PA5 / 6

main.c
- printing
- equality
- CLI args input

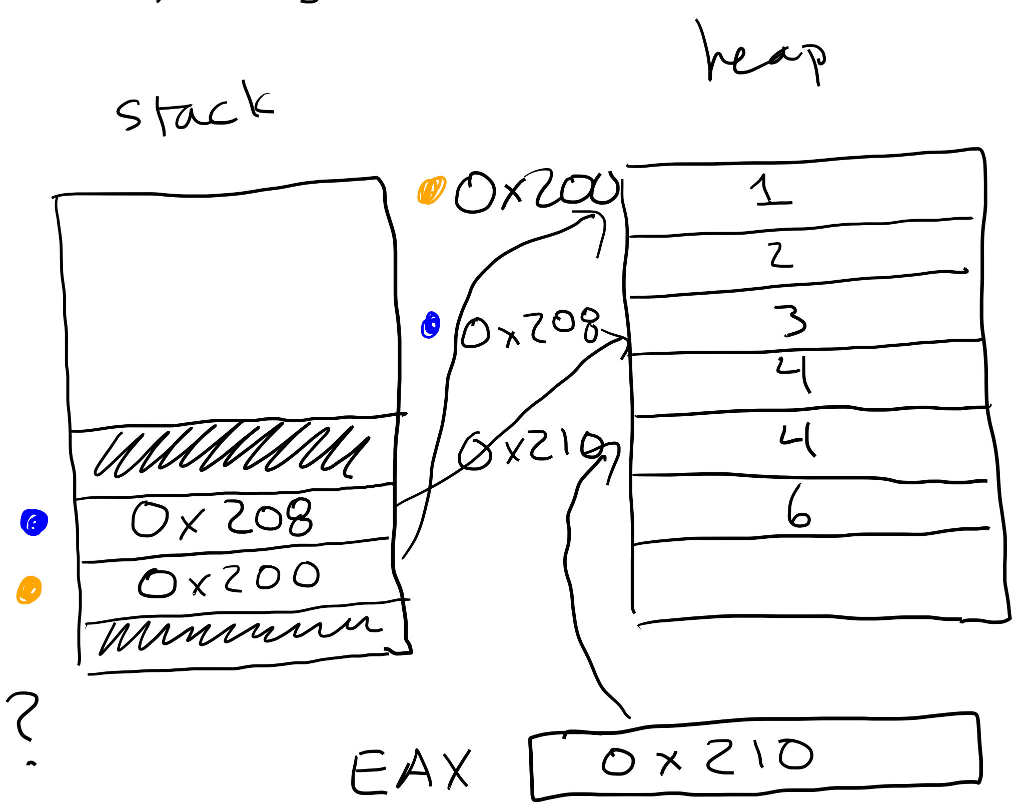
compile.ml
- tuples
- is-tuple
- tup-get

OPEN

[pairs to repr points]

```
(def (add p1 p2)
    (pair (+ (fst p1) (fst p2))
          (+ (snd p1) (snd p2))))

(def (our-main _)
    (add (pair 1 2) (pair 3 4)))
```



How many pairs are allocated by this program?

A: 1
B: 2
[✓ C: 3]
D: 4
E: more

What address for ● ?

A: 200
B: 204
[✓ C: 208]
D: 20C

Will return value from add be stored on stack?

A: Yes
[✓ B: No] (except main)

What should print?

#<pair>

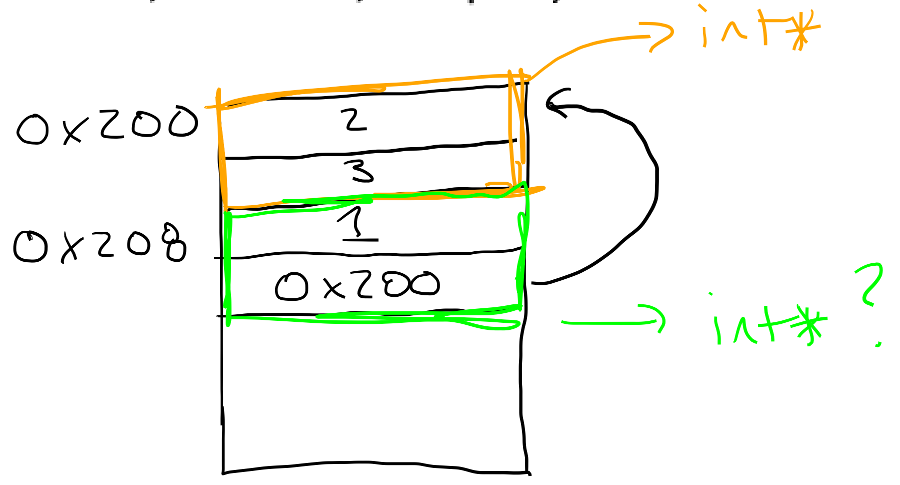
[Pair@ ~~~~]

how? → (4, 6)

Value	**Representation (bits)** 31-bit, 2's complement number	**Representation** hex	decimal	**Useful C type**
9	`0000 0000 0000 0000 0000 0000 0001 0011`	0x00000013	19	int
-2	`1111 1111 1111 1111 1111 1111 1111 1101`	0xFFFFFFFD	-3	int
true	`0000 0000 0000 0000 0000 0000 0000 0110`	0x00000006	6	int
false	`0000 0000 0000 0000 0000 0000 0000 0010`	0x00000002	2	int
(pair 1 2)	`XXXX XXXX XXXX XXXX XXXX XXXX XXXX XX00`	0xXXXXXXX$\boxed{\ }$ ↑ 0, 4, 8, c	big num	int *

tag bits (X1=num, 10=bool, 00=pair)

(pair 1 (pair 2 3))



```
extern int our_code_starts_here()
  asm("our_code_starts_here");

void print_val(int val) {
  if(val & 1) { printf("%d", (val - 1) / 2); }
  else if (val == 6) { printf("true"); }
  else if (val == 2) { printf("false"); }
  else {
     int* vptr = (int*)val;
     printf("(");
     print_val(vptr[0]);
     printf(", ");
     print_val(vptr[1]);
     printf(")");
  }
}

int main(int argc, char** argv) {
  int input = 0;

  int* MEMORY = calloc(10000, sizeof(int));

  if(argc > 1) { input = atoi(argv[1]); }
  int result = our_code_starts_here(input, MEMORY);
  print_val(result);
  printf("\n");
  fflush(stdout);
  return 0;
}
```

why 1 and not 4?

```
union snake_val {
  int as_int;
  union snake_val* as_ptr;
};
extern union snake_val our_code_starts_here()
  asm("our_code_starts_here");

void print_val(union snake_val val) {
  if(val.as_int & 1) {
    printf("%d", (val.as_int - 1) / 2);
  }
  else if (val.as_int == 6) { printf("true"); }
  else if (val.as_int == 2) { printf("false"); }
  else { // It's a pair!
      printf("(");
      print_val(val.as_ptr[0]);
      printf(",");
      print_val(val.as_ptr[1]);
      printf(")");
  }
}

int main(int argc, char** argv) {
  int input = 0;
  int* MEMORY = calloc(10000, sizeof(int));
  if(argc > 1) { input = atoi(argv[1]); }
  union snake_val result;
  result = our_code_starts_here(input, MEMORY);
  print_val(result);
  printf("\n");
  fflush(stdout);
  return 0;
}
```

$\rightarrow$ Union

A: Never seen it

B: Seen it

C: Used it

Discriminated Union

Haskell / OCaml

Undiscriminated Union

C