CSc 352

Malloc, Free, and the Heap

Benjamin Dicken

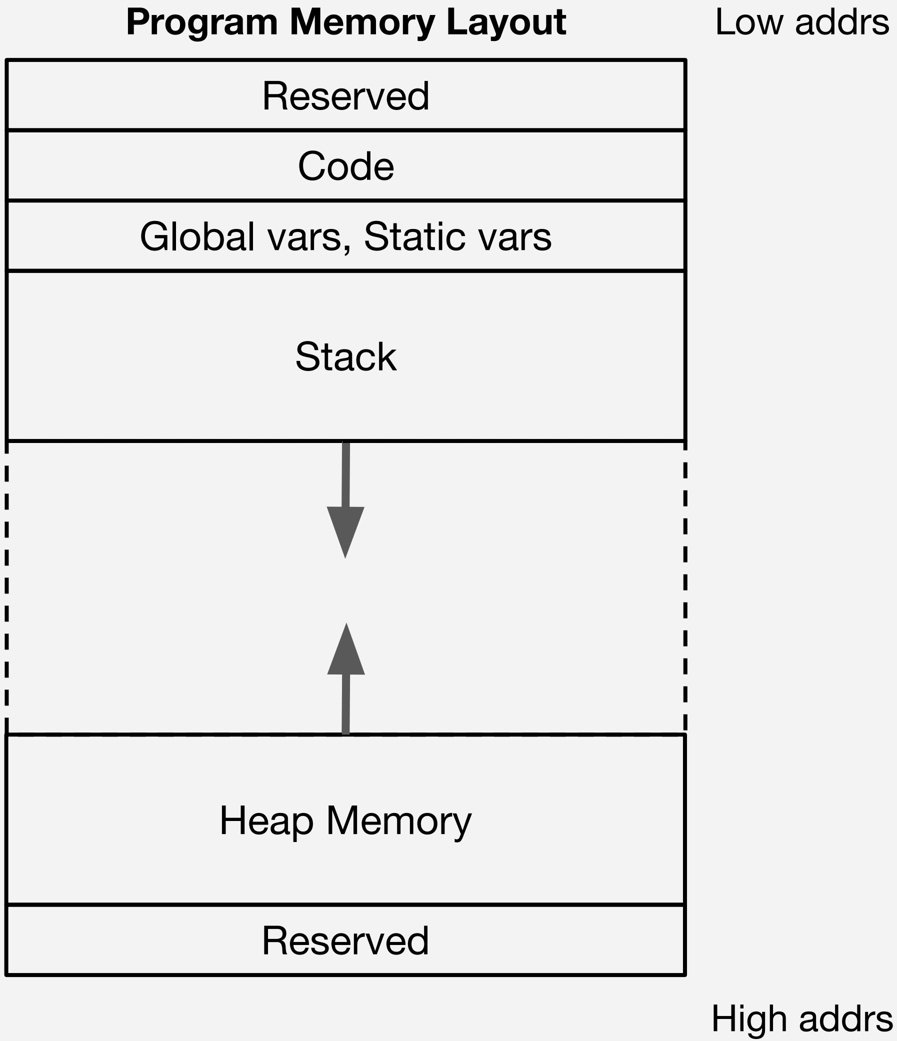
Announcements

- Exam 1 grades posted
    - Exam 1 question 6
- The next two PAs

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

Low addrs

Reserved

Code

Global vars, Static vars

Stack

Heap Memory

Reserved

High addrs

Activity

What will it print? (probably?)

```
#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;
}
```

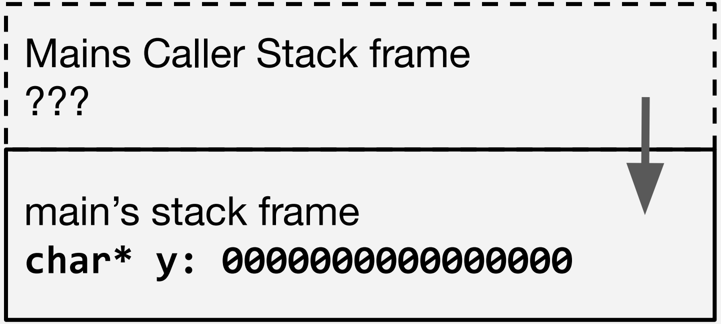
Stack Diagram

```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;
}
```



Mains Caller Stack frame
???

main’s stack frame
`char* y: 0000000000000000`

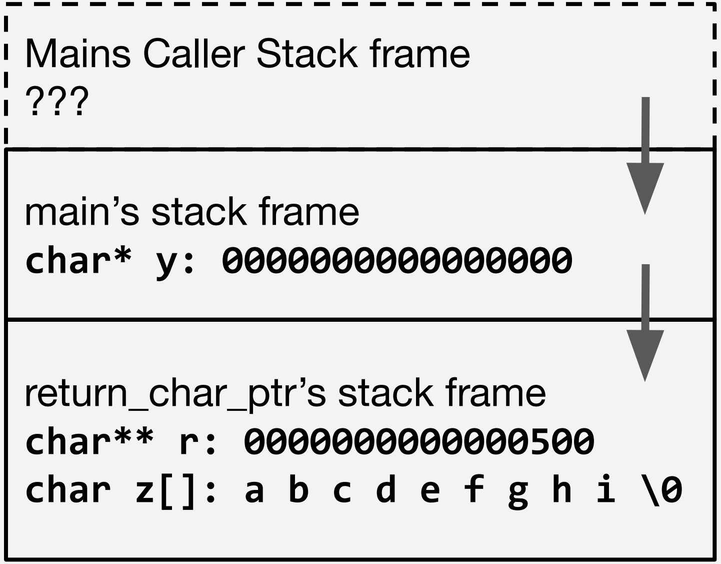
Stack Diagram

```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;
}
```



Mains Caller Stack frame
???

main’s stack frame
char* y: 0000000000000000

return_char_ptr’s stack frame
char r: 0000000000000500**
char z[]: a b c d e f g h i \0

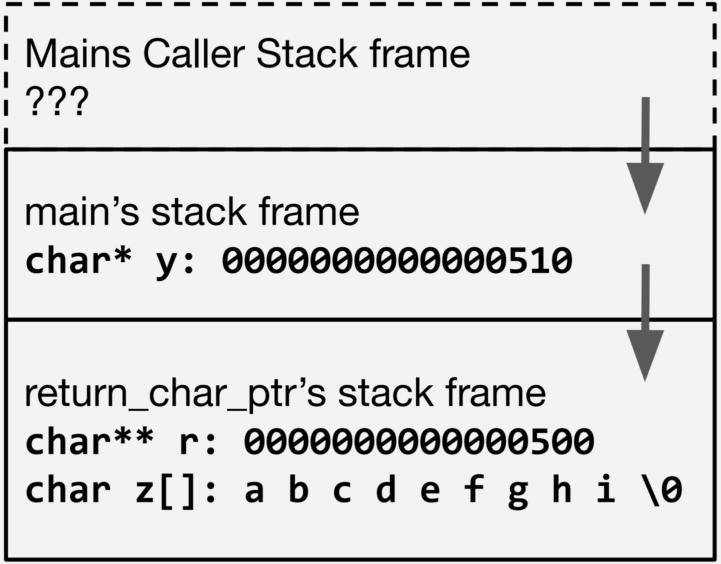
Stack Diagram

```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;
}
```



Mains Caller Stack frame
???

main’s stack frame
char* y: 0000000000000510

return_char_ptr’s stack frame
char r: 0000000000000500**
char z[]: a b c d e f g h i \0

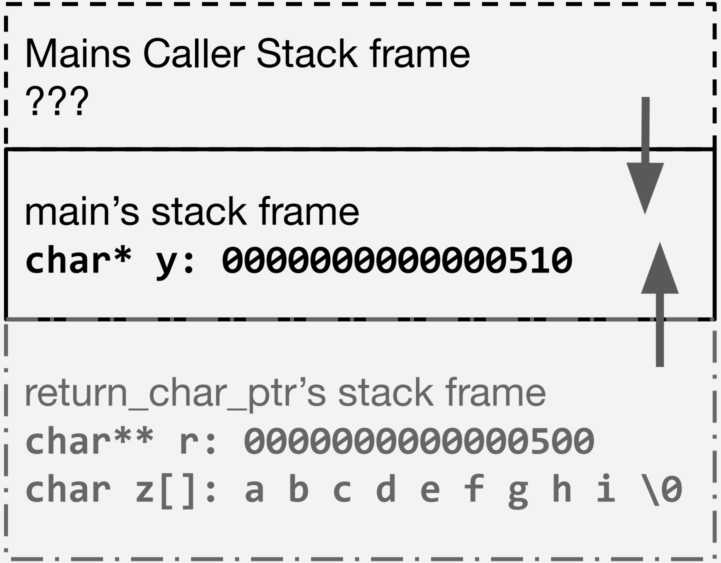
Stack Diagram

```
#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;
}
```



Mains Caller Stack frame

???

main's stack frame

`char* y: 0000000000000510`

return_char_ptr's stack frame

`char** r: 0000000000000500`

`char z[]: a b c d e f g h i \0`

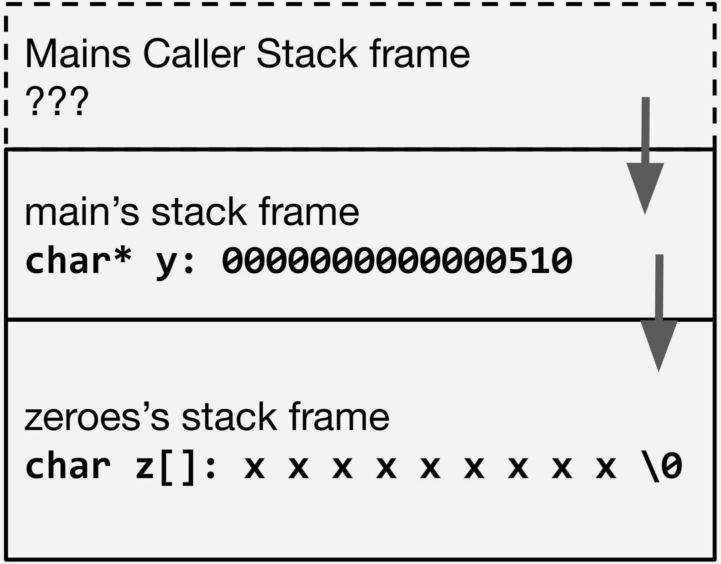
Stack Diagram

```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;
}
```



Mains Caller Stack frame
???

main’s stack frame
char* y: 0000000000000510

zeroes’s stack frame
char z[]: x x x x x x x x x \0

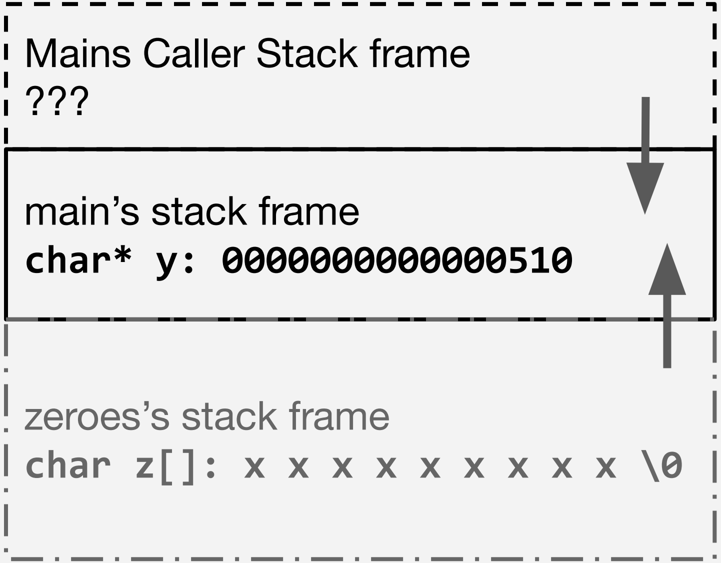
Stack Diagram

```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;
}
```



Mains Caller Stack frame
???

main’s stack frame
char* y: 0000000000000510

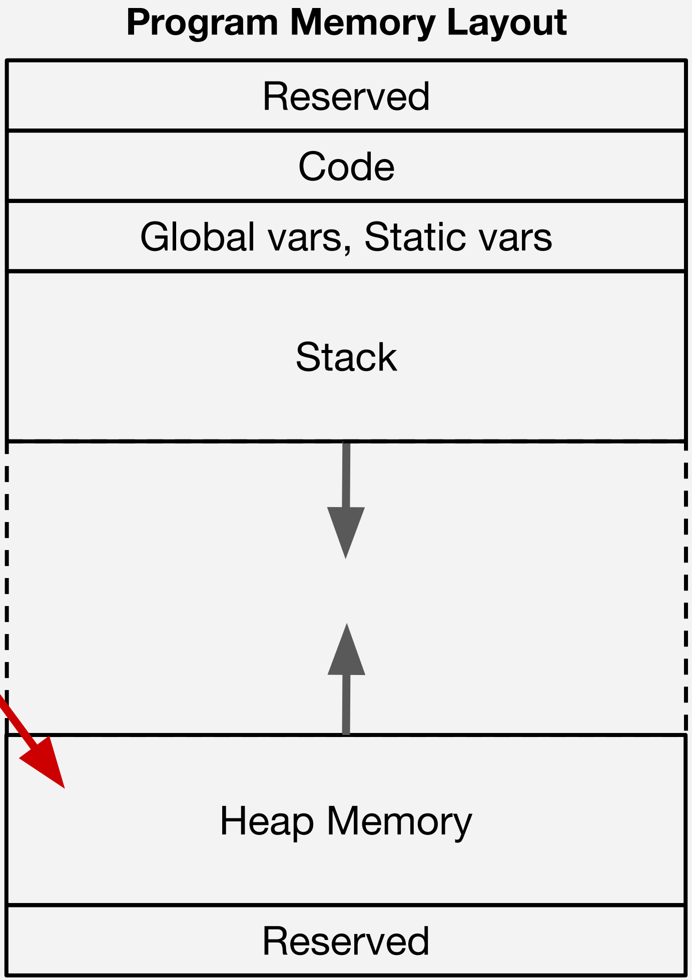
zeroes’s stack frame
char z[]: x x x x x x x x x \0

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

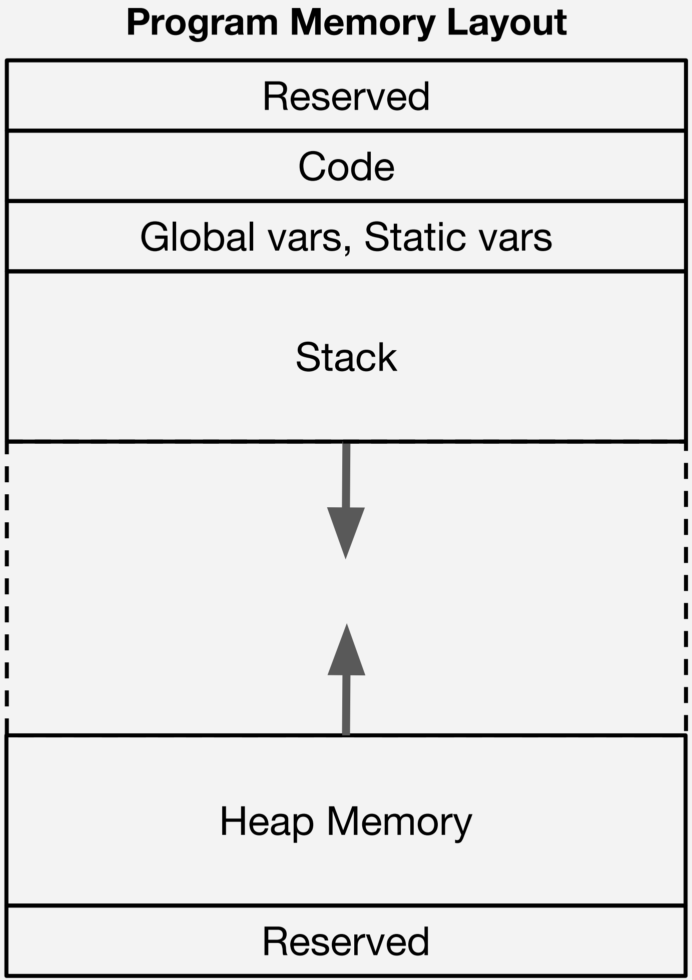
Malloc

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?



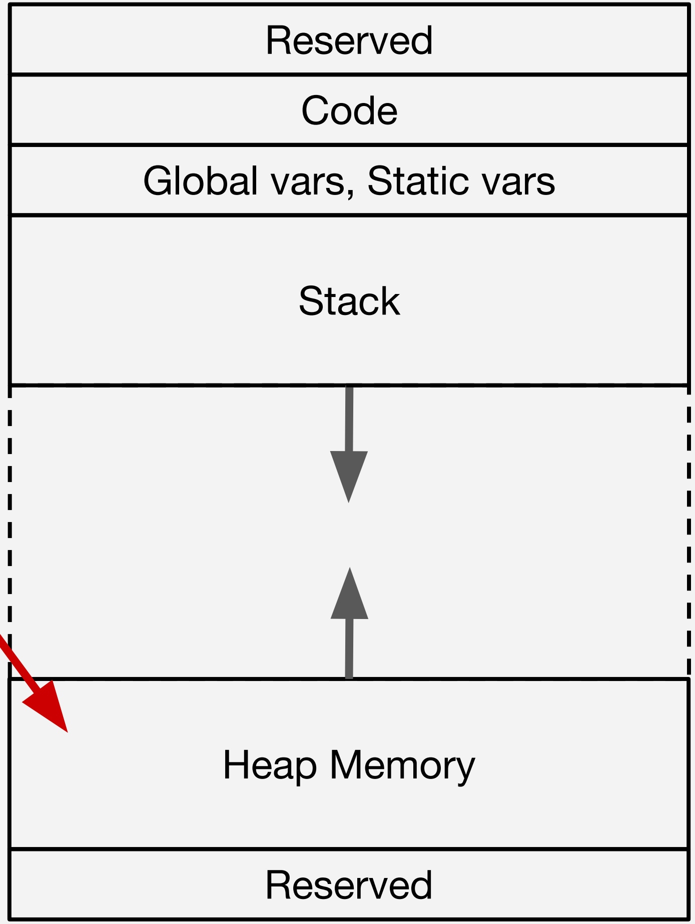
Free

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



Activity

What will it print?

```
#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';
  }
  z[9] = ‘\0’;
  *r = z;
}
```

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

Activity

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?

Activity

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;
}
```

Activity

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**

calloc

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

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

Activity

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

More than one value?

- In C, you can return one value from a function (pointer, int, char, etc)

- What if you want to return more than one value?

- For example, a function that:
    - splits a C string exactly in half, and returns both halves
    - Takes a physical address, returns a lat and long value
    - . . . .

Out-Parameters

- An out-parameter is a way of getting a value “out” of a function call without relying on a **return** statement
- If you are calling function Y from function X, you can send Y the address of a local variable from X to store a value into
- This gives the ability to “return” multiple things!

Out-Parameters

- An out-parameter is a way of getting a value “out” of a function call without relying on a **return** statement
- If you are calling function Y from function X, you can send Y the address of a local variable from X to store a value into
- This gives the ability to “return” multiple things!

```c
void split_in_half(char* to_split, char** half_one, char** half_two) {
  int half = (int) (strlen(to_split) / 2);
  *half_one = calloc(1, half+1);
  *half_two = calloc(1, half+1);
  strncpy(*half_one, to_split, half);
  strncpy(*half_two, (to_split+half), half);
}

int main() {
  char alphabet[27] = "abcdefghijklmnopqrstuvwxyz";
  char * h1;
  char * h2;
  split_in_half(alphabet, &h1, &h2);
  printf("alphabet: %s\n", alphabet);
  printf("h1: %s\n", h1);
  printf("h2: %s\n", h2);
  return 0;
}
```