Lab 6

Pointers, Dynamic Memory Allocation and their Applications

Module ID: CX-202

Problem Solving Skills using C

By: Nehal Alsuhaibani

Submission date: 30 Oct 2024

Source code and Git Repository:

Please find the Git repository in the link: https://github.com/Nehal-2/Lab6_C.git

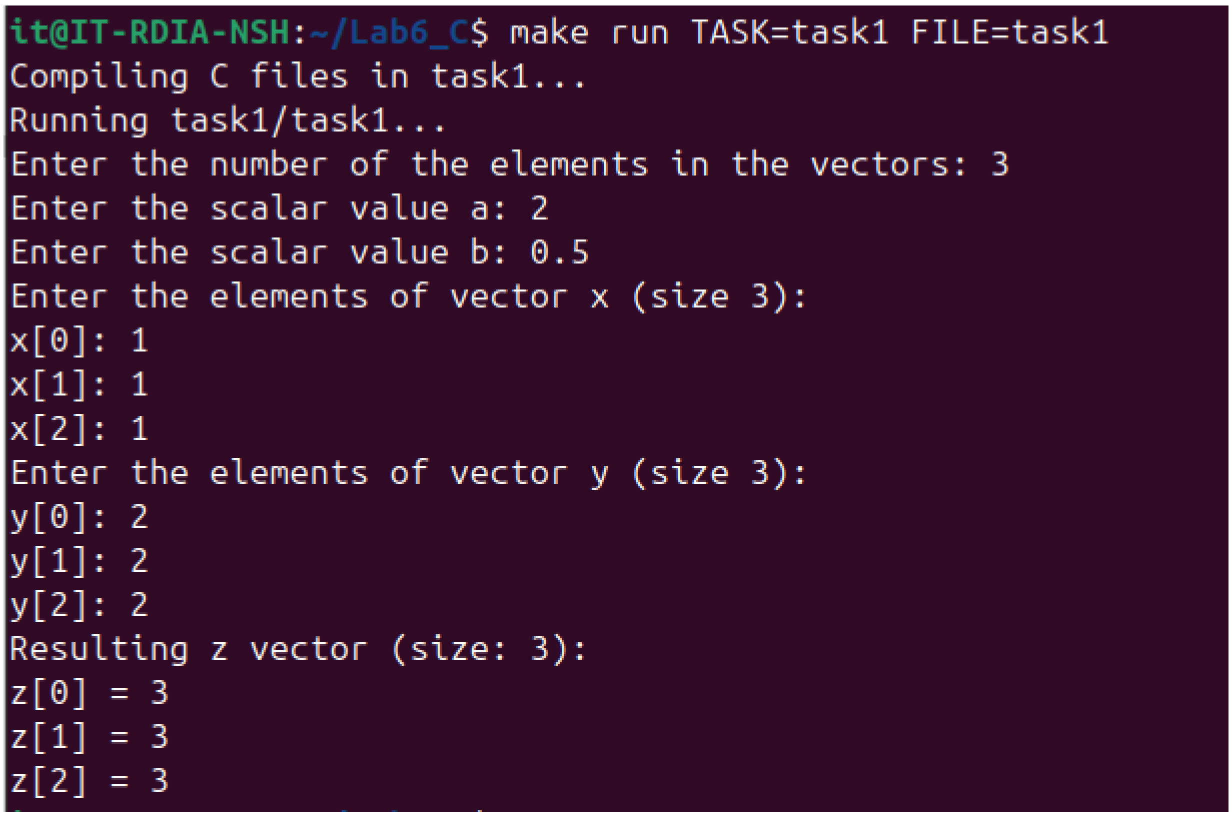
TASK 1: axpy (z = ax + by)

Write a c program to do the following:

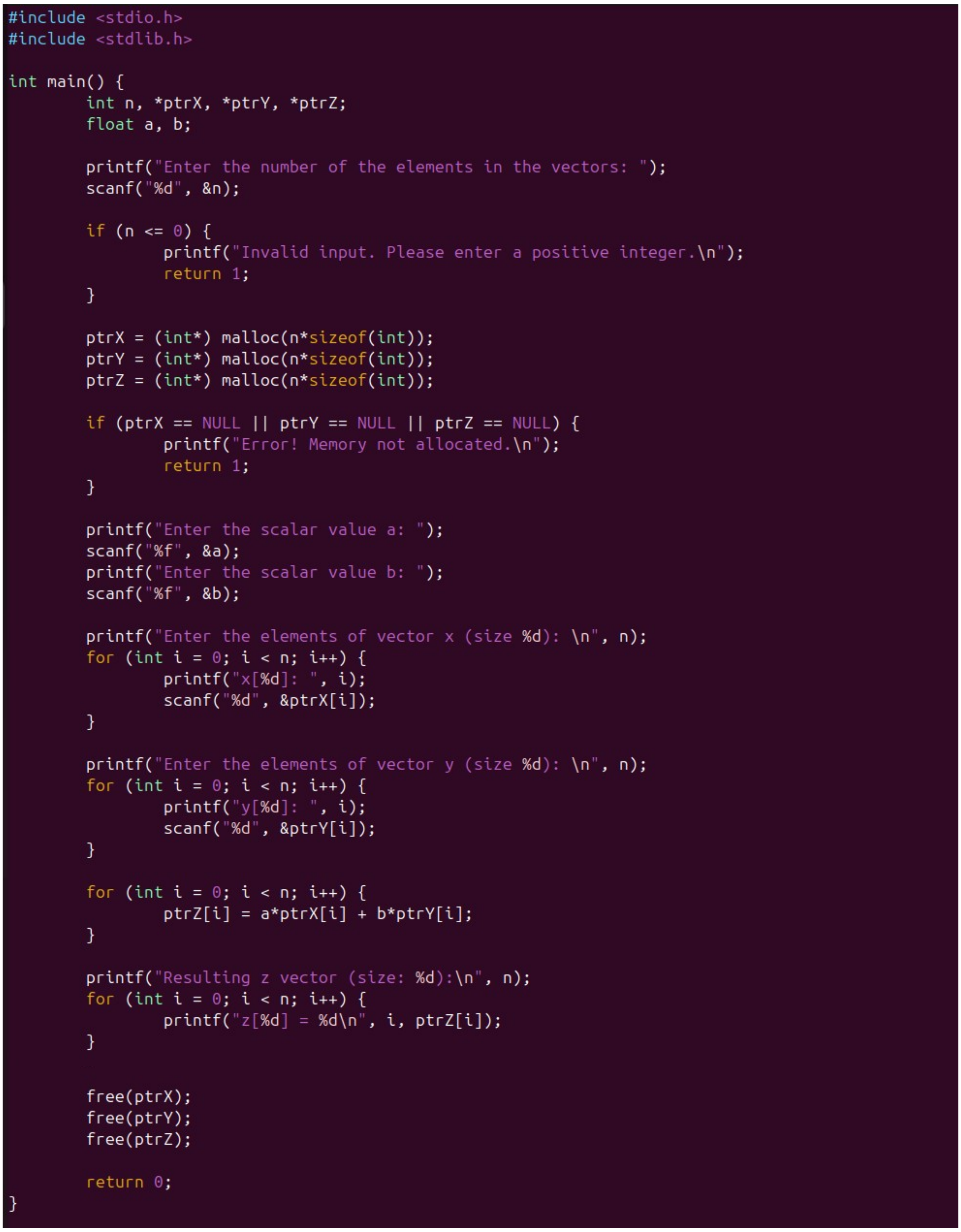
1. Prompt the user for the size of each vector.
2. Allocate memory dynamically for three integer arrays: x, y, and z.
3. Read the scalar values a and b and the vector elements for x and y

from the user.

4. Compute the operation ax + by, store the results in z, and print z



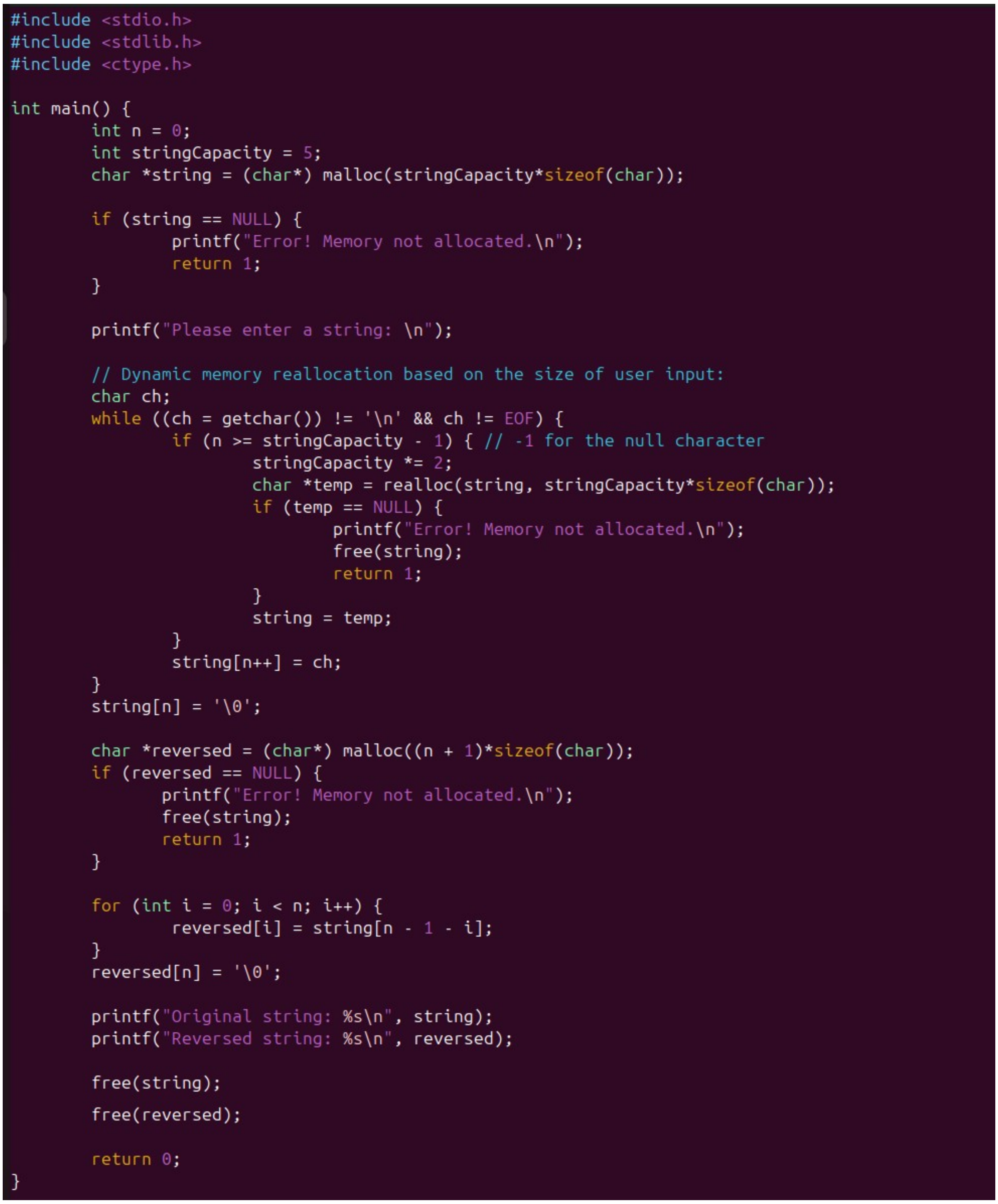
```
it@IT-RDIA-NSH:~/Lab6_C$ make run TASK=task1 FILE=task1
Compiling C files in task1...
Running task1/task1...
Enter the number of the elements in the vectors: 3
Enter the scalar value a: 2
Enter the scalar value b: 0.5
Enter the elements of vector x (size 3):
x[0]: 1
x[1]: 1
x[2]: 1
Enter the elements of vector y (size 3):
y[0]: 2
y[1]: 2
y[2]: 2
Resulting z vector (size: 3):
z[0] = 3
z[1] = 3
z[2] = 3
```



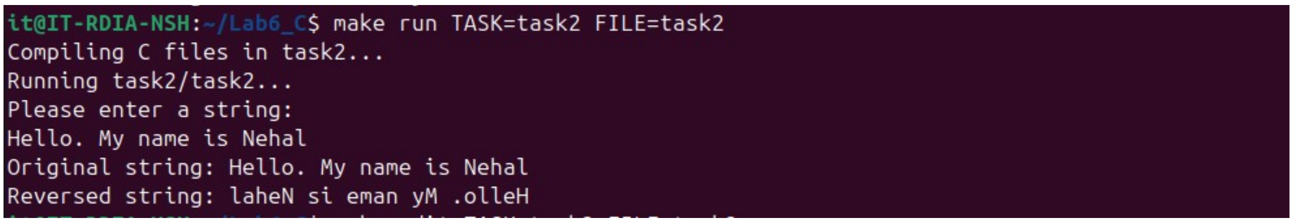
```c
#include <stdio.h>
#include <stdlib.h>

int main() {
        int n, *ptrX, *ptrY, *ptrZ;
        float a, b;

        printf("Enter the number of the elements in the vectors: ");
        scanf("%d", &n);

        if (n <= 0) {
                printf("Invalid input. Please enter a positive integer.\n");
                return 1;
        }

        ptrX = (int*) malloc(n*sizeof(int));
        ptrY = (int*) malloc(n*sizeof(int));
        ptrZ = (int*) malloc(n*sizeof(int));

        if (ptrX == NULL || ptrY == NULL || ptrZ == NULL) {
                printf("Error! Memory not allocated.\n");
                return 1;
        }

        printf("Enter the scalar value a: ");
        scanf("%f", &a);
        printf("Enter the scalar value b: ");
        scanf("%f", &b);

        printf("Enter the elements of vector x (size %d): \n", n);
        for (int i = 0; i < n; i++) {
                printf("x[%d]: ", i);
                scanf("%d", &ptrX[i]);
        }

        printf("Enter the elements of vector y (size %d): \n", n);
        for (int i = 0; i < n; i++) {
                printf("y[%d]: ", i);
                scanf("%d", &ptrY[i]);
        }

        for (int i = 0; i < n; i++) {
                ptrZ[i] = a*ptrX[i] + b*ptrY[i];
        }

        printf("Resulting z vector (size: %d):\n", n);
        for (int i = 0; i < n; i++) {
                printf("z[%d] = %d\n", i, ptrZ[i]);
        }


        free(ptrX);
        free(ptrY);
        free(ptrZ);

        return 0;
}
```

TASK 2:

Write a program that takes a string as input, dynamically allocates memory to store the reversed string, and then prints the reversed string.



```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main() {
        int n = 0;
        int stringCapacity = 5;
        char *string = (char*) malloc(stringCapacity*sizeof(char));

        if (string == NULL) {
                printf("Error! Memory not allocated.\n");
                return 1;
        }

        printf("Please enter a string: \n");

        // Dynamic memory reallocation based on the size of user input:
        char ch;
        while ((ch = getchar()) != '\n' && ch != EOF) {
                if (n >= stringCapacity - 1) { // -1 for the null character
                        stringCapacity *= 2;
                        char *temp = realloc(string, stringCapacity*sizeof(char));
                        if (temp == NULL) {
                                printf("Error! Memory not allocated.\n");
                                free(string);
                                return 1;
                        }
                        string = temp;
                }
                string[n++] = ch;
        }
        string[n] = '\0';

        char *reversed = (char*) malloc((n + 1)*sizeof(char));
        if (reversed == NULL) {
               printf("Error! Memory not allocated.\n");
               free(string);
               return 1;
        }

        for (int i = 0; i < n; i++) {
                reversed[i] = string[n - 1 - i];
        }
        reversed[n] = '\0';

        printf("Original string: %s\n", string);
        printf("Reversed string: %s\n", reversed);

        free(string);

        free(reversed);

        return 0;
}
```



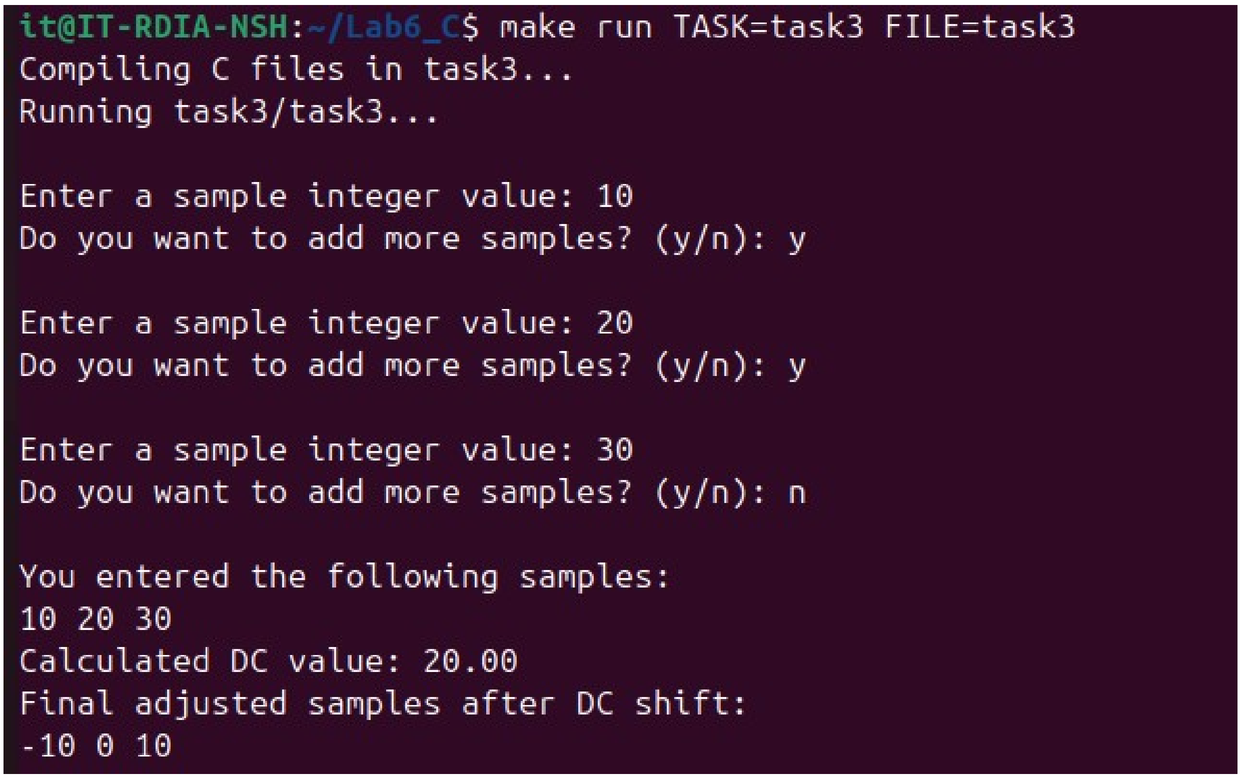
```
it@IT-RDIA-NSH:~/Lab6_C$ make run TASK=task2 FILE=task2
Compiling C files in task2...
Running task2/task2...
Please enter a string:
Hello. My name is Nehal
Original string: Hello. My name is Nehal
Reversed string: laheN si eman yM .olleH
```

TASK 3 :

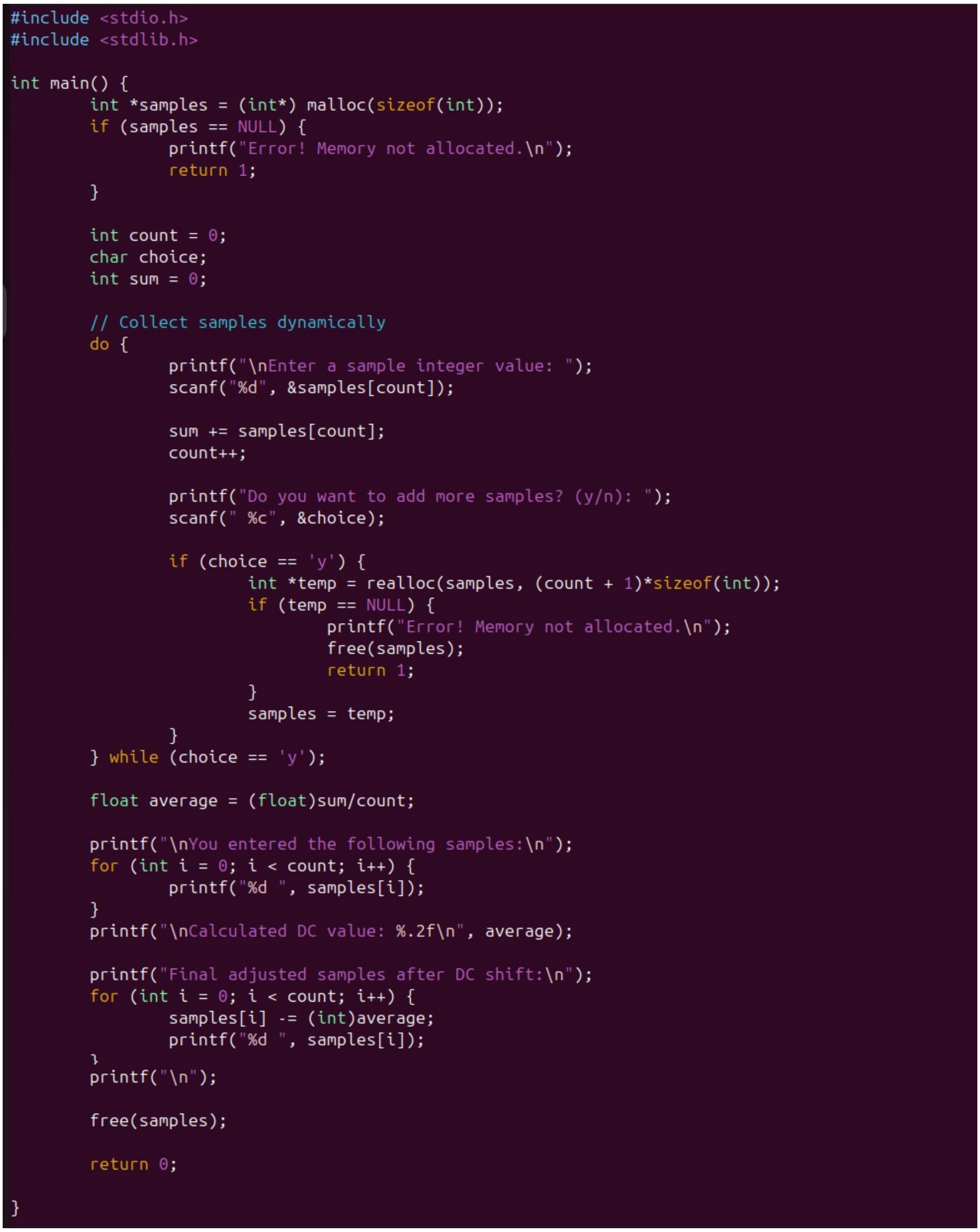
In engineering and signal processing, the concept of DC Shift (or DC Offset) is widely utilized to adjust signals by removing their average value (DC component). This task involves implementing a program to dynamically manage a sequence of integers, allowing the user to input samples until they decide to stop. The program will calculate the average of the collected samples and display the final adjusted samples after performing a DC shift.

Perform the following steps:

1. Allocate memory for one integer using malloc.
2. Prompt the user to input an integer sample.
3. Ask the user if they want to add more samples (y/n or 1/0).
4. If yes, use realloc to expand the memory for the additional integer.
5. Repeat steps 2-4 until the user decides to stop.
6. Calculate the average (DC value) of the entered samples.
7. Subtract the average from each sample to perform the DC shift.
8. Display the final adjusted samples.



```
it@IT-RDIA-NSH:~/Lab6_C$ make run TASK=task3 FILE=task3
Compiling C files in task3...
Running task3/task3...

Enter a sample integer value: 10
Do you want to add more samples? (y/n): y

Enter a sample integer value: 20
Do you want to add more samples? (y/n): y

Enter a sample integer value: 30
Do you want to add more samples? (y/n): n

You entered the following samples:
10 20 30
Calculated DC value: 20.00
Final adjusted samples after DC shift:
-10 0 10
```



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

int main() {
        int *samples = (int*) malloc(sizeof(int));
        if (samples == NULL) {
                printf("Error! Memory not allocated.\n");
                return 1;
        }

        int count = 0;
        char choice;
        int sum = 0;

        // Collect samples dynamically
        do {
                printf("\nEnter a sample integer value: ");
                scanf("%d", &samples[count]);

                sum += samples[count];
                count++;

                printf("Do you want to add more samples? (y/n): ");
                scanf(" %c", &choice);

                if (choice == 'y') {
                        int *temp = realloc(samples, (count + 1)*sizeof(int));
                        if (temp == NULL) {
                                printf("Error! Memory not allocated.\n");
                                free(samples);
                                return 1;
                        }
                        samples = temp;
                }
        } while (choice == 'y');

        float average = (float)sum/count;

        printf("\nYou entered the following samples:\n");
        for (int i = 0; i < count; i++) {
                printf("%d ", samples[i]);
        }
        printf("\nCalculated DC value: %.2f\n", average);

        printf("Final adjusted samples after DC shift:\n");
        for (int i = 0; i < count; i++) {
                samples[i] -= (int)average;
                printf("%d ", samples[i]);
        }
        printf("\n");

        free(samples);

        return 0;

}
```

TASK 4 :

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. Implement a dynamic stack using C that grows as needed and supports basic operations.

Do the following steps:

1. Start with an empty stack and allocate memory dynamically to accommodate the stack elements.

2. Prompt the user to enter a command:

● Use 'p' to push an integer (ask the user for the integer value).

● Use 'o' to pop an integer from the stack.

● Use 'd' to display the current contents of the stack.

● Use 'e' to exit the program.

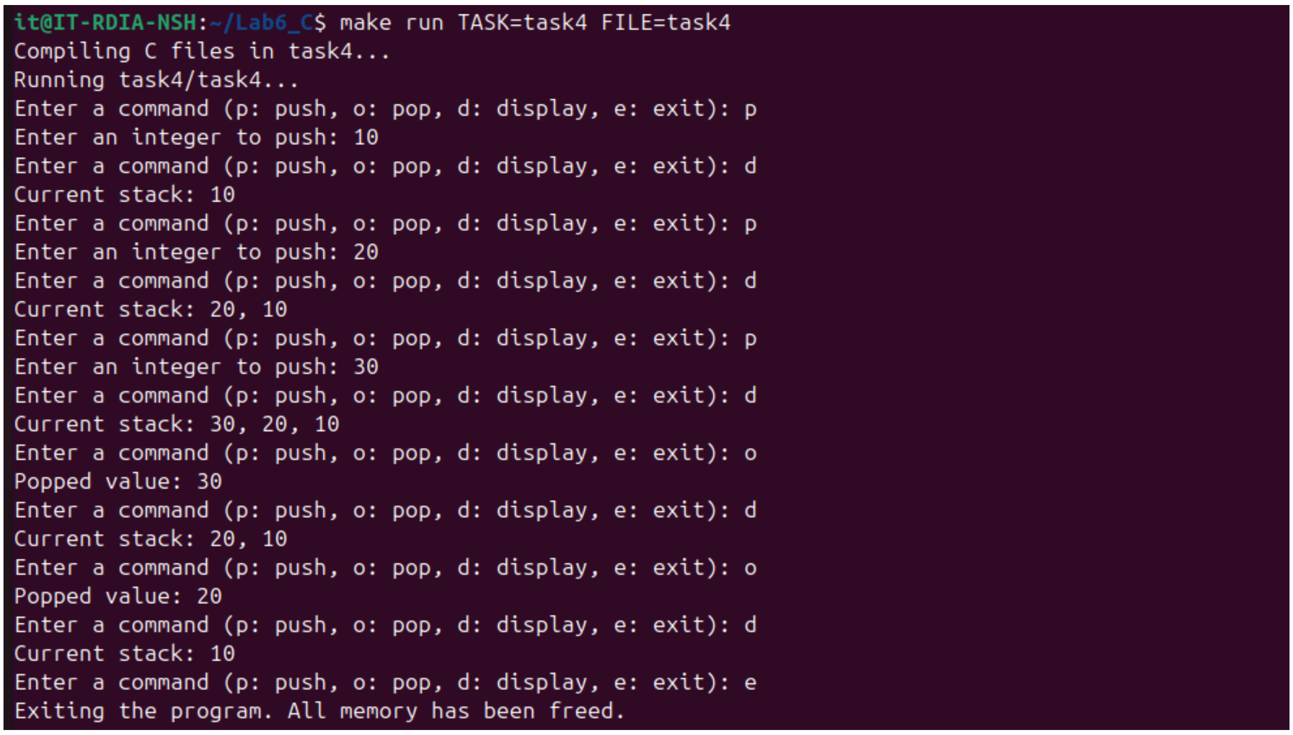
3. If the command is 'p', read an integer from the user and add it to the stack, using realloc to increase the size as needed.

4. If the command is 'o', remove the top integer from the stack, deallocating memory if necessary.

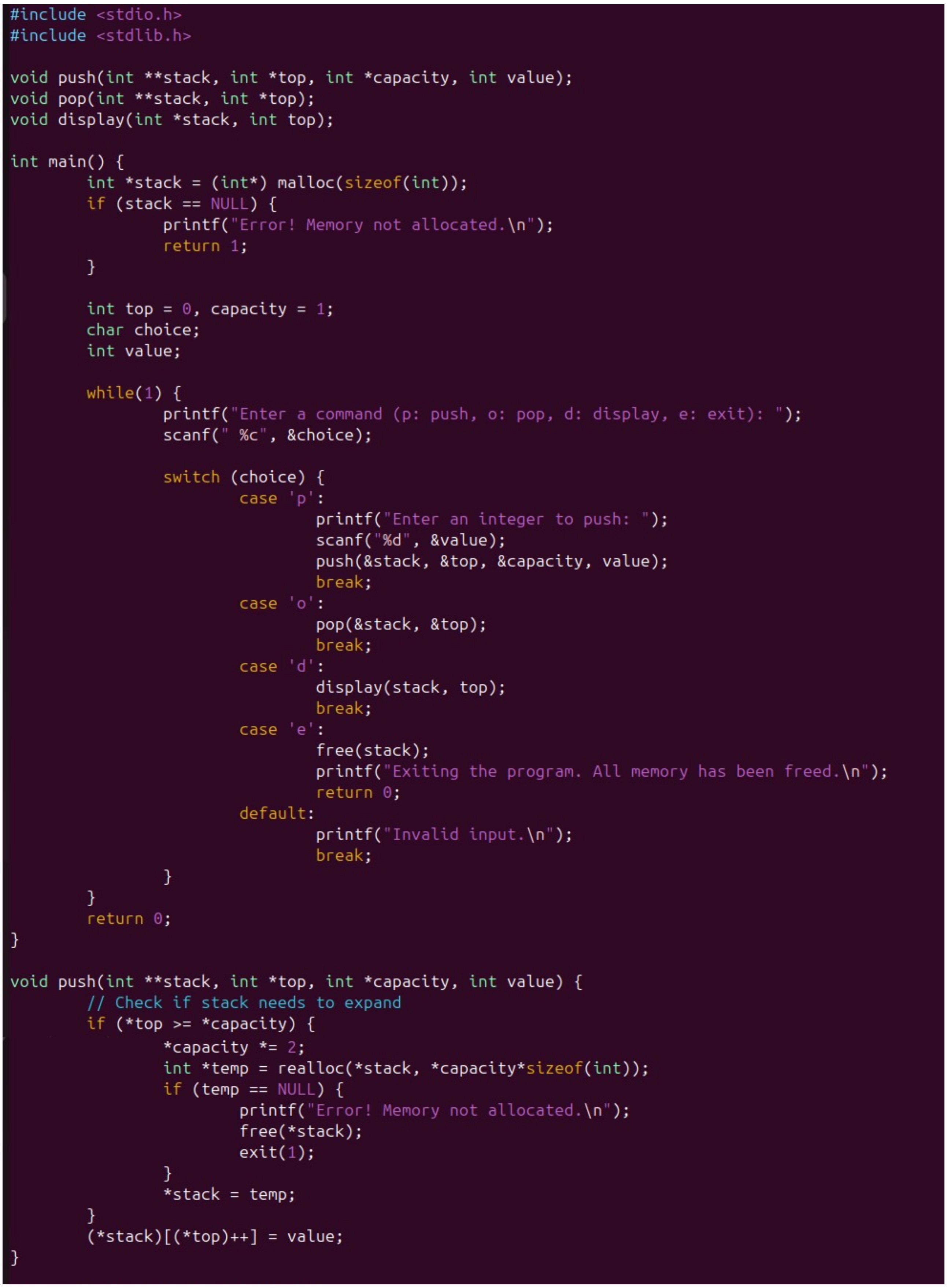
5. If the command is 'd', display the current contents of the stack

6. If the command is 'e', terminate the program cleanly, ensuring all allocated memory is properly freed.

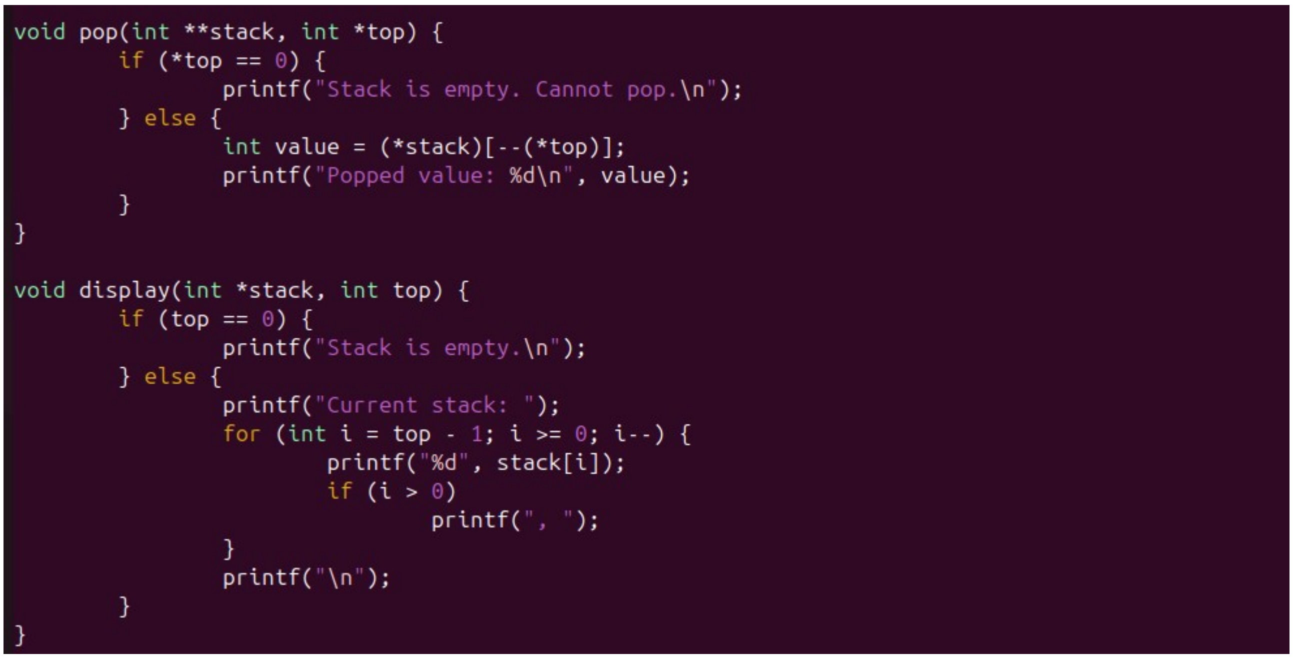
7. Treat any command that does not match 'p,' 'o,' 'd,' or 'e' as invalid and prompt the user for a valid command.



```
it@IT-RDIA-NSH:~/Lab6_C$ make run TASK=task4 FILE=task4
Compiling C files in task4...
Running task4/task4...
Enter a command (p: push, o: pop, d: display, e: exit): p
Enter an integer to push: 10
Enter a command (p: push, o: pop, d: display, e: exit): d
Current stack: 10
Enter a command (p: push, o: pop, d: display, e: exit): p
Enter an integer to push: 20
Enter a command (p: push, o: pop, d: display, e: exit): d
Current stack: 20, 10
Enter a command (p: push, o: pop, d: display, e: exit): p
Enter an integer to push: 30
Enter a command (p: push, o: pop, d: display, e: exit): d
Current stack: 30, 20, 10
Enter a command (p: push, o: pop, d: display, e: exit): o
Popped value: 30
Enter a command (p: push, o: pop, d: display, e: exit): d
Current stack: 20, 10
Enter a command (p: push, o: pop, d: display, e: exit): o
Popped value: 20
Enter a command (p: push, o: pop, d: display, e: exit): d
Current stack: 10
Enter a command (p: push, o: pop, d: display, e: exit): e
Exiting the program. All memory has been freed.
```



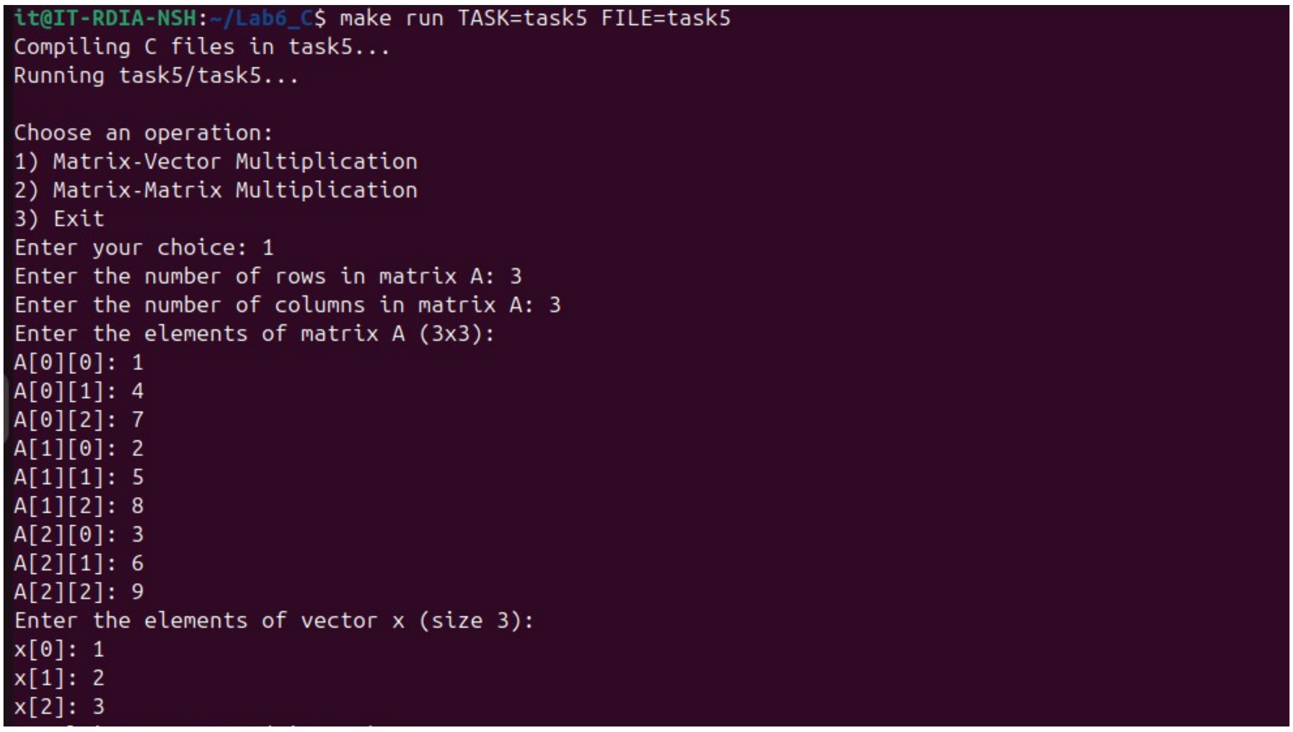
```c
#include <stdio.h>
#include <stdlib.h>

void push(int **stack, int *top, int *capacity, int value);
void pop(int **stack, int *top);
void display(int *stack, int top);

int main() {
        int *stack = (int*) malloc(sizeof(int));
        if (stack == NULL) {
                printf("Error! Memory not allocated.\n");
                return 1;
        }

        int top = 0, capacity = 1;
        char choice;
        int value;

        while(1) {
                printf("Enter a command (p: push, o: pop, d: display, e: exit): ");
                scanf(" %c", &choice);

                switch (choice) {
                        case 'p':
                                printf("Enter an integer to push: ");
                                scanf("%d", &value);
                                push(&stack, &top, &capacity, value);
                                break;
                        case 'o':
                                pop(&stack, &top);
                                break;
                        case 'd':
                                display(stack, top);
                                break;
                        case 'e':
                                free(stack);
                                printf("Exiting the program. All memory has been freed.\n");
                                return 0;
                        default:
                                printf("Invalid input.\n");
                                break;
                }
        }
        return 0;
}

void push(int **stack, int *top, int *capacity, int value) {
        // Check if stack needs to expand
        if (*top >= *capacity) {
                *capacity *= 2;
                int *temp = realloc(*stack, *capacity*sizeof(int));
                if (temp == NULL) {
                        printf("Error! Memory not allocated.\n");
                        free(*stack);
                        exit(1);
                }
                *stack = temp;
        }
        (*stack)[(*top)++] = value;
}
```



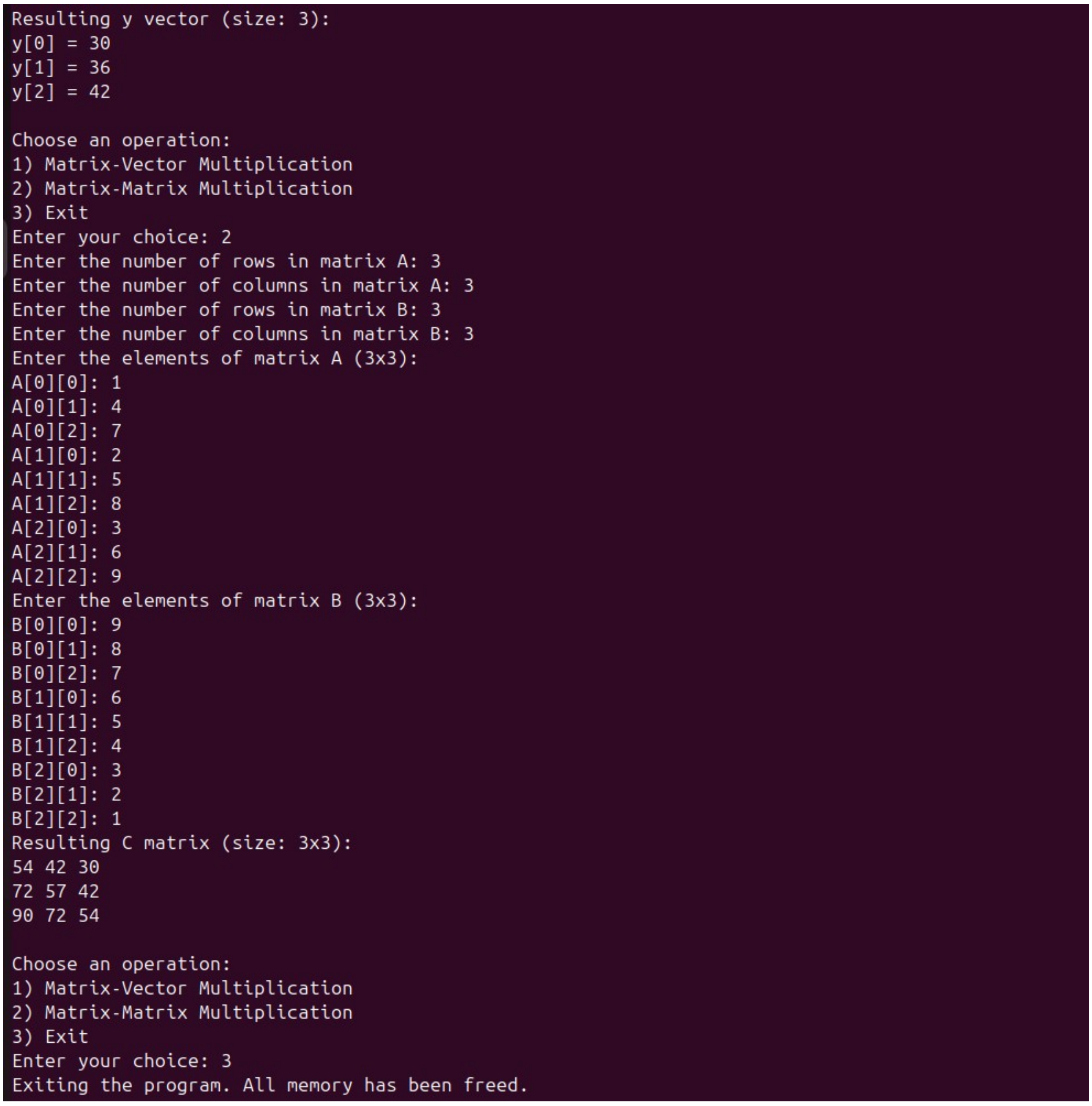
```c
void pop(int **stack, int *top) {
        if (*top == 0) {
                printf("Stack is empty. Cannot pop.\n");
        } else {
                int value = (*stack)[--(*top)];
                printf("Popped value: %d\n", value);
        }
}

void display(int *stack, int top) {
        if (top == 0) {
                printf("Stack is empty.\n");
        } else {
                printf("Current stack: ");
                for (int i = top - 1; i >= 0; i--) {
                        printf("%d", stack[i]);
                        if (i > 0)
                                printf(", ");
                }
                printf("\n");
        }
}
```

TASK 5: Dynamic Memory Allocation for Linear Algebra Operations

Write a program that performs basic linear algebra operations using dynamic memory allocation. The program should allow the user to choose from two operations: matrix-vector multiplication, and matrix-matrix multiplication. For each operation, prompt the user to input the size of vectors or matrices, dynamically allocate memory for them, and then perform the chosen operation. Provide an option to exit the program. Ensure proper handling of memory allocation and deallocation for each operation.



```
it@IT-RDIA-NSH:~/Lab6_C$ make run TASK=task5 FILE=task5
Compiling C files in task5...
Running task5/task5...

Choose an operation:
1) Matrix-Vector Multiplication
2) Matrix-Matrix Multiplication
3) Exit
Enter your choice: 1
Enter the number of rows in matrix A: 3
Enter the number of columns in matrix A: 3
Enter the elements of matrix A (3x3):
A[0][0]: 1
A[0][1]: 4
A[0][2]: 7
A[1][0]: 2
A[1][1]: 5
A[1][2]: 8
A[2][0]: 3
A[2][1]: 6
A[2][2]: 9
Enter the elements of vector x (size 3):
x[0]: 1
x[1]: 2
x[2]: 3
```



```
Resulting y vector (size: 3):
y[0] = 30
y[1] = 36
y[2] = 42

Choose an operation:
1) Matrix-Vector Multiplication
2) Matrix-Matrix Multiplication
3) Exit
Enter your choice: 2
Enter the number of rows in matrix A: 3
Enter the number of columns in matrix A: 3
Enter the number of rows in matrix B: 3
Enter the number of columns in matrix B: 3
Enter the elements of matrix A (3x3):
A[0][0]: 1
A[0][1]: 4
A[0][2]: 7
A[1][0]: 2
A[1][1]: 5
A[1][2]: 8
A[2][0]: 3
A[2][1]: 6
A[2][2]: 9
Enter the elements of matrix B (3x3):
B[0][0]: 9
B[0][1]: 8
B[0][2]: 7
B[1][0]: 6
B[1][1]: 5
B[1][2]: 4
B[2][0]: 3
B[2][1]: 2
B[2][2]: 1
Resulting C matrix (size: 3x3):
54 42 30
72 57 42
90 72 54

Choose an operation:
1) Matrix-Vector Multiplication
2) Matrix-Matrix Multiplication
3) Exit
Enter your choice: 3
Exiting the program. All memory has been freed.
```