Exercise 13-14

Jonas Pedersen, Søren Graae

January 10, 2025

Exercise 13

13a

What would you find if you read the memory where a pointer is stored?

If you read the memory where a pointer is stored, you would find the memory address that the pointer is pointing to. This address is the value of the pointer itself. To access the data at that address, you would dereference the pointer, which retrieves the value stored at the pointed-to memory location.

13b

Why does the value of the pointer in 16.2 change to something seemingly random after the first five iterations?

The pointer starts at the address of the first element in the array and increments through the array elements. After the fifth iteration, the pointer moves beyond the array bounds, accessing memory outside the array, which leads to undefined behavior and seemingly random values.

13c

Use a pointer to change the value of an integer variable. You can declare a pointer to an integer variable like this:

```
    int *pointer = &var;
```

```c
#include <stdio.h>

int main() {
    int var = 10;              // Declare an integer
       variable and initialize it
    int *pointer = &var;       // Declare a pointer and
       assign the address of the variable

    printf("Original value of var: %d\n", var);

    *pointer = 20;             // Use the pointer to
       change the value of the variable
    printf("Updated value of var: %d\n", var);

    return 0;
}
```

13d

Examine the following code. What is it doing?

```c
int a[5] = {9,2,42,5,8};
int *pointer = &a[0];
void loop() {
    Serial.print("Address of pointer is ");
    Serial.print((unsigned int)pointer, HEX);
    Serial.println();
    Serial.print("Value of pointer is ");
    Serial.print(*pointer);
    Serial.println();
    pointer++;;
    delay(3000);
}
```

The code uses a pointer to iterate through an array, printing the address the pointer is pointing to and the value at that address. After printing, the pointer increments to the next array element, but it continues past the array bounds after five iterations, leading to undefined behavior. The loop includes a 3-second delay to space out the outputs for easier observation.

13e

Write a function that swaps the value of two integer variables using pointers

```c
void swap(int *pointer_a, int *pointer_b) {
    int temp = *pointer_a;    // Store the value pointed to by pointer_a in temp
    *pointer_a = *pointer_b; // Assign the value pointed to by pointer_b to pointer_a
    *pointer_b = temp;       // Assign the value in temp to pointer_b
}
```

Exercise 14

14a

What is a member?
A member is a variable (data member) or function (member function) that is part of a class, struct, or union, defining its attributes and behaviors. Data members store information about an object, while member functions define actions that can be performed by the object.

14b

1. `struct.member`:
   Accesses the member of a **struct instance** directly. It is used when `struct` is a regular object (not a pointer).

2. `*(struct).member`:
   This is incorrect or unusual syntax. The `*` operator here attempts to dereference a non-pointer object (`struct`), leading to an error. If the parentheses are removed and `struct` is actually a pointer (e.g., `*struct.member`), it would dereference the member.

3. `*struct.member`:
   Accesses and dereferences the value of `struct.member` if `member` itself is a pointer. It requires that `struct` is not a pointer but has a member that is a pointer.

4. `struct->member`:
   Accesses the member of a struct when `struct` is a pointer. The `->` operator is shorthand for dereferencing the struct pointer and accessing its member (equivalent to `(*struct).member`).

When Should `struct` Be a Pointer?

- `struct` should be a pointer when:
  - You want to dynamically allocate the struct on the heap.
  - You need to pass the struct to a function without copying the entire object.
  - You are working with a linked structure (e.g., linked lists).

When Should `member` Be a Pointer?

- `member` should be a pointer when:
  - The member references data stored outside the struct.
  - The size of the data being referenced is large, and copying it would be inefficient.
  - The struct needs to point to dynamically allocated or shared data.

14c

Make a structure called Animal, include at least 5 members with 3 different data types. (Family, weight, alive, place of capture etc.)

```c
    #include <stdio.h>
#include <stdbool.h> // For using the 'bool' data type in C

struct Animal {
    char family[50];        // String to store the family of the animal
    double weight;          // Double to store the weight of the animal in kilograms
    bool alive;             // Boolean to indicate if the animal is alive
    char placeOfCapture[100]; // String to store the place of capture
    int age;                    // Integer to store the age of the animal
};

int main() {
    struct Animal tiger = {"Felidae", 220.5, true, "Sundarbans, India", 5};

    printf("Animal Information:\n");
    printf("Family: %s\n", tiger.family);
    printf("Weight: %.2f kg\n", tiger.weight);
    printf("Alive: %s\n", tiger.alive ? "Yes" : "No");
    printf("Place of Capture: %s\n", tiger.placeOfCapture);
    printf("Age: %d years\n", tiger.age);

    return 0;
}
```

14d

Declare two different animals of your choosing. Set all members of the two animals.

```c
#include <stdio.h>
#include <stdbool.h> // For using the 'bool' data type in C

struct Animal {
    char family[50];          // String to store the family of the animal
    double weight;            // Double to store the weight of the animal in kilograms
    bool alive;               // Boolean to indicate if the animal is alive
    char placeOfCapture[100]; // String to store the place of capture
    int age;                  // Integer to store the age of the animal
};

int main() {
    // Declare and initialize two Animal instances
    struct Animal lion = {"Felidae", 190.0, true, "Masai Mara, Kenya", 8};
    struct Animal elephant = {"Elephantidae", 5400.0, true, "Chobe National Park, Botswana", 25};

    // Print the details of the lion
    printf("Lion Information:\n");
    printf("Family: %s\n", lion.family);
    printf("Weight: %.2f kg\n", lion.weight);
    printf("Alive: %s\n", lion.alive ? "Yes" : "No");
    printf("Place of Capture: %s\n", lion.placeOfCapture);
    printf("Age: %d years\n\n", lion.age);

    // Print the details of the elephant
    printf("Elephant Information:\n");
    printf("Family: %s\n", elephant.family);
    printf("Weight: %.2f kg\n", elephant.weight);
    printf("Alive: %s\n", elephant.alive ? "Yes" : "No");
    printf("Place of Capture: %s\n", elephant.placeOfCapture);
    printf("Age: %d years\n", elephant.age);

    return 0;
}
```

14e

```c
    #include <stdio.h>
#include <stdbool.h>

// Define the Animal structure
struct Animal {
    char family[50];
    double weight;
    bool alive;
    char placeOfCapture[100];
    int age;
};

```

```c
// Function to print the details of an Animal
void printAnimal(struct Animal a) {
    printf("Animal Information:\n");
    printf("Family: %s\n", a.family);
    printf("Weight: %.2f kg\n", a.weight);
    printf("Alive: %s\n", a.alive ? "Yes" : "No");
    printf("Place of Capture: %s\n", a.placeOfCapture);
    printf("Age: %d years\n", a.age);
    printf("\n");
}

int main() {
    // Declare and initialize two Animal instances
    struct Animal lion = {"Felidae", 190.0, true, "Masai Mara, Kenya", 8};
    struct Animal elephant = {"Elephantidae", 5400.0, true, "Chobe National Park, Botswana", 25};

    // Print their details using the printAnimal function
    printAnimal(lion);
    printAnimal(elephant);

    return 0;
}
```