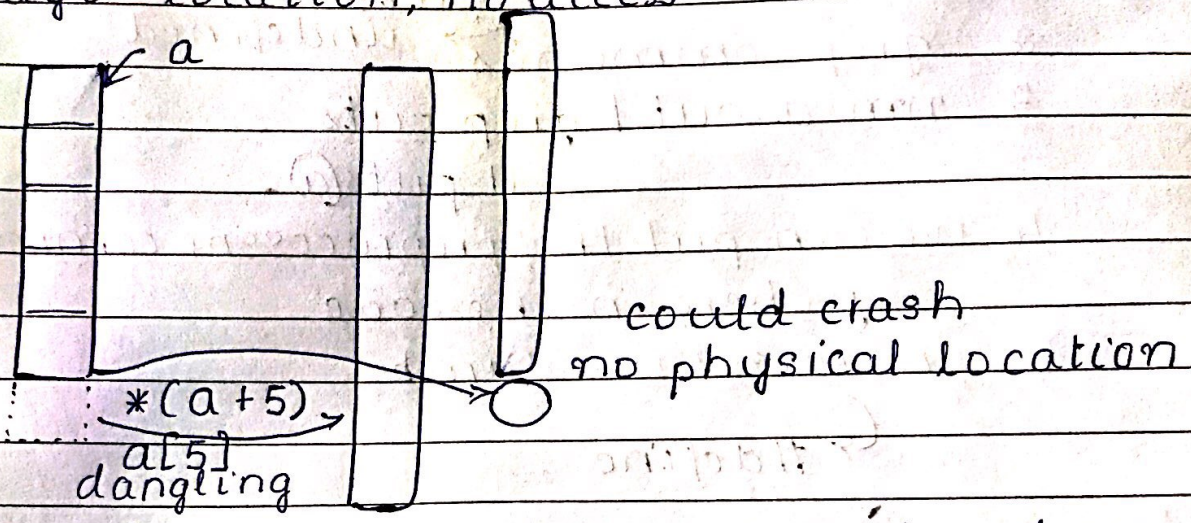
__/__/__

1. $ nm intro-1.o → Text
00000000 T main
U puts → undefined
2. grep commands
3 nm a.out | grep puts
U puts@
4 we can put the preprocessor commands anywhere in the code
5 preprocessor -dumb
↳ #define
6. preprocessor stores table of var
7.
```
int main(){
    foo(); // not defined, but can be
}          compiled.
```
↳ gt says U foo.
8. linker gives error if foo is not defined
normally gives 2 errors
unresolved external
multiple definition
hijacking
9 we could define on command
# cmd.
gcc -DABCD=11 -c 1-client.c
10. we could #define foo = bar

__/__/____

1) type is compile-time & value is run-time
2) dangling - no location, access
   garbage - location, no access
3)




4) The unit of memory that is transferred b/t OS and program
5) At runtime array becomes const pointer. It wont check array at runtime. C/C++ are languages for efficiency
6) free(q+1) → undefined behaviour
   free(q) called 2 times ↗
   ↳ make the prog safe by hijacking a fn
7) can we write our own

gcc -E : shows output of the preprocessor

gcc -c : compile

nm 1_intro.o
using this we can find out what the object file contains

something about puts - means undefined
Unique only to gcc compiler.

~~teach~~ learn how to use grep
command
grep → group regular expressions

pipe symbol → | → output ~~out~~ of left command is input to right.

```
#include <stdio.h>
#define what 100
int main()
{ #define what1 200
  // Print (what)
  // Print (what1) {
```

We can put preprocessor directive anywhere in the program, doesn't matter where.

```
#define what 100
  printf ("%d \n", what)
#define what 300
  printf ("%d \n", what)
```

overrides 100, no problem.

changes at that point only

So o/p: 100
300

If this wasn't there & we tried to call what, we'd get an error error during compilation, but not in the preprocessor stage

nm 3_client.o (xpose)

```
                 U foo
00000000 T main
```

idk how many 0's

#if0 ——→ works like block comment in C

#endif

compile

A program will ~~run~~ if its interface exists and implementation doesn't.

↓

But if we run such a program, we will get an error (usually a linker error)

Linked lists, stacks → interfaces.

linker errors:

1. undefined externals
2. multiple definition

gcc -DABCD=111 -c 3_client.c

(-c: to compile)

↓ (ABCD: variable name)

defining on the command line

~~Hey, to call one function~~ .

PTO.

```
int main()
{
    foo();
}
```

gcc -Dfoo=bar -c 3-client.c

this will make the above code bar() & call that.

So we've changed the file w/o touching the file

This is called Hijacking

* Type is a compile time mechanism
  Value is a runtime mechanism.

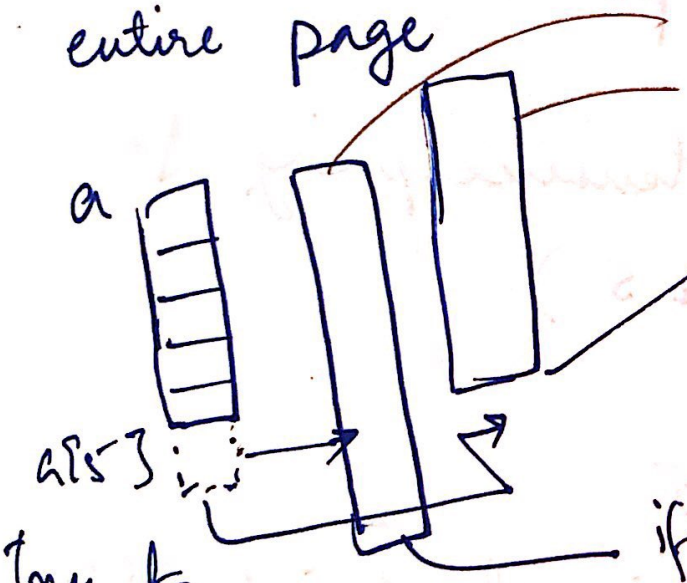
no access, location — garbage

access, no location — dangling pointer

Dangling pointer can be anywhere but ~~heap~~ garbage is only in heap.

The unit by which memory is transferred b/w OS to user program is called page

So when we ask for say 1 byte, OS can't give so less, gives an entire page




what the page might look like

if the page is like this, then program will crash.

Try to access a[5]

if it's this, then program will go through

Try to access a[10000] will give error cause we'll run out of page

Defer Dereferencing dangling pointers is dangerous. Dangling pointers are not.

Java is designed for safety & networking

C & C++ are defined for efficiency

At runtime pointers and arrays are the same.

★ Think.

a ⟶ integer pointer

&a ⟶ array pointer

Bookkeeping & Housekeeping ☆

q = malloc(int * 5)

free(q)

↳ free all 5

free(q+1) ⟶ tries to find this memory location according to book-keeping formula it has

∴ results in undefined behaviour.

malloc, calloc can be hijacked by mymalloc, mycalloc of the same signature

mymalloc can call malloc and it'll work the same way

↓

just keep record of the pointers in a data structure i.e make it a data structure.

---

can we write our own malloc & free where we keep our own bookkeeping ? Something to think about

Double free — program crash.

Steps of getting a C program to run
preprocessing...

- nm objectfile.o ← shows what object files contains

- ↑ T ← text segment
  U ← prtf becomes puts
  once linked, it still stays U but now it shows some global thing → dll → dynamic linked library or shared object (so in linux)

gcc called ld ↑ linker

- We can use ~~#~~ preprocessor commands anywhere in the code
  - no concept of scope, overwrites value, can be put anywhere

- you can compile a program if it has interface even if its implementation isn't there
  ↑
  nm command shows U
  once we have an implementation it ~~#~~ shows some stuff with

Linker ~~error~~ errors → unresolved extern, multiple definitions.
↳ if you have an undefined thing

- To #define on cmd line
  gcc -DABCD=111 -c
- To change foo() to bar() at compile time: -Dfoo=bar
  ↓ w/o touching the file
  Hijacking

- type is compile time mechanism
  value is runtime mechanism

- dangling ptr ⟶ on stack
  ↳ ~~location but no access~~ access, no location

  garbage ⟶ no access, location
  ↳ always on heap

- The unit of memory given by OS to program ⟶ page
  ↑
  ~~page~~ a setting that can be changed

  if you run out of physical space
  ↓
  it'll crash

- dangling ptr. itself isn't dangerous, but dereferencing it is dangerous

- Bookkeeping ⟶ the amt. of heap mem. assigned to a pointer as a record

  free(q+1) ⟶ ~~crash~~ undefined behaviour

- Is it possible to hijack malloc/free?
- Can we create malloc/free ourselves?
  △