```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h> // for malloc

#define READLINE_LENGTH 4096

typedef struct String {
  int len;
  char* contents;
} String;

String readline() {
  char input[READLINE_LENGTH];
  char* result = fgets(input, READLINE_LENGTH, stdin);
  if(result == NULL) {
    String s = { -1, NULL };
    return s;
  }
  int length = strlen(input);
  char* contents = malloc(length + 1);
  strcpy(contents, input);
  String s = { strlen(input), contents };
  return s;
}



int main(int argc, char** argv) {
  char* to_find = argv[1];
  while(1) {
    String s = readline();
    if(s.contents == NULL) { break; }
    printf("%p (%d)\n", s.contents, s.len);
    if(strstr(s.contents, to_find) != NULL) {
      printf("%s\n", s.contents);
    }

    free( s.contents );

  }
}
```

output before adding free()

```
$ gcc -Wall -g search.c -o search
$ ./search hi
hihello
0x58eafd1546b0 (8)
hihello

hi
0x58eafd154ae0 (3)
hi

hey!
0x58eafd154b00 (5)
$ valgrind ./search hi
==1198410== Memcheck, a memory error detector
==1198410== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et
    al.
==1198410== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright
    info
==1198410== Command: ./search hi
==1198410==
hihello
0x4a8b480 (8)
hihello

hey!
0x4a8b910 (5)
more hi
0x4a8b960 (8)
more hi

==1198410==
==1198410== HEAP SUMMARY:
==1198410==     in use at exit: 24 bytes in 3 blocks
==1198410==   total heap usage: 5 allocs, 2 frees, 2,072 bytes allocated
==1198410==
==1198410== LEAK SUMMARY:
==1198410==    definitely lost: 24 bytes in 3 blocks
==1198410==    indirectly lost: 0 bytes in 0 blocks
==1198410==      possibly lost: 0 bytes in 0 blocks
==1198410==    still reachable: 0 bytes in 0 blocks
==1198410==         suppressed: 0 bytes in 0 blocks
==1198410== Rerun with --leak-check=full to see details of leaked memory
==1198410==
==1198410== For lists of detected and suppressed errors, rerun with: -s
==1198410== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

addresses in increasing order

hi hello\n\0 (9)

hey\n\0 (6)

more hi\n\0 (9)

s { 8, • }

`void* malloc(size_t size)`

`void free(void* ptr)` - marks the memory for ptr as available for malloc again

↑ pointer of any type

(must be called on a ptr from malloc)

```
  $ vim search.c # added call to free!
$ gcc -g -Wall search.c -o search # make a fix!
$ ./search hi
hihello
0x5bba414976b0 (8)
hihello

hi
0x5bba414976b0 (3)
hi

hey!
0x5bba414976b0 (5)
$ valgrind ./search hi
==1199633== Memcheck, a memory error detector
==1199633== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et
    al.
==1199633== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright
    info
==1199633== Command: ./search hi
==1199633==
hihello
0x4a8b480 (8)
hihello

hi
0x4a8b910 (3)
hi

hey!
0x4a8b960 (5)
==1199633==
==1199633== HEAP SUMMARY:
==1199633==     in use at exit: 0 bytes in 0 blocks
==1199633==   total heap usage: 5 allocs, 5 frees, 2,067 bytes allocated
==1199633==
==1199633== All heap blocks were freed -- no leaks are possible
==1199633==
==1199633== For lists of detected and suppressed errors, rerun with: -s
==1199633== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

• bitcat0a\n\0
hey!\n\0

— still different addresses!

happy valgrind output!

free(ptr) constraints:

- ptr must have been malloc'd
- ptr must not be NULL
- ptr must not have been free()'d before the free()
- ptr must not be used after the free()

Bad free

Double Free

Use-after-free

UNDEFINED BEHAVIOR

the process can do anything it is capable of

It is inhumanly possible to use free() correctly in large programs

Rust Java Python
Automated Memory Mgmt