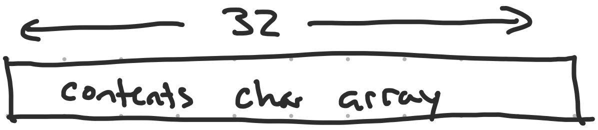


```
typedef struct Len32String {
  char contents[32];
} Len32String;

Len32String return_struct() {
  Len32String str_struct = { "some string content" };
  return str_struct;
}
```

struct instances are copied on arg/return

```
char* return_array() {
  char str[] = "some string content";
  return str;
}
```

returning address of stack-allocated

```
void example2() {
  char* from_array1 = return_array();
  printf("from_array1:\t%p\n", from_array1);
  char* from_array2 = return_array();
  printf("from_array2:\t%p\n", from_array2);

  Len32String from_struct1 = return_struct();
  printf("from_struct1:\t%p\n", from_struct1.contents);
  Len32String from_struct2 = return_struct();
  printf("from_struct2:\t%p\n", from_struct2.contents);
}
```

Q1: Do these print the same address? Yes

Q2: Do these print the same address? No

How badly did Joe lie?

"We cannot return arrays from functions in C"

Q3: Fill in the blanks/fix to print with no type errors.

```
typedef struct {
  double x, y;
} Point;

void print_point(Point* p) {
  printf("(%~~d~~f, %~~d~~f)\n", p->x, p->y);
}

int main() {
  Point p = { 4.0, 5.0 };

  print_point(&p);

  printf("(%~~d~~f, %~~d~~f)\n", p.x, p.y);
}
```

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

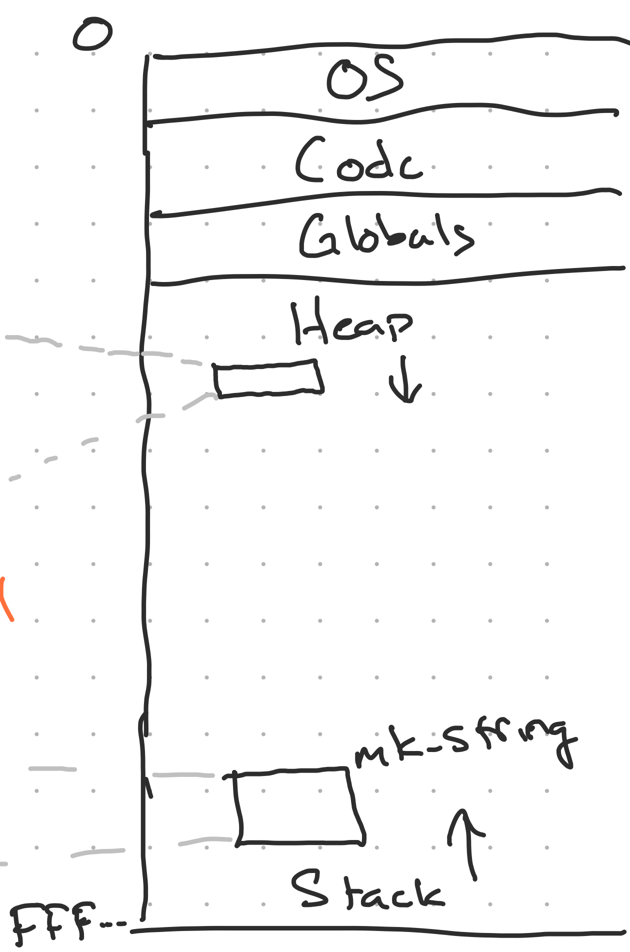
just int

an address with unknown underlying type (pointer)

we can assign void* to any pointer type

malloc: Sets aside size bytes on the heap and returns a pointer to the start address of that space (can return NULL if not enough space)

```
char* mk_string() {
    char* s = malloc(8);
    strcpy(s, "hello!");
    return s;
}
```



Heap	0x16...40	h	e	l	l	o	!	\0	■

The malloc'd memory persists across function calls, etc until the program exits*

Stack s 0x16...40

```
char* strdup(char* s) {   //return a heap-allocated
                          copy of s
    char* new_s = malloc(strlen(s) + 1);
    strcpy(new_s, s);
    return new_s;
}
```

```
> python3
Python 3.13.7
>>> x = "hello"
>>> y = " class"
>>> x + y
'hello class'
>>> x
'hello'
>>> y
' class'
>>> fruit = "apple,banana,cranberry"
>>> fruit.split(",")
['apple', 'banana', 'cranberry']
>>> len(x)
5
```

```
> jshell
|  Welcome to JShell -- Version 24.0.1
|  For an introduction type: /help intro

jshell> String x = "hello", y = " class";
x ==> "hello"
y ==> " class"

jshell> x + y
$3 ==> "hello class"

jshell> String fruit = "apple,banana,cranberry";
fruit ==> "apple,banana,cranberry"

jshell> fruit.split(",")
$5 ==> String[3] { "apple", "banana", "cranberry" }
```

https://github.com/python/cpython/blob/main/Include/cpython/unicodeobject.h#L166

```
typedef struct {
    PyASCIIObject _base;
    Py_ssize_t utf8_length;     /* Number of bytes in utf8, excluding the
                                 * terminating \0. */
    char *utf8;                 /* UTF-8 representation (null-terminated) */
} PyCompactUnicodeObject;
```

this field usually is an address of a heap-allocated string!

```
typedef struct {
    uint64_t len;
    char* utf8;
} Str;
```

```
Str concat(Str s1, Str s2) {
    uint64_t new_len = s1.len + s2.len;
    char* new_data = malloc(new_len + 1);
    strcpy(new_data, s1.utf8);
    strcat(new_data, s2.utf8);
    Str to_return = { new_len, new_data };
    return to_return;
}
```

```
def concat_all(strs):
    s = ""
    for str in strs:
        s = s + str
    return s
```

```
Str concat_all( Str strs[], int count) {
    Str s = new_str("");
    for(int i=0; i<count; i+=1) {
        char* old_s = s.utf8;
        s= concat(s, strs[i]);
        free(old_s);
    }
    return s;
}
```

free(void* p) tells malloc the space for this pointer can be re-used now

strs

"apple" "banana" "cranberry" "donut"

What calls to malloc happen?

malloc(6) for "" + "apple"

malloc(12) for "apple" + "banana"

malloc(21) for "applebanana" + "cranberry"

malloc(26) for "a... b... c" + "donut"

allocated and will never be used again!

Memory Leak

returned

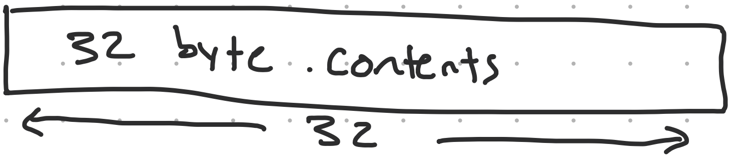
```c
typedef struct Len32String {
  char contents[32];
} Len32String;

Len32String return_struct() {
  Len32String str_struct = { "some string content" };
  return str_struct;
}

char* return_array() {
  char str[] = "some string content";
  return str;
}

void example2() {
  char* from_array1 = return_array();
  printf("from_array1:\t%p\n", from_array1);
  char* from_array2 = return_array();
  printf("from_array2:\t%p\n", from_array2);

  Len32String from_struct1 = return_struct();
  printf("from_struct1:\t%p\n", from_struct1.contents);
  Len32String from_struct2 = return_struct();
  printf("from_struct2:\t%p\n", from_struct2.contents);
}
```



struct instances are copied on return/arg

Returning address of stack-allocated array

Q1: Could these print the same address? Yes

Q2: Could these print the same address? No

```c
typedef struct {
  double x, y;
} Point;

void print_point(Point* p) {
  printf("(%~~d~~f, %~~d~~f)\n", p->x, p->y);
}

int main() {
  Point p = { 4.0, 5.0 };

  print_point(&p);

  printf("(%~~d~~f, %~~d~~f)\n", p.x, p.y);
}
```

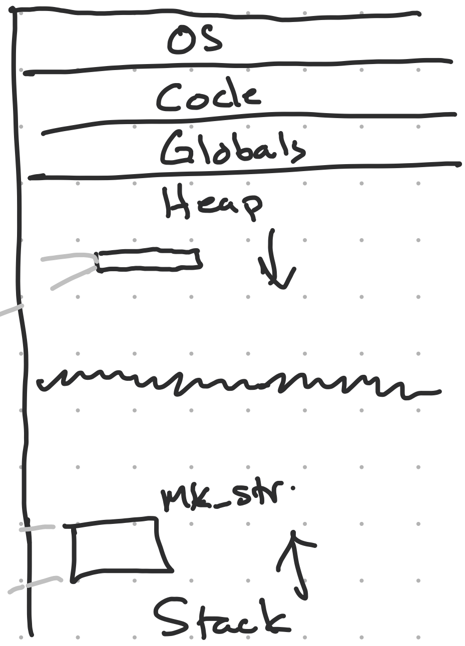
Q3: Fill in blanks/fix to print w/no type errors just int64_t

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

└ void* is the type of pointers with unknown underlying type. It is assignable to any pointer type.

malloc sets aside size bytes on the heap and returns the address of the start of that space (NULL if out of space)

```
char* mk_str() {
    char* s = malloc(8);
    strcpy(s, "hello!");
    return s;
}
```



Heap	0x16..40	h	e	l	l	o	!	\0	▨

Heap data will persist and not be overwritten ever!*

Stack	s	0x16...40

```
char* strdup(char* s) {    // return a heap-allocated copy
    char* new_s = malloc(strlen(s) +1);
    strcpy( new_s , s )
    return new_s;
}
```

```
> python3
Python 3.13.7
>>> x = "hello"
>>> y = " class"
>>> x + y
'hello class'
>>> x
'hello'
>>> y
' class'
>>> fruit = "apple,banana,cranberry"
>>> fruit.split(",")
['apple', 'banana', 'cranberry']
>>> len(x)
 5
```

```
> jshell
|  Welcome to JShell -- Version 24.0.1
|  For an introduction type: /help intro

jshell> String x = "hello", y = " class";
x ==> "hello"
y ==> " class"

jshell> x + y
$3 ==> "hello class"

jshell> String fruit = "apple,banana,cranberry";
fruit ==> "apple,banana,cranberry"

jshell> fruit.split(",")
$5 ==> String[3] { "apple", "banana", "cranberry" }
```

https://github.com/python/cpython/blob/main/Include/cpython/unicodeobject.h#L166

```
typedef struct {
    PyASCIIObject _base;
    Py_ssize_t utf8_length;     /* Number of bytes in utf8, excluding the
                                 * terminating \0. */
    char *utf8;                 /* UTF-8 representation (null-terminated) */
} PyCompactUnicodeObject;
```

these C strings are heap-allocated

```
typedef struct {
    int64_t len;
    char* utf8;
} Str;
```

s1 + s2 in Python

```
Str concat(Str s1, Str s2) {
    int64_t new_len = s1.len + s2.len;
    char* new_utf8 = malloc( new_len + 1 );
    Str to_return = { new_len , new_utf8 };
    strcpy(new_utf8, s1.utf8);
    strcat(new_utf8, s2.utf8);     // strcpy(&new_utf8[s1.len],
                                   //        s2.utf8);
    return to_return;
}

Str new_str(char* s) {
    char* duped = strdup(s);
    Str str = { strlen(s), duped };
    return str;
}
```

this is what a string literal in Python is

```
Str concat_all(Str strs[], int count) {
    Str s = new_str("");
    for(int i=0; i<count; i+=1) {
        char * old_utf8 = s.utf8;
        s = concat(s, strs[i]);
        free(old-utf8);
    }
    return s;
}
```

```
def concat_all(strs):
    s = ""
    for str in strs:
        s = s + str
    return s
```

free(void* p)

Takes a pointer returned by malloc and tells malloc it can reuse that space

"apple" "banana" "cran"

How many/which mallocs happen?

malloc(1)	for " "
malloc(6)	for " " + "apple"
malloc(12)	for "apple" + "banana"
malloc(16)	for "a...b..." + "cran"

Memory Leak

will never be used again... but is on the heap

returned from concat_all (utf8 field)