2510 : Lecture 23 || March 22, 2017

▷ **Function Pointers**

↳ a function pointer is a variable that stores the address of a function.

We need to be able to:

1) declare a function pointer
2) get the address of a function
3) go to the function where address is stored in the function pointer (and call it)

▷ **Address of a function**

f a function

* We can use &f (or just f) as its address *

void g(int); (we can use &g or g)

▷ **Invoking a function through function pointer**

eg. void g (int);

suppose we have stored its address in the function pointer p. Calling g through p.

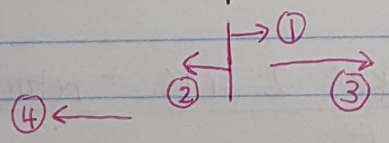
(*p)(2); or p(2);

cannot use → b/c that's accessing a member

▷ **Declaring a function pointer**

void g (int);

void (*p) (int) = &g;



Right-Left Rule

p is a pointer to a fn take

② an int + returns nothing

③ ④

... Lecture 23

Right - Left Rule

1. go right until you hit a right wall (')')
2. then go left until you hit a left wall ('(')
3. jump out + repeat

examples:

1) char * const *p; p is a pointer to a const point to char

② ← ① ↦

2) int *a [100]; a is an array of 100 pointers to int

② ← ① ↦

3) int (*a) [100]; a is a pointer to array of 100 ints

② ← ① ↦ ③ ④ ←

what is the difference between a pointer to an int and a pointer to an array of 100 ints?

int *p; p [0] → 1 int

int (*q) [100]; q [0] → array of 100 ints

4) int add (int, int);

int (*p) (int, int) = add;

④ ← ② ← ① ↦ ③

p is a pointer to a fn that takes 2 ints + returns int

② ③ ④

... Lecture 23

examples cont:

5) int add (int, int); int sub (int, int);
int mult (int, int); int div (int, int);

int (* ops [4]) (int, int);
④ ← ② ← → ① → ③

ops [0] = add;
ops [1] = minus;

We can also use:
int (* ops [ ]) (int, int) = {add, mius, mult, div};

We can index into ops and call the corresponding fN:
printf("%d\n", (* ops [1])(2, 3));
same as
ops [1] (2, 3)

▸ A complicated example

#include <signal.h>
void (*signal (int, void (*)(int)))(int);

signal → returns a function pointer
2nd param → is also a function pointers] 2 fn pointers have the same type

SIGSEGV