GCXX

GC: what is needed?

- Allocator
- Mark
- Sweep

Allocator

The whole API:

```
std::map<void*, size_t> allocated;
void* heap;
static constexpr size_t MIN_SIZE =
  sizeof(void*) * 2;

[[nodiscard]] void* allocate(size_t size);
void deallocate(void* p);
```

Mark

Find all reachable objects.
Search begins with GC roots, i.e.

- Globals
- Registers
- Stack

GC roots

— [Logic – Challenging 12] I know how to find the roots.

GC roots

— [Logic – Challenging 12] I know how to find the roots.
LOGIC [Challenging: Failure] — You didn't think that scanning the globals and the stack would be so easy, right?

Globals and stack

Turns out, finding the location of static memory is quite hard. There's no easy way to do it, only parsing the current ELF file and asking all of the loaded dynamic libraries.

The solution

AUTHORITY [Trivial: Success] — **Who cares about globals after all? Check only the registers and the stack.**

The solution

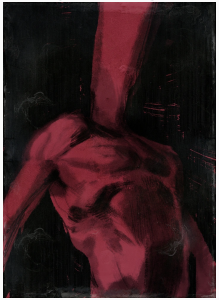
AUTHORITY [Trivial: Success] — Who cares about globals after all? Check only the registers and the stack.
LOGIC [Easy: Success] — In GC constructor save the current rsp value as the stack base and then iterate over the whole stack, finding the roots.

Mark phase

Mark needs two crutial things: looking into the memory behind found pointers and registering them as reachable.

Mark phase

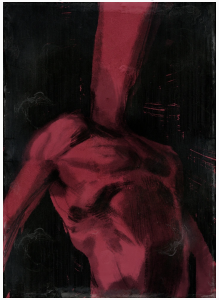
Mark needs two crutial things: looking into the memory behind found pointers and registering them as reachable.



— How do I know the size of the object behind a pointer?

Mark phase

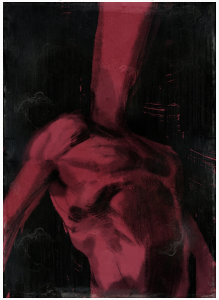
Mark needs two crutial things: looking into the memory behind found pointers and registering them as reachable.



— How do I know the size of the object behind a pointer?
PHYSICAL INSTRUMENT [Medium: Success] — The %!@& do you need memory for? Look at those bytes. You *must* use them. You *must* rule them. Map is the way.

Mark phase

Mark needs two crutial things: looking into the memory behind found pointers and registering them as reachable.



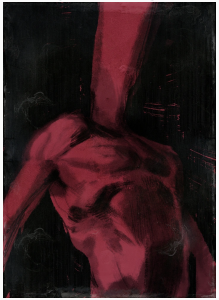
— How do I know the size of the object behind a pointer?

PHYSICAL INSTRUMENT [Medium: Success] — The %!@& do you need memory for? Look at those bytes. You *must* use them. You *must* rule them. Map is the way.

— Sure thing. But how do we mark the objects that are reachable?

Mark phase

Mark needs two crutial things: looking into the memory behind found pointers and registering them as reachable.



— How do I know the size of the object behind a pointer?
PHYSICAL INSTRUMENT [Medium: Success] — The %!@& do you need memory for? Look at those bytes. You *must* use them. You *must* rule them. Map is the way.
— Sure thing. But how do we mark the objects that are reachable?
PHYSICAL INSTRUMENT [Heroic: Success] — I'll say it again. The %!@& do you need memory for? Write a bitset for the whole heap.

Sweep phase

Three lines:

```
for (auto&& [ptr, _] : allocator.allocated) {
  if (!reachable[...]) {
    allocator.deallocate(ptr);
  }
}
```

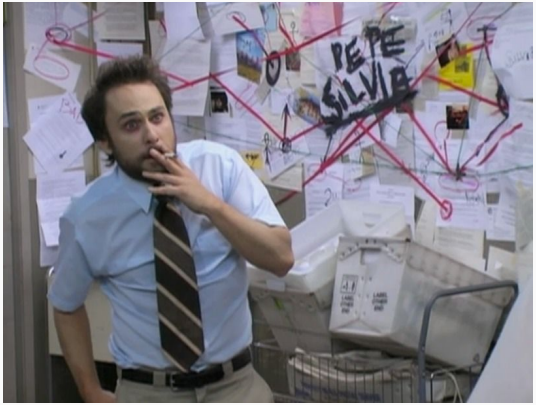
Just find all unreachable blocks and free them.

Usage

We overloaded global operator 'new'.

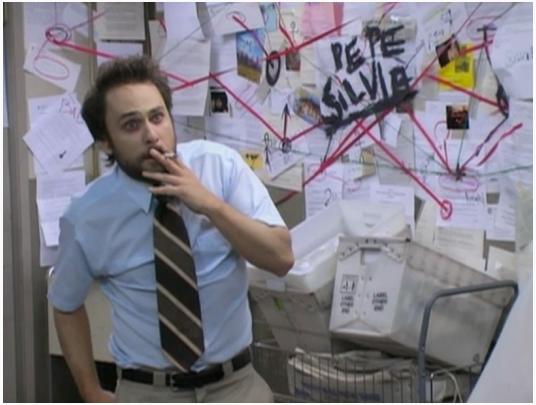
Usage

We overloaded global operator 'new'.
That led to some funny cosequences while testing: 'std::println' uses 'std::format', which, in turn, allocates a string, and its constructor is using 'new'.



Usage

We overloaded global operator ‘new‘.
That led to some funny cosequences while testing: ‘std::println‘ uses ‘std::format‘, which, in turn, allocates a string, and its constructor is using ‘new‘.
But we used ‘println‘ for logging in our allocator. We got to the point where doing literally anything with dynamic memory will cause a stack overflow.



IT WORKS