Lecture 25 (3-23-26)

Structs, Unions (C)

Motivating Questions:

- What is the difference between:
  - a **struct**
  - a **union**

`mock.c`

```
char *mock(const char *s){
     //char t[BUFSIZ]; // you can't Return this since it perishes with stack
frame
     // thats why we need to allocate on heap
     char *t = malloc(strlen(s) + 1);
     strcpy(t, s);

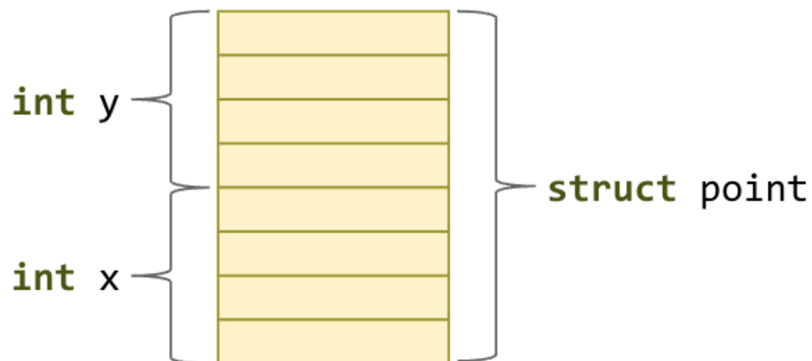
     for (char *c = t; *c; c++){
          *c = ((intptr_t)c % 2)
               ? toupper(*c)
               : tolower(*c);
     }

     return t;
}

int main(int argc, char *argv[]){
     for (int i = 1; i< argc; i++){
          char *t = mock(argv[i])
          puts(t);
          free(t);
     }

}
```

Structs: Overview

- Instead of having **multiple variables** for a **single object**, we can group related data into a **composite data type**
- Struct composite data type:
    - ```
      typedef structP{
        int x;
        int y;
      } Point;
      ```
- Structs: **Memory Allocation**
    - While a **struct** is viewed as **one chunk of memory**, each **attribute** has its own portion of the total allocation:
        - 

    - So structs behave like arrays, the last element is stored at the top in memory (arrays are stored "backwards" in memory)
    - sometimes alignment issues occur, in which the struct size in bytes is rounded up (this enables faster access)

`point.c`

```
/* point.c */

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

typedef struct {    // TODO: s/struct/union
    int x;
    int y;
} Point;
```
```

```c
void point_format(Point *p, FILE *stream) {
    // TODO: Print point to stream
    fprintf(stream, "Point{x: %d, y: %d}\n", p->x, p->y);
}

int main(int argc, char *argv[]) {
    // TODO: Define some points
    Point p0 = {0};
    Point p1 = {1, 2};

    // TODO: Print size and locations of point structure
    printf("sizeof(Point) = %ld\n", sizeof(Point));
    printf("&p0   = 0x%lx\n", (intptr_t)&p0);
    printf("&p0.x = 0x%lx\n", (intptr_t)&p0.x);
    printf("&p0.y = 0x%lx\n", (intptr_t)&p0.y);

    // TODO: Use print_format to print point structure
    point_format(&p0, stdout);
    point_format(&p1, stdout);

    // TODO: Define array of points
    Point pa[] = {
     {3, 4},
     {5, 6},
     {0},       // Sentinel
    };

    // TODO: Loop through array with pointers and print each point
    for (Point *p = pa; p->x && p->y; p++) {
         point_format(p, stdout);
    }

    return EXIT_SUCCESS;
}
```

- if you go from struct to union:
    - ```c
      typedef union {    // TODO: s/struct/union
        int x;
        int y;
      } Point;
      ```
    - then the following addresses are the same:
        - ```c
          printf("&p0   = 0x%lx\n", (intptr_t)&p0);
          printf("&p0.x = 0x%lx\n", (intptr_t)&p0.x);
          ```

```
printf("&p0.y = 0x%lx\n", (intptr_t)&p0.y);
```

Unions: Polymorphic Data Type

- We can view the **same chunk of memory** as **different types** using a **union:**
- ```
  typedef union { //Looks and behaves like
    char c;     // a struct, but all
    int i;      // attributes are stored in
  } Value;       // the same chunk of memory
  ```

`value.c`

```
/* value.c */

#include <limits.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>

typedef union {
    uint64_t number;
    char     string[6];
} Value;

void value_format(Value *v, FILE *stream) {
    // TODO: Print out attributes in Value
    fprintf(stream, "Value{number = %016lx, string = %s}\n", v, v);
}

void value_bytes(Value *v, FILE *stream) {
    // TODO: Print out each byte in Value
}

int main(int argc, char *argv[]) {
     printf("Sizeof(Value) = %lu\n", sizeof(Value));
     puts("")

     Value v0 = {0};
     value_format(&v0, stdout);
     puts("");
```
```

```
    Value v1 = {-1};
    value_format(&v1, stdout);
    puts("");

    Value v2 = {ULONG_MAX};
    printf("ULONG_MAX is %lu\n", ULONG_MAX);
    value_format(&v2, stdout);
    puts("");

    Value v3 = {0x4150524546};
    value_format(&v3, stdout);
    puts("");
    //prints "FERPA" when printed as string
    // "F" is 0x46 so at the end of the hex
    return EXIT_SUCCESS;
}
```