```
int _main(int argc, char** strs) {
  char str[] = "abcde";

  printf("%s\n", str);
  printf("%c %d\n", str[0], str[0]);
  printf("%c %d\n", str[5], str[5]);
  printf("%d\n", sizeof(str));
}
```

Stack-allocated array

strs

main			
a 97	b 98	c 99	d 100
e 101	\0 0		

1. strings are char arrays
2. strings are terminated with \0 \0

```
int main(int argc, char** strs) {
  char* str = calloc(56, sizeof(char));
  str[0] = 'a'; str[1] = 'b';
  str[2] = 'c'; str[3] = 'd';
  str[4] = 'e'; str[5] = '\0';

  printf("%s\n", str);
  printf("%c %d\n", str[0], str[0]);
  printf("%c %d\n", str[5], str[5]);
  printf("%d\n", sizeof(str));



}
```

strs

main
@P

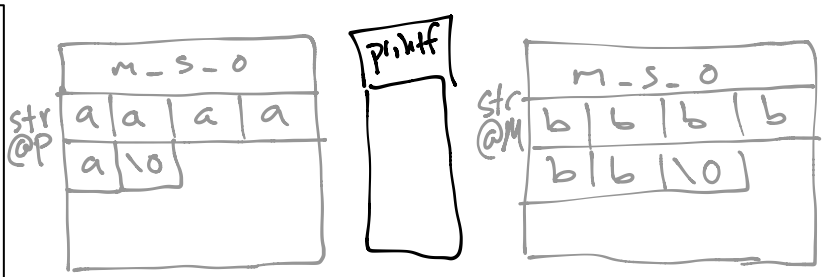
@P

a	b	c	d
e	\0		

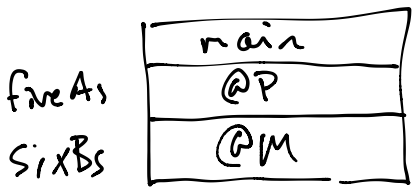
heap-allocated array

```
char* make_string_of(char c, int count) {
  char str[count + 1];
  int index = 0;
  for(index = 0; index < count; index += 1) {
    str[index] = c;
  }
  str[count] = '\0';
  return str;
}
```

DO NOT return stack arrays

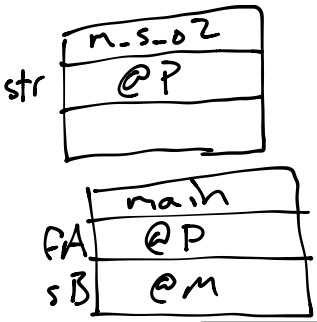



```
int main(int argc, char** strs) {
  char* fiveAs = make_string_of('a', 5);
  printf("Five As: %s\n", fiveAs);
  char* sixBs = make_string_of('b', 6);
  printf("Six Bs: %s\n", sixBs);
}
```



UNDEFINED BEHAVIOR
- read/write out of bounds
- use "old" stack data

```
char* make_string_of2(char c, int count) {
  char* str = calloc(count + 1, sizeof(char));
  int index = 0;
  for(index = 0; index < count; index += 1) {
    str[index] = c;
  }
  str[count] = '\0';
  return str;
}
```




```
int main(int argc, char** strs) {
  char* fiveAs = make_string_of2('a', 5);
  printf("Five As: %s\n", fiveAs);
  char* sixBs = make_string_of2('b', 6);
  printf("Six Bs: %s\n", sixBs);
  free(fiveAs);
  free(sixBs);
}
```

calloc called? A: 1 B: 2 C: 3

free(pointer)

Takes a pointer (either interpretation – a struct reference or an array!) and tells calloc/malloc that the calloc()'ed space it refers to can be used again in the future. The pointer must not be used again after freeing it. free() should be called once for each time calloc() is used. Failing to free is called a **memory leak**.

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

typedef struct AList {
  int* contents;
  int size;
  int capacity;
} AList;

AList* make_alist(int start_capacity) {









}

void expandCapacity(AList* alist) {







}

void add(AList* alist, int element) {







}

int get(AList* alist, int index) {







}

void print_alist(AList* alist) {
  int i = 0;
  for(i = 0; i < alist->size; i += 1) {
    printf("%d, ", alist->contents[i]);
  }
}


int main(int argc, char** args) {
  AList* a = make_alist(4);
  add(a, 5);
  add(a, 3);
  add(a, 1);
  printf("%d\n", get(a, 0));
  printf("%d\n", get(a, 1));
  printf("%d\n", get(a, 2));

  print_alist(a);









}
```

How many times is calloc() called in the make_string2 example?

A: 1 B: 2

How much space is allocated for characters with calloc() in total in the make_string2 example?

A: 11 chars B: 12 chars C: 13 chars D: Something else

A

```
AList* make_alist(int start_capacity) {
  AList* alist = calloc(start_capacity, sizeof(AList));
  int* contents = calloc(1, sizeof(int));
  alist->contents = contents;
  alist->size = 0;
  alist->capacity = start_capacity;
  return alist;
}
```

B

```
AList* make_alist(int start_capacity) {
  AList* alist = calloc(1, sizeof(AList*));
  int* contents = calloc(start_capacity, sizeof(int*));
  alist->contents = contents;
  alist->size = 0;
  alist->capacity = start_capacity;
  return alist;
}
```

C

```
AList* make_alist(int start_capacity) {
  AList** alist = calloc(1, sizeof(AList*));
  int* contents = calloc(start_capacity, sizeof(int));
  alist->contents = contents;
  alist->size = 0;
  alist->capacity = start_capacity;
  return alist;
}
```

D

```
AList* make_alist(int start_capacity) {
  AList* alist = calloc(1, sizeof(AList));
  int* contents = calloc(start_capacity, sizeof(int));
  alist->contents = contents;
  alist->size = 0;
  alist->capacity = start_capacity;
  return alist;
}
```

```
void add(AList* alist, int element) {
  if(alist.size >= alist.capacity) { expandCapacity(alist); }
  alist[alist->size] = element;
  alist->size += 1;
}
```

```
void add(AList* alist, int element) {
  if(this.size >= this.capacity) { expandCapacity(); }
  alist[alist->size] = element;
  alist->size += 1;
}
```

```
void add(AList* alist, int element) {
  if(alist->size >= alist->capacity) { expandCapacity(alist); }
  alist->contents[alist->size] = element;
  alist->size += 1;
}
```

```
void add(AList* alist, int element) {
  if(alist->size >= alist->capacity) { expandCapacity(alist); }
  alist.contents[alist->size] = element;
  alist->size += 1;
}
```