Today's plan

- Class interactions
  - Ex 4-3
  - Today contents
  - Quiz
- Class exercises
  - Ex 5-1

Ex 4-3

- Binary search using pointers:
  ```
  int* search(int* start, int* end, int search_val);
  ```
- `int *start`: a pointer to the start address.
- `int *end`: a pointer to the end address.
- Given `*start` and `*end`, how do you get a pointer that is pointing to the mid element?

  A
  ```
  int* mid = (start + end)/2;
  ```
  B
  ```
  int* mid = (*start + *end)/2;
  ```
  C
  ```
  int* mid = &((*start + *end)/2);
  ```
  D
  ```
  int* mid = start + (end - start)/2;
  ```
  E
  ```
  int* mid = start + (end - start)/sizeof(int)/2;
  ```

Ex 4-3

- Sudoku solution checker:
  `makeCol, checkRows, checkCols, checkCubes`
- Multi-dimensional arrays.
- Dynamic memory allocation managements:
  `malloc, calloc, realloc, free`.
- Using valgrind to check if there is a memory problem.
- How do you allocate dynamically an integer array of size 99?

A
```
int* foo = malloc(99);
```
B
```
int* foo = malloc(100);
```
C
```
int* foo = malloc(sizeof(int)*99);
```
D
```
int* foo = malloc(sizeof(int)*100);
```
E
```
int* foo = calloc(99, sizeof(int));
```

Ex 4-3

- Sudoku solution checker:
  `makeCol, checkRows, checkCols, checkCubes`
- Multi-dimensional arrays.
- Dynamic memory allocation managements:
  `malloc, calloc, realloc, free`.
- Using valgrind to check if there is a memory problem.
- Given `int a[10] = {[3]=1}, b = 3;`, which of the following is true?

  A `a[b] == *(a + b);`
  B `&a[10] - &a[0] == 10;`
  C `a[2] == a[8];`
  D `&a[b] == a + b;`
  E `*(&a[0] + b) == 1;`

Lifetime and scope

- What is the output of the below program?

```c
#include <stdio.h>
int foo;
void bar() {
    int foo = 3;
    {
        int foo = 1;
        printf("%d; ", foo);
        foo = 2;
    }
    printf("%d; ", foo);
}
void baz() { printf("%d; ", foo); }
int main() {
    {
        int foo = 5;
        bar();
        printf("%d; ", foo);
    }
    baz();
    return 0;
}
```

A 1; 5; 5; 0;

B 1; 3; 5; 0;

C 0; 3; 3; 0;

D 1; 3; 5; 3;

E Unpredictable.

Lifetime and scope

- What is the output of the below program?

```
1  #include <stdio.h>
2  void m() {
3      static int x = 5;
4      x++;
5      printf("%d ", x);
6  }
7  int main(void) {
8      m();
9      m();
10     return 0;
11 }
```

A 6 7

B 6 6

C 5 6

D 5 5

E Unpredictable.

Lifetime and scope

- (extern) What is the output of the below program?

```
1   #include <stdio.h>
2   int foo;
3   void bar() {
4       int foo = 3;
5       {
6           extern int foo;
7           printf("%d; ", foo);
8           foo = 2;
9       }
10      printf("%d; ", foo);
11  }
12  void baz() { printf("%d; ", foo); }
13  int main() {
14      {
15          int foo = 5;
16          bar();
17          printf("%d; ", foo);
18      }
19      baz();
20      return 0;
21  }
```

A 0; 5; 5; 0;

B 0; 3; 5; 2;

C 0; 3; 3; 2;

D 0; 3; 5; 0;

E Unpredictable.

Lifetime and scope

- Key takeaways:
  - `static` and `global` variables are initialized to zeros (those are in data segment).
  - inner scope will shadow the outer scope's variables if they have the same name.
  - `extern` sometimes can be used to shadow back the local variable to access the global one.
  - Lifetime of a variable is the lifespan a variable lives in the memory.
  - Scope of a variable is where the variable can be accessed.

Struct

- What is the output of the following program?

```
1  #include <stdio.h>
2  struct Pokemon {
3      char type;
4      char name[12];
5  };
6  struct Pokemon makeElectric(struct Pokemon p) {
7      p.type = 'E';
8      return p;
9  }
10 int main(void) {
11     struct Pokemon charmander = {
12         'F', "Charmander"
13     };
14     makeElectric(charmander);
15     printf("%s (%c)\n",
16     charmander.name,
17     charmander.type);
18     return 0;
19 }
```

A Charmander (F)

B Charmander (E)

C Pikachu (E)

D Some other output

E Unpredictable.

Struct

- What is the size of `struct Pokemon`?

```
1  #include <stdio.h>
2  struct Pokemon {
3      char type;
4      char name[12];
5  };
6  struct Pokemon makeElectric(struct Pokemon p) {
7      p.type = 'E';
8      return p;
9  }
10 int main(void) {
11     struct Pokemon charmander = {
12         'F', "Charmander"
13     };
14     makeElectric(charmander);
15     printf("%s (%c)\n",
16     charmander.name,
17     charmander.type);
18     return 0;
19 }
```

A 12

B 13

C 14

D 16

E Unpredictable.

Struct

- (Padding) What is the size of `struct Pokemon`?

```
1  #include <stdio.h>
2  struct Pokemon {
3      char type;
4      char name[12];
5      int foo[2];
6  };
7  struct Pokemon makeElectric(struct Pokemon p) {
8      p.type = 'E';
9      return p;
10 }
11 int main(void) {
12     struct Pokemon charmander = {
13         'F', "Charmander", {0, 1}
14     };
15     makeElectric(charmander);
16     printf("%s (%c)\n",
17     charmander.name,
18     charmander.type);
19     return 0;
20 }
```

A 20

B 21

C 22

D 24

E Unpredictable.

Struct

- (Assignment) What is output of the following program?

```c
#include <stdio.h>
struct Pokemon {
    char type;
    char name[12];
    int foo[2];
};
struct Pokemon makeElectric(struct Pokemon p) {
    p.type = 'E';
    p.foo[0] = 3;
    return p;
}
int main(void) {
    struct Pokemon charmander = {
        'F', "Charmander", {0, 1}
    };
    charmander = makeElectric(charmander);
    printf("%s (%c) - %d\n",
    charmander.name,
    charmander.type,
    charmander.foo[0]);
    return 0;
}
```

A Charmander (F) - 0

B Charmander (E) - 0

C Charmander (F) - 3

D Charmander (E) - 3

E Unpredictable.

Struct and random numbers

- Key takeaways:
  - You can define your own type as a `struct` by grouping other types together.
  - Padding happens. Usually C compiler aligns it to 4-bytes.
  - Memory is linear in a `struct`.
  - `struct` assignment copies everything in a `struct` to another. Same as `memcpy`.
  - A pointer to a `struct` is pointing to its start address.
  - Pointer arithmetic will advance pointer of a `struct` by its byte size.
  - You can use `rand()` in `stdlib.h` to generate random integers.
  - For testing purpose, you can set the seed using `srand()` to fix the random integer sequence.
  - Otherwise, you should use `srand(time(0))` to generate a time dependent random integer sequence.

Quiz

Quiz on Menti!

What is struct in C?

Mentimeter

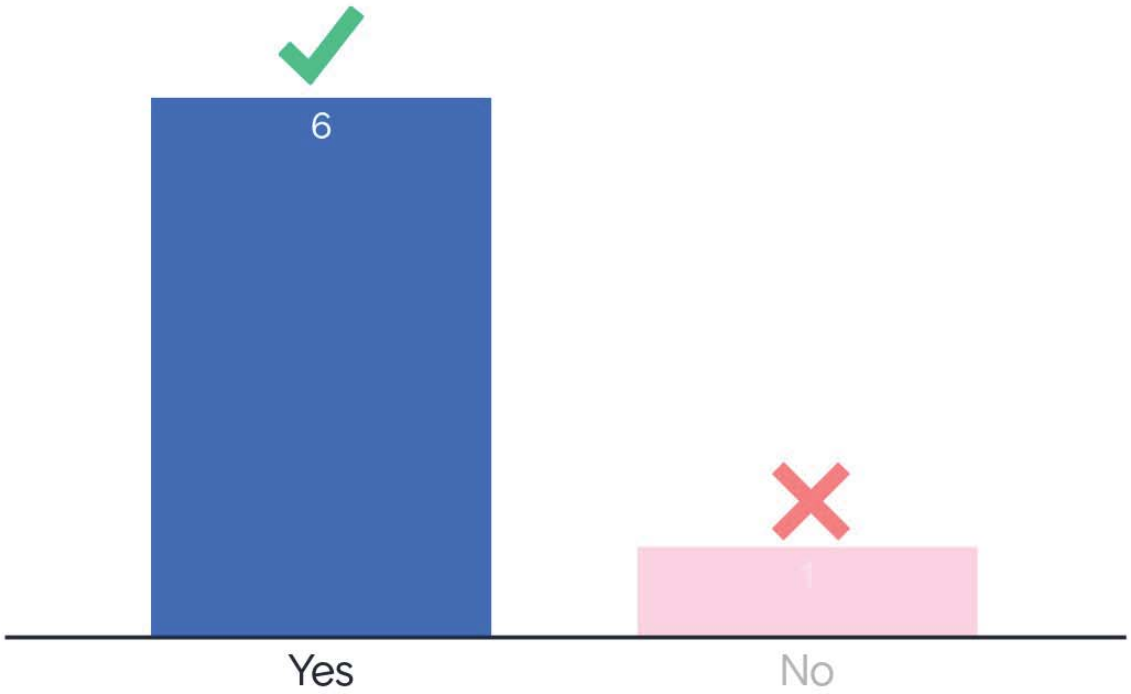
- group variables together ✕
- Collection of related variables ✕
- A group of fields. ✕
- essentially a "multi-variable" ✕
- an array to define variables ✕
- a collection of related variables ✕

The correct answer is: A user defined data type, which is a collection of related variables, called fields.

6

Is passing a struct into a function passed by value?

Mentimeter



7

What is the size of a *struct*? What is structure padding in C?

Mentimeter

- size = sum of sizes of the fields, padding happens when fields are of different types to "align" the data in memory
- The minimum size is the sum of the size of all its fields, however padding can occur, often to the next multiple of 4 bytes
- The sum of the size of its fields. Padding is used when fields are of different types
- 32 bits and padding is

The correct answer is: It is the total byte sizes of all fields in a struct + padding. Typically C compiler pad the struct to align with 4-bytes.

4

What is the difference between lifetime and scope of a variable?

Mentimeter

- Lifetime is period of time when value exists in memory while scope is the region of code where the variable is accessible ✕
- Scope is when a variable is no longer accessible, life time is when it exists ✕
- lifetime - how long it exists in memory, scope - where it can be called/used ✕
- lifetime is how long a variable exists and scope is what functions the variable can be accessed by ✕
- scope is where the variable can be accessed lifetime is how long the variable is stored ✕

The correct answer is: Lifetime is the lifespan of a variable in the memory. Scope is where the variable can be accessed.

5

What is variable shadowing (i.e. hiding)?

Mentimeter

- Variable hiding is when a variable declared within inner scope has the same name as a variable declared in outer scope. ✗
- When a variable has the same name in a lower scope, but also upper scope ✗
- when inner scope supercedes outer scope ✗
- when variable has the same name as one in a different scope ✗
- when a global variable overrides a local variable ✗

The correct answer is: When an inner scope variable and an outer scope variable have the same name. The inner one hides the outer one.

6

Ask me anything

Mentimeter

0 questions
0 upvotes

Class exercises

Ex 5-1