Solutions for Quiz 1: C Primer

CS 351-CUG, Fall 2023

Problem 2

```
int cat_ages[20];
```

Declares an array called `cat_ages` consisting of 20 ints

Problem 3

```
double feeding_amounts[20][3] = {[0][0] = 5., [0][1] = 0.5, [0][2] = 1.5};
```

Answer Defines an array that is 2D with dimensions 20x3 called `feeding_amounts`. Initializes the first row to the values 5.0, 0.5, and 1.5.

Problem 4

```
char *kitty_complaints[15];
kitty_complaints[0] = "not enough belly rubs";
kitty_complaints[1] = "too many belly rubs";
kitty_complaints[2] = "food bowl is empty again";
kitty_complaints[3] = "meow."
```

Defines an array of pointers to characters (or an array of variable length strings, whichever the student prefers to use) 15 elements long. Sets the first four of these elements to various strings.

Problem 5

```
enum Coats {TABBY, GREY, BLACK, TORTOISE};
enum Coats fur_color;
fur_color = GREY;
printf("This cat has category %s fur \n", fur_color )
```

This code creates an enum table called `Coats` with four options Then declares an enum called `fur_color` Then prints "This cat has category 1 fur"

Problem 6

```
double front_paws, back_paws, *eyes, *ears;
```

Declares two doubles, `front_paws` and `back_paws`, and two pointers to doubles, `eyes` and `ears`

Problem 7

```
int tail_length = 8;
int *point_to_the_tail = &tail_length;
```

Defines an int called `tail_length` and initializes it to 8, then defines a pointer to an int called `point_to_the_tail` and sets it to the address of `tail_length`

Problem 8

```
double *kitty_sleeping_area, kitties_needed_more_room;
int num_kitties = 4;
kitty_sleeping_area = (double *) malloc(sizeof(double) * num_kitties * 3);
...
kitties_needed_more_room = (double *) realloc(kitty_sleeping_area,
sizeof(double) * num_kitties * 10);
```

1. Declares two double pointers, `kitty_sleeping_area` and `kitties_needed_more_room`
2. Defines an int called `num_kitties` and sets it to 4
3. Allocates space to `kitty_sleeping_area` using `malloc`, and casts the returned pointer to a double
4. Size allocated is `(size of a double * number of kitties * 3)`
5. Reallocates space to `kitties_needed_more_room` using `realloc`, and casts the returned pointer to a double
6. Uses `kitty_sleeping_area`, and now uses size `(size of a double * number of kitties * 10)`

Problem 9

```
#include <stdio.h>

#DEFINE YES 500
#DEFINE NO 0

int is_this_a_good_kitty(char *name){
    return YES;
}

int main(){
    char *this_cat = "Fluffy";
    if (is_this_a_good_kitty(this_cat)) {
        printf("%s is a good cat\n", this_cat);
    }
    else {
        printf("%s is NOT a good cat\n", this_cat);
    }
}
```

1. Makes two macros, `YES = 500` and `NO = 0`
2. Defines a function that returns an int called `is_this_a_good_kitty` and takes a pointer to a char (or a string) called `name`
3. Always returns `YES`
4. In main,
   (a) defines a char pointer (or a string) called `this_cat` and sets it to `Fluffy`
   (b) has an if loop that calls `is_this_a_good_kitty` on `Fluffy`
   (c) Since the function always returns a non-zero value, the if will always be true, so always prints `Fluffy is a good cat`

Problem 10

```
char * kitty_caught_a_mouse = (char *) malloc(sizeof(char) * 50);

kitty_caught_a_mouse[0] = 'o';
kitty_caught_a_mouse[1] = 'h';
kitty_caught_a_mouse[2] = ' ';
kitty_caught_a_mouse[3] = 'n';
kitty_caught_a_mouse[4] = 'o';
kitty_caught_a_mouse[5] = '!';
kitty_caught_a_mouse[6] = '\0';

printf("%s \n", kitty_caught_a_mouse);
```

1. Makes a pointer to a character called `kitty_caught_a_mouse`
2. `Mallocs` it space for 50 characters (creates a buffer of 50 characters)
3. One character at a time, adds `oh no!\0` to the buffer
4. Prints it, making `oh no!` print out
5. `Frees` the space

Problem 11

```
void is_kitty_clean(int had_bath, int nails_trimmed, int teeth_brushed,
int fur_brushed){
    int is_clean = had_bath && nails_trimmed && teeth_brushed && fur_brushed;
    if (had_bath) {
        printf("I hope kitty didn't scratch you in the bath! \n");
    }
    if (is_clean) {
        print("What a tidy kitty!\n");
    }
    else {
        print("Kitty isn't cleaned up yet.\n");
    }
}
```

1. Defines a function `is_kitty_clean` which takes four ints: `had_bath`, `nails_trimmed`, `teeth_brushed`, `fur_brushed`
2. If `had_bath` was non-zero, prints `I hope kitty didn't scratch you in the bath!`
3. If all inputs are non-zero, prints `What a tidy kitty!`
4. Otherwise, prints `Kitty isn't cleaned up yet.`

Problem 12

```
#DEFINE YES 1
#DEFINE NO 0
#DEFINE UNKNOWN -1
enum is_fluffy {NOPE, VERY};

struct kitty_profile {
    char * name;
```

```
    enum is_fluffy;
    int likes_pets;
    double height;
    int age;
}

...

struct kitty_profile cat_over_here;
cat_over_here.name = "Charlie";
cat_over_here.is_fluffy = VERY;
cat_over_here.likes_pets = NO;
cat_over_here.height = 8.5;
cat_over_here.age = 2;
struct kitty_profile * cat_over_there = malloc(sizeof(struct kitty_profile));
cat_over_there->"Not sure yet";
cat_over_there->is_fluffy = NOPE;
cat_over_there->likes_pets = UNKNOWN;
cat_over_there->height = 10.0;
cat_over_there->age = UNKNOWN;
```

1. Defines three macros: `YES`, `NO`, and `UNKNOWN` with values 1, 0, and -1
2. Makes an enum called `is_fluffy` with two members: `NOPE` and `VERY`
3. Makes a structure called `kitty_profile` with the following:
    - A pointer to a character array (or a string) called `name`
    - an instance of the enum `is_fluffy`
    - an int called `likes_pets`
    - a double called `height`
    - an int called `age`
4. Defines a `kitty_profile` called `cat_over_here` with attributes `Charlie`, `VERY`, `YES`, 8.5, and 2 (and uses dot notation)
5. Defines a `kitty_profile` POINTER called `cat_over_there` with attributes `Not sure yet`, `NOPE`, `UNKNOWN`, 10.0, and `UNKNOWN` (using `->` notation)