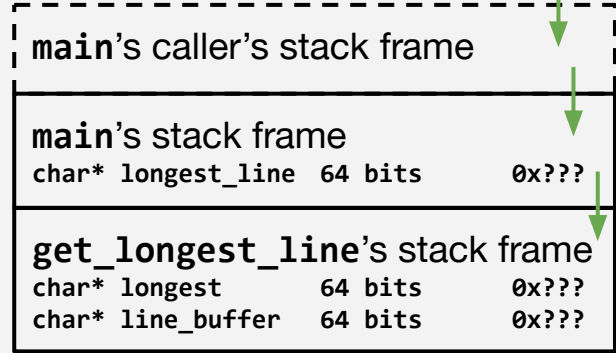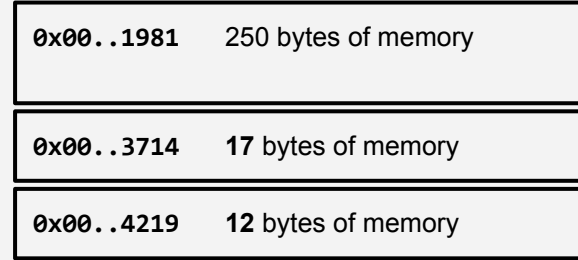
Stack

main's caller's stack frame

main's stack frame
char* longest_line   64 bits   0x???

get_longest_line's stack frame
char* longest   64 bits   0x???
char* line_buffer   64 bits   0x???

....

Heap

0x00..1981   250 bytes of memory

0x00..3714   **17** bytes of memory
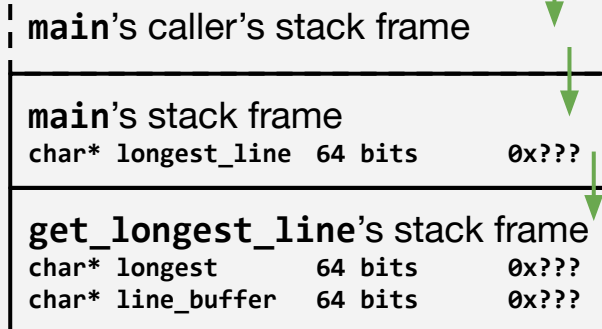
0x00..4219   **12** bytes of memory

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```
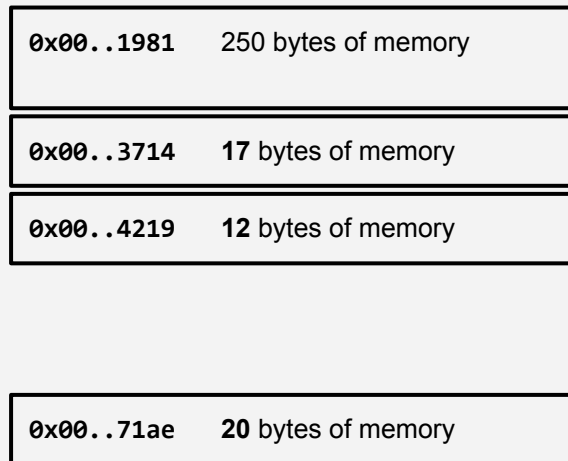
Stack

main's caller's stack frame

main's stack frame
char* longest_line   64 bits   0x???

get_longest_line's stack frame
char* longest   64 bits   0x???
char* line_buffer   64 bits   0x???

....

Heap

0x00..1981   250 bytes of memory

0x00..3714   **17** bytes of memory

0x00..4219   **12** bytes of memory

0x00..71ae   **20** bytes of memory

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```

Stack

main's caller's stack frame

main's stack frame
char* longest_line   64 bits   0x???

get_longest_line's stack frame
char* longest   64 bits   0x???
char* line_buffer   64 bits   0x???

....

Heap

0x00..1981   250 bytes of memory

0x00..3714   **17** bytes of memory

0x00..4219   **12** bytes of memory

0x00..51c4   **23** bytes of memory
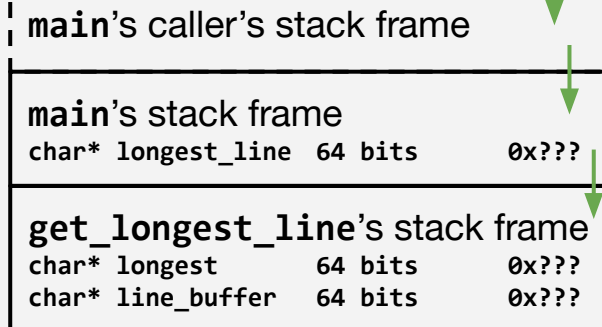
0x00..71ae   **20** bytes of memory

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```
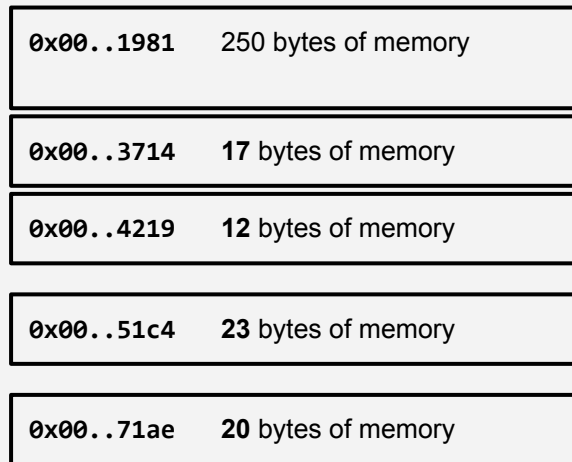
Stack

main's caller's stack frame

main's stack frame
char* longest_line   64 bits   0x???

get_longest_line's stack frame
char* longest   64 bits   0x???
char* line_buffer   64 bits   0x???

....

Heap

0x00..1981   250 bytes of memory

0x00..3714   **17** bytes of memory

0x00..4219   **12** bytes of memory

0x00..51c4   **23** bytes of memory

0x00..71ae   **20** bytes of memory

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```

Stack

main's caller's stack frame

main's stack frame
char* longest_line   64 bits   0x???

get_longest_line's stack frame
char* longest   64 bits   0x???
char* line_buffer   64 bits   0x???

....

Heap

0x00..1981   250 bytes of memory

0x00..3714   17 bytes of memory

0x00..4219   12 bytes of memory

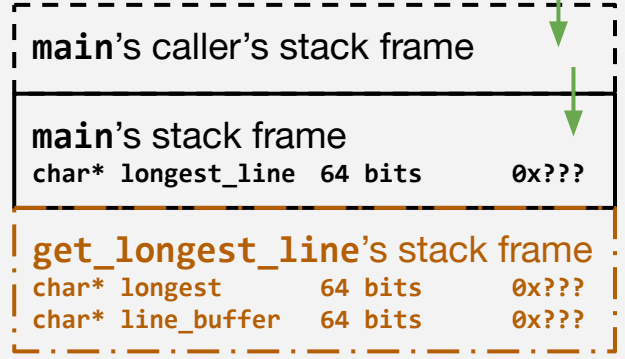0x00..51c4   23 bytes of memory

0x00..71ae   20 bytes of memory

Memory freed

```c
#define LARGE 250

char* get_longest_line() {
  char* longest = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest == NULL || length > strlen(longest)) {
      longest = malloc(length+1);
      longest[length] = '\0';
      strncpy(longest, line_buffer, length);
    }
  }
  return longest;
}

int main() {
  char* longest_line = get_longest_line();
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(longest_line);
  return 0;
}
```

Stack

main's caller's stack frame

main's stack frame
`char* longest_line   64 bits      0x???`

get_longest_line's stack frame
`char* longest        64 bits      0x???`
`char* line_buffer    64 bits      0x???`

. . . .

Heap

| `0x00..1981` | 250 bytes of memory |

| `0x00..3714` | **17** bytes of memory |

| `0x00..4219` | **12** bytes of memory |

| `0x00..51c4` | **23** bytes of memory |

| `0x00..71ae` | **20** bytes of memory |

What about these though?