CS101 Lab-9

Fundamentals of Computer Programming

Shreya Agarwal & Rachit Nimavat

This is the last lab before your practical exams. Clarify doubts with the help of your lab instructors and peer mentors!

Instructions for the lab assignment:

Solve the **first three problems** in the lab. The other problems are for homework. You should attempt them after we cover pointer `malloc`/`free` in the class. The last problem is like a course micro-project that ties up a lot of concepts.

1. Recall the string-reversal problem from Lab-7. Write a **recursive function** `void *reverse(char *start, char* end);` that takes a pointer to its first character and a pointer to its last character and modifies the string directly in memory so that it becomes reversed. It does not return anything. Hint: test your function with `char s[] = "hello";` and **not** `char *s = "hello";` Why? You'll need to call the function with parameters `start=s` and `end=s+strlen(s)-1`. Why?

2. Create a struct to store points in 3D space.

   ```
   typedef struct point {
        int x, y, z;
   } Point;
   ```

   Implement the function with the following prototype to calculate the distance between two points.

   ```
   double dist(Point a, Point b);
   ```

   Declare an array `Point arr[];` of points and use the function `dist` to find the point in the array that is closest to `(0,0,0)`, breaking ties arbitrarily. Hint: include `<math.h>` and use `gcc -lm` to compile.

3. Create an inventory system. Write a function that searches the inventory array for an item by its ID and returns a *pointer* to that item's `struct`.

```
typedef struct {
    int id;
    char name[100];
    int quantity;
} Product;
```

Implement the following functions:

- **`Product* findProductById(Product inventory[], int inventory_size, int id)`**
  - Takes the inventory array, the number of items, and an ID to search for. It loops through the array. If it finds a Product with a matching id, it returns a pointer to that Product struct within the array. If no product is found, it returns **`NULL`**.
- **`void restockProduct(Product *product, int amount)`**
  - Takes a pointer to a single Product. Increments the `product->quantity` by `amount`.

4. Implement the function with the following prototype:

```
int *evenize(int *arr, int len, int* new_len);
```

This function takes a pointer to an `int` array **`arr`** (which you can assume was allocated via `malloc`), its **`len`**, and a pointer **`int* new_len`** to an integer. It then counts even numbers in **`arr`**, and uses `malloc` to create a new array on the heap that is just large enough to store these even numbers. It should write the size of the new array at the location pointed by **`new_len`**. It should then free the memory allocated to **`arr`** and return the pointer to the new array. Return `NULL` if malloc fails or no even numbers are found.

5. Recall the calculator program we did when studying if-else and switch. Today, we will implement the same using function pointers. Create a "calculator" where the operations themselves are stored in an array of `structs`. Each `struct` will pair a symbol (like `+`) with a pointer to the function that performs that operation.

```
// Define a type for a function pointer that takes two ints and returns an int
typedef int (*ArithmeticFunc)(int, int);
int op_add(int a, int b) { return a + b; }
```

```
int op_subtract(int a, int b) { return a - b; }
int op_multiply(int a, int b) { return a * b; }
typedef struct {
    char symbol;
    ArithmeticFunc func_ptr; //function pointer to op.
} Operation;
Operation operations[] = {
    {'+', op_add},
    {'-', op_subtract},
    {'*', op_multiply}
};
```

Ask the user to enter a number, an operator, and a second number (e.g., **10*5**). Loop through the operations' array and find the `struct` that matches the user's operator. When found, call the function using the pointer from the struct: **`int result = operations[i].func_ptr(num1, num2);`** Print the calculated result. If no matching operator is found, print an error.

Solution for Lab-8 Problem-1

```
#include<stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 5, y = 10;
    printf("Before swap: x = %d, y = %d\n", x, y);
    swap(&x, &y);
    printf("After swap: x = %d, y = %d\n", x, y);
    return 0;
}
```

Solution for Lab-8 Problem-2

```c
#include<stdio.h>
long sum_of_digits(long n) {
    if(n < 0) return sum_of_digits(-n);
    if(n == 0) return 0;
    return (n % 10) + sum_of_digits(n / 10);
}

int main() {
    long n = -12345;
    printf("Sum of digits of %ld is %ld\n", n, sum_of_digits(n));
    return 0;
}
```

Solution for Lab-8 Problem-3

```c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>
#define MAX_LEN 256

long get_input_and_time(char *s) {
    time_t start = time(NULL);
    fgets(s, MAX_LEN, stdin);
    s[strlen(s) - 1] = '\0'; // Remove newline char from fgets input
    //yes, this is coarse measurement. Getting time in milliseconds is
too much hassle that's not worth the effort.
    return time(NULL) - start;
}

int count_words(const char line[]) {
    //proxy: just count number of spaces and add 1
    int words = 1;
    for(int i=0; i<strlen(line); i++) words += (line[i] == ' ');
    return words;
}

void test(const char line[]) {
    printf("Test line: %s\n", line);
    printf("Type the line above and press Enter: ");

    char s[MAX_LEN];
    float typing_time = get_input_and_time(s);
    float WPM = 60*count_words(s)/typing_time;
    if(strcmp(s, line)) printf("Incorrect!");
```

```
    else printf("Correct! Time taken: %.1f seconds. WPM:%.1f\n",
typing_time, WPM);
}

int main() {
    srand(time(NULL)); //commenting this line will produce the same
output on every run
    char *lines[] = {
        "Mogambo khush hua",
        "Mere paas Maa hai",
        "Don ko pakadna mushkil hi nahin, namumkin hai",
        "Kitne aadmi the?",
        "Babumoshai, zindagi badi honi chahiye, lambi nahi"
    };
    int total_lines = sizeof(lines) / sizeof(lines[0]);
    int random_index = rand() % total_lines;
    char *random_line = lines[random_index];
    test(random_line);
    return 0;
}
```

Solution for Lab-8 Problem-4

```
#include<stdio.h>
#include<string.h>

void case_sensitive_censor(char line[], const char badword[]) {
    char *res = strstr(line, badword);
    if(res == NULL) return;
    int start = res - line;
    for(int i=start; i<start + strlen(badword); i++) line[i] = '*';
    case_sensitive_censor(line, badword);
}

void case_insensitive_censor(char line[], const char badword[]) {
    //strstr is case sensitive, so we need to use strcasestr
    //If strcasestr is not available, we can copy both strings to
lower case and then use case_sensitive_sensor. We must then port the
censored solution back to the original string.
    char *res = strcasestr(line, badword);
    if(res == NULL) return;
    int start = res - line;
    for(int i=start; i<start + strlen(badword); i++) line[i] = '*';
    case_insensitive_censor(line, badword);
}
```

```c
int main() {
    char line[] = "Kabaddi is a great sport! Badminton is also fun.";
    char *badWord = "bad";
    case_sensitive_censor(line, badWord);
    printf("%s\n", line);
    case_insensitive_censor(line, badWord);
    printf("%s\n", line);
}
```

Solution for Lab-8 Problem-5

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

void scale_array(int arr[], int n, int f);
void print_array(int arr[], int n);

int main() {
    int arr[] = {1, 3, 5, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    scale_array(arr, n, 2);
    print_array(arr, n);
}

void scale_array(int arr[], int n, int f) {
    //arr decays to pointer here. sizeof(arr) would give size of
pointer, not array
    for (int i = 0; i < n; i++) {
        arr[i] *= f;
    }
}

void print_array(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```