EXPERIMENT 2

Demonstrate the use of Arrays, Pointers, dynamic memory allocation for String pattern matching. Tabulate the output for various inputs and verify against expected values. Analyze the efficiency of the algorithm. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome in conclusion. create and apply a vector ADT which uses the dynamic memory allocation concept. Use the created vector ADT for string pattern matching.

. Your report should include:

1. Introduction and Purpose of Experiment
2. Aim and Objectives
3. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
4. Implement C program
5. Presentation of Results
6. Conclusion

1. Introduction and purpose of the experiment

String operations can be said to be the backbone of user interaction. Much of the data that is acquired by the user is in the form of strings. Performing operations on these strings are of vital importance. Dynamic memory allocation is an important concept in C for allocating and releasing of the allocated memory dynamically, which gives an application a certain degree of flexibility. This experiment includes creating and applying a vector ADT which uses the dynamic memory allocation concept and using said ADT for pattern matching, a common string operation.

2. Aim and Objectives

To demonstrate string pattern matching and to apply a vector ADT for the same. Gaining knowledge about the usage and importance of ADTs is one of the objectives.

3. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code

Algorithm:

Vector.h file:

```
STEP 1: START
STEP 2: Declare struct Vector:
           int size;
           void * data;
STEP 3: Declare function prototypes
           Vector * newString(char *)
           Vector * readString()
           int cmp(Vector*, int, Vector* int)
           char * getRaw(Vector*)
           int length(Vector*)
STEP 4: STOP
```

Vector.c file:

```
STEP 1: START

STEP 2: Implement Vector * newString(char* c)

          2.1  Declare variable Vector s

          2.2  s.size ← length of c

          2.3  allocate memory to s.data

          2.4  copy string from c to s.data

STEP 3: Implement Vector * readString()

          3.1  Declare variable char * string
          3.2  string ← read input
          3.3  newString(string)

STEP 4: Implement int cmp (Vector*s1, int offset, Vector* s2, int len)

          4.1  call strncmp from <string.h>

STEP 5: Implement char * getRaw(Vector * s)

          5.1  return s.data

STEP 6: Implement int length(Vector * s)

          6.1  return s.size

STEP 7: STOP
```

Main.c

```
STEP 1: START

STEP 2: implement usingArrays()

          2.1  declare variables string, substring, flag←false, i←0

          2.2  read inputs to string, substring

          2.3  repeat until i = length of string – 1
          2.3.1 if strncmp(string+i, substring, length of substring) = 0 then
               2.3.1.1 flag ← true
               2.3.1.2 display details
     2.4 if flag = false, then
          2.4.1 display message
STEP 3: Implement usingMemAlloc()
     3.1  declare variables string, substring, flag←false, i←0
     3.2  allocate memory to string, substring
     3.3  read inputs to string, substring
     3.4  repeat until i = length of string – 1
          3.4.1 if strncmp(string+i, substring, length of substring) = 0 then
               3.4.1.1 flag ← true
               3.4.1.2 display details
     3.5 if flag = false, then
          3.5.1 display message
STEP 4: Implement usingStruct()
     4.1 Declare variables
          string ← readString()
          substring ← readString()
          flag ← false
          i ← 0
     4.2 repeat until i = length of string - 1
```

```
                4.2.1.1 if cmp(string, i, substring, length of substring) = 0 then
                     4.2.1.1 flag ← true
                     4.2.1.2 display details
           4.3 if flag = false, then
                4.3.1 display message
STEP 5: implement main()
           5.1 call usingArrays()
           5.2 call usingMemAlloc()
           5.3 call usingStruct()
STEP 6: STOP
```

Pseudocode:

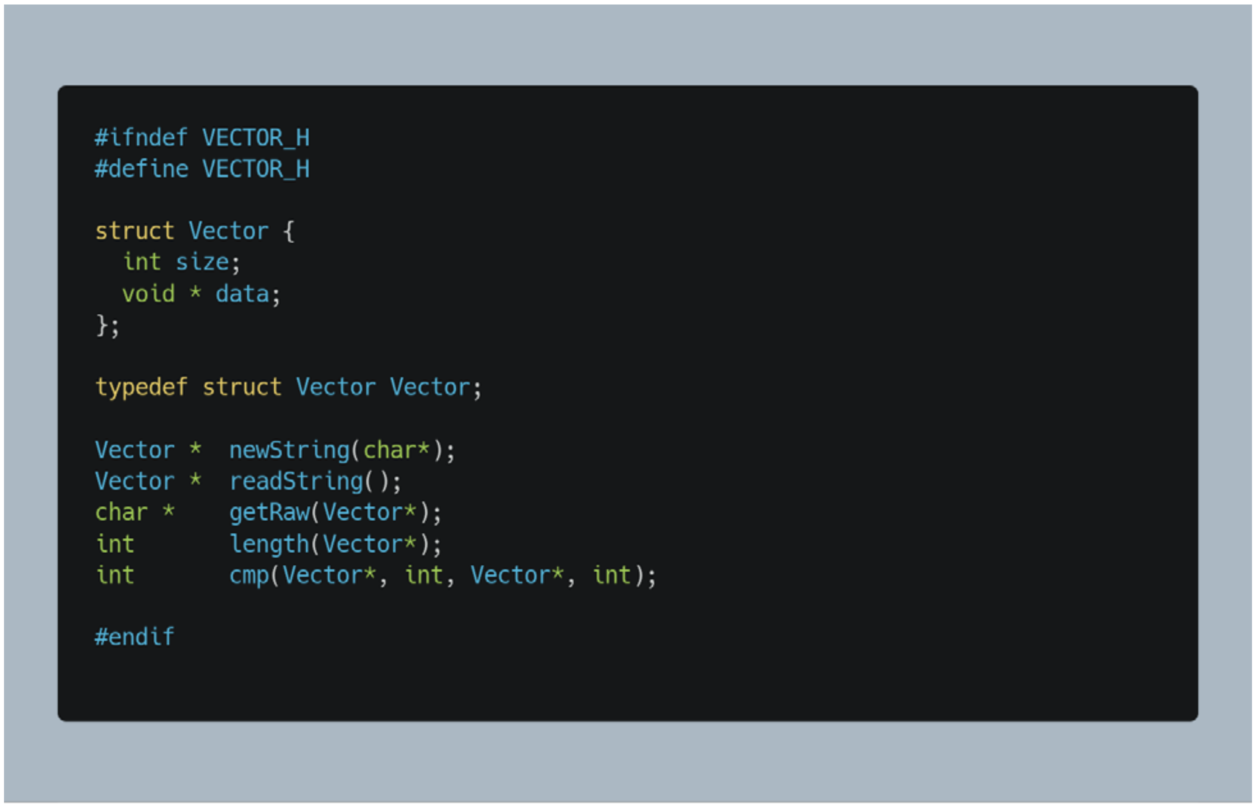
Main.c

```
Function usingArrays():
Display message
Initialize char string[100], char substring[100]
Read inputs to string, substring
Initialize flag = false, i = 0
While i++<strlen(substring)
     If strncmp(string + i, substring, strlen(substring)) == 0
          flag = true
          display details
if flag = false
```

```
     display message


Function usingMemAlloc():
Display message
Initialize char * string, char * substring
string = malloc(100)
substring = malloc(100)
Read inputs to string, substring
Initialize flag = false, i = 0
While i++<strlen(substring)
     If strncmp(string + i, substring, strlen(substring)) == 0
          flag = true
          display details
if flag = false
     display message


Function usingStruct():
Display message
Initialize Vector * string, Vector * substring to readString()
Initialize flag = false, i = 0
While i++<strlen(substring)
     If strncmp(string, i, substring, strlen(substring)) == 0
          flag = true
          display details
```

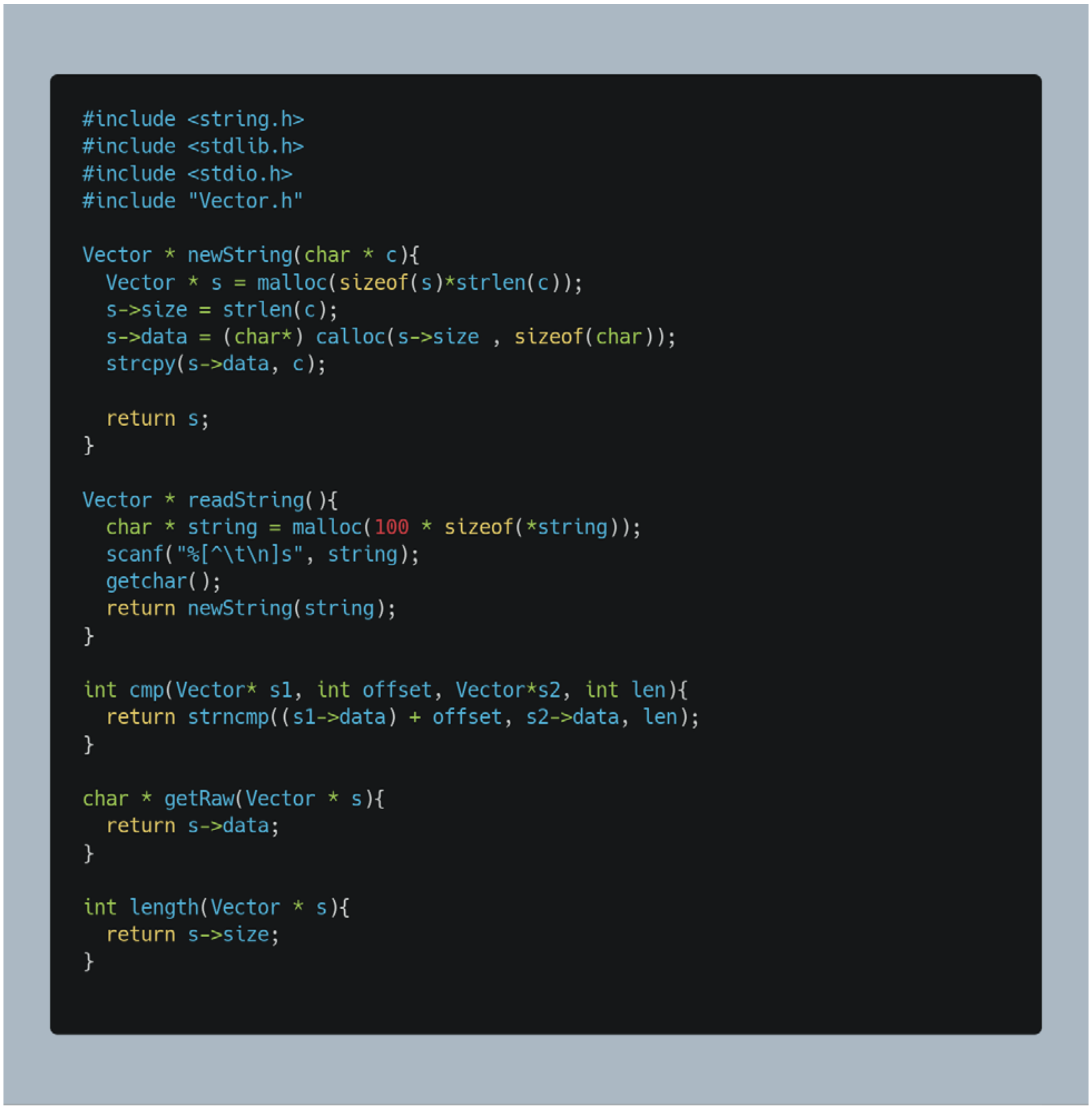
```
if flag = false
    display message
```

4. **Implement the C program**



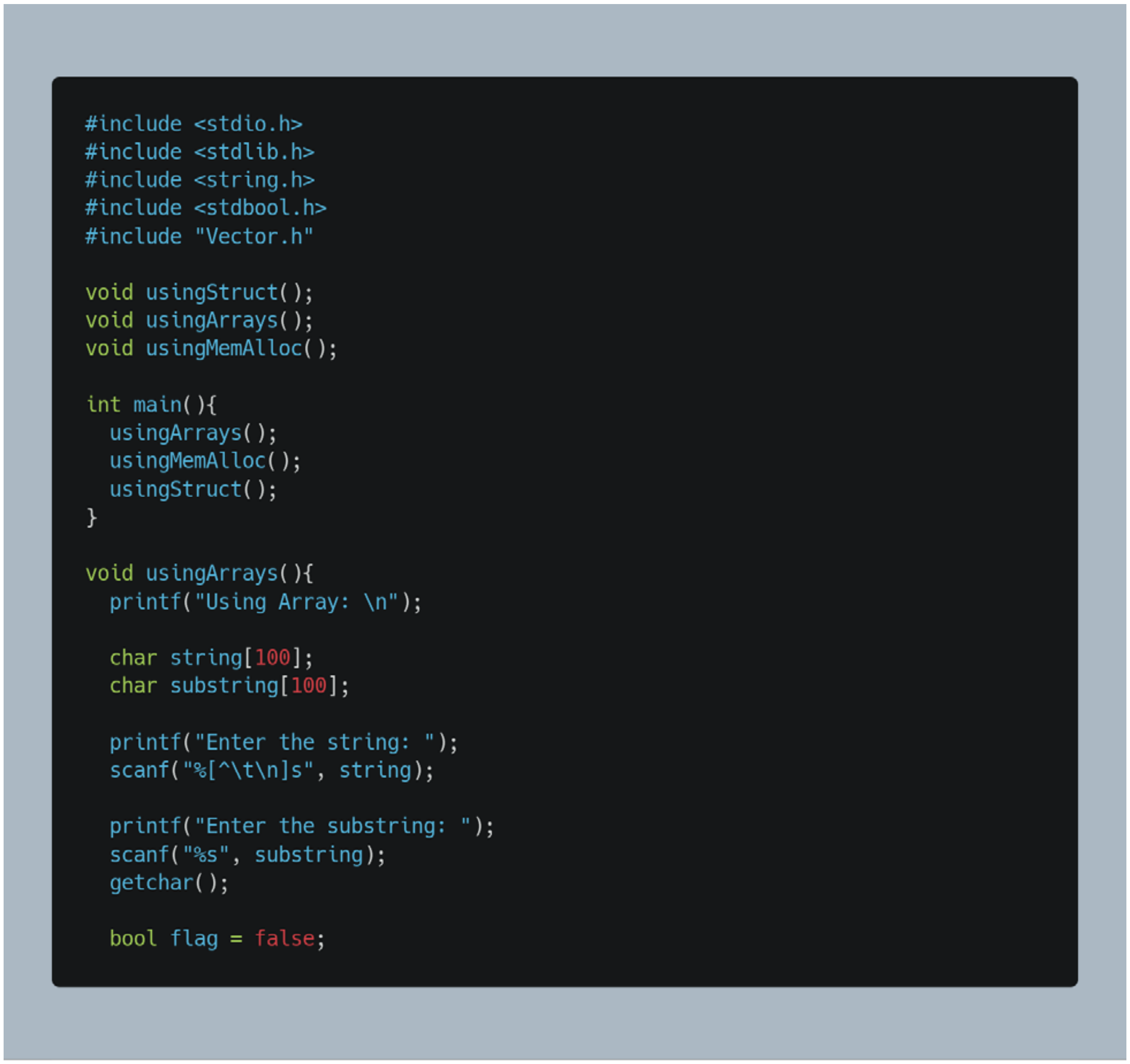
```c
#ifndef VECTOR_H
#define VECTOR_H

struct Vector {
  int size;
  void * data;
};

typedef struct Vector Vector;

Vector *  newString(char*);
Vector *  readString();
char *    getRaw(Vector*);
int       length(Vector*);
int       cmp(Vector*, int, Vector*, int);

#endif
```

Figure 1: Vector.h



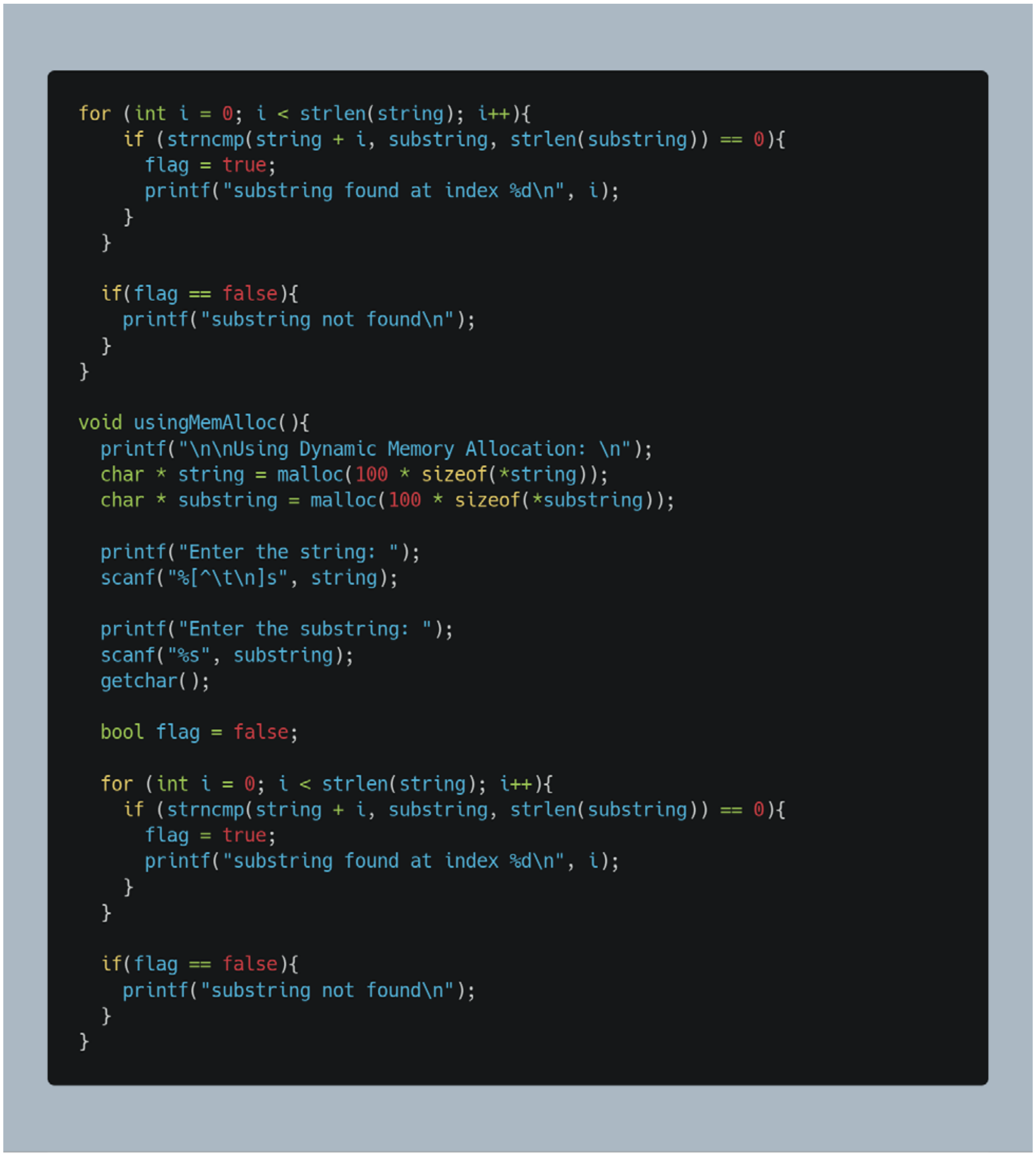
```c
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "Vector.h"

Vector * newString(char * c){
  Vector * s = malloc(sizeof(s)*strlen(c));
  s->size = strlen(c);
  s->data = (char*) calloc(s->size , sizeof(char));
  strcpy(s->data, c);

  return s;
}

Vector * readString(){
  char * string = malloc(100 * sizeof(*string));
  scanf("%[^\t\n]s", string);
  getchar();
  return newString(string);
}

int cmp(Vector* s1, int offset, Vector*s2, int len){
  return strncmp((s1->data) + offset, s2->data, len);
}

char * getRaw(Vector * s){
  return s->data;
}

int length(Vector * s){
  return s->size;
}
```

Figure 2: Vector.c



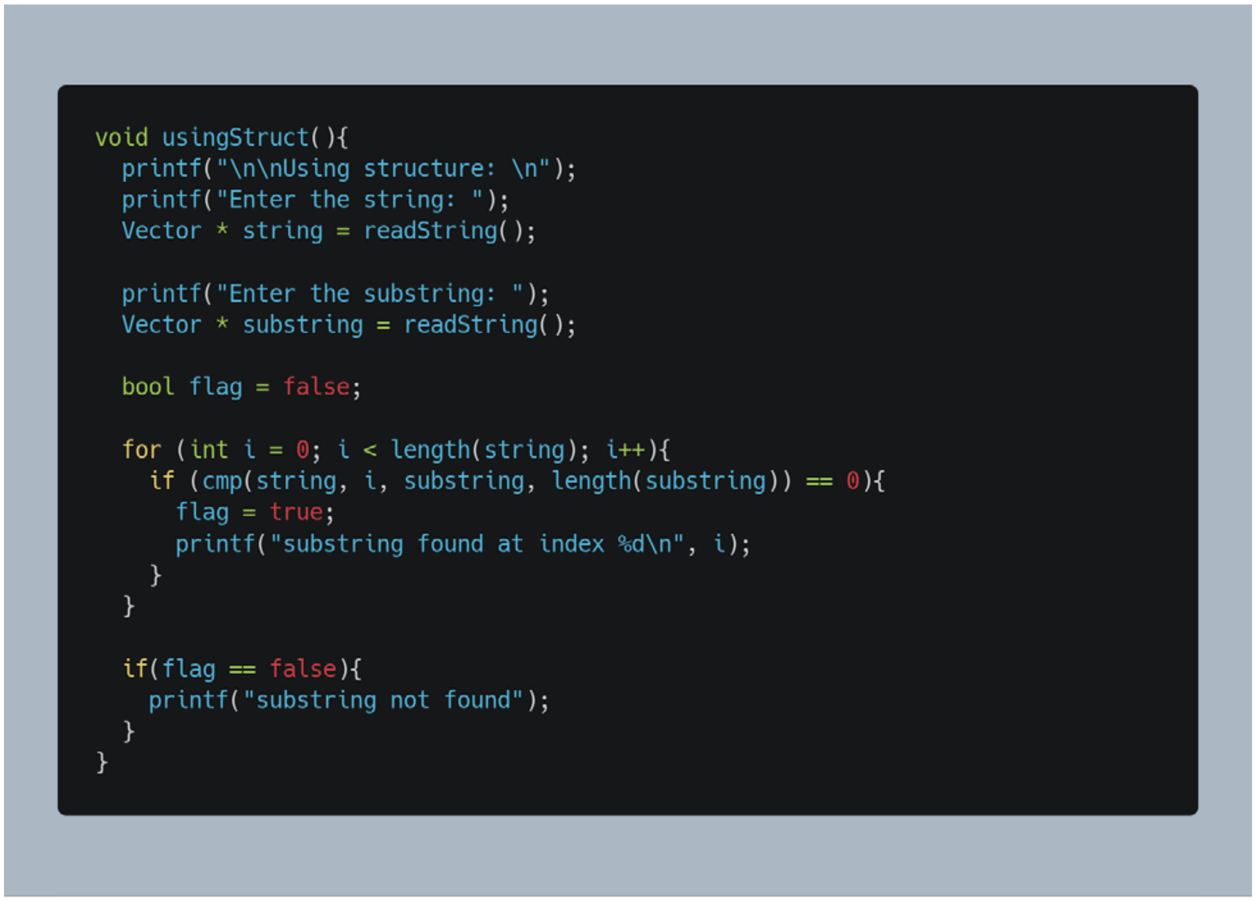
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "Vector.h"

void usingStruct();
void usingArrays();
void usingMemAlloc();

int main(){
  usingArrays();
  usingMemAlloc();
  usingStruct();
}

void usingArrays(){
  printf("Using Array: \n");

  char string[100];
  char substring[100];

  printf("Enter the string: ");
  scanf("%[^\t\n]s", string);

  printf("Enter the substring: ");
  scanf("%s", substring);
  getchar();

  bool flag = false;
```

Figure 3: Main.c (1)



```
for (int i = 0; i < strlen(string); i++){
    if (strncmp(string + i, substring, strlen(substring)) == 0){
      flag = true;
      printf("substring found at index %d\n", i);
    }
  }

  if(flag == false){
    printf("substring not found\n");
  }
}

void usingMemAlloc(){
  printf("\n\nUsing Dynamic Memory Allocation: \n");
  char * string = malloc(100 * sizeof(*string));
  char * substring = malloc(100 * sizeof(*substring));

  printf("Enter the string: ");
  scanf("%[^\t\n]s", string);

  printf("Enter the substring: ");
  scanf("%s", substring);
  getchar();

  bool flag = false;

  for (int i = 0; i < strlen(string); i++){
    if (strncmp(string + i, substring, strlen(substring)) == 0){
      flag = true;
      printf("substring found at index %d\n", i);
    }
  }

  if(flag == false){
    printf("substring not found\n");
  }
}
```

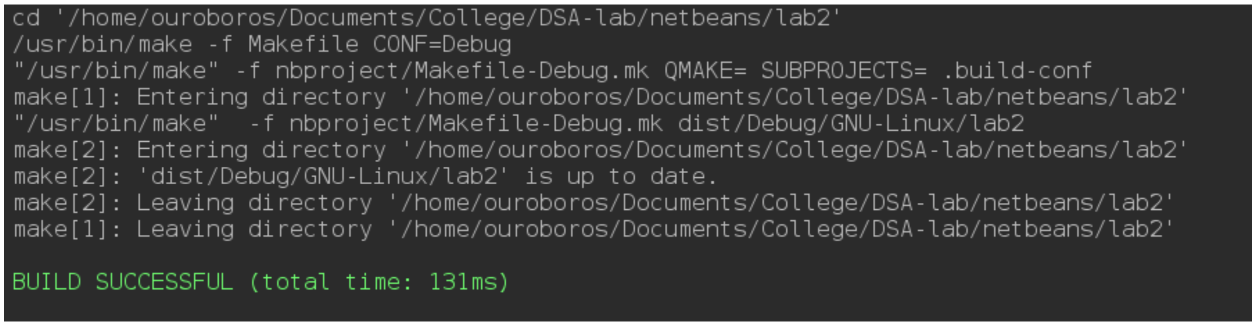
Figure 4: Main.c (2)



```c
void usingStruct(){
  printf("\n\nUsing structure: \n");
  printf("Enter the string: ");
  Vector * string = readString();

  printf("Enter the substring: ");
  Vector * substring = readString();

  bool flag = false;

  for (int i = 0; i < length(string); i++){
    if (cmp(string, i, substring, length(substring)) == 0){
      flag = true;
      printf("substring found at index %d\n", i);
    }
  }

  if(flag == false){
    printf("substring not found");
  }
}
```

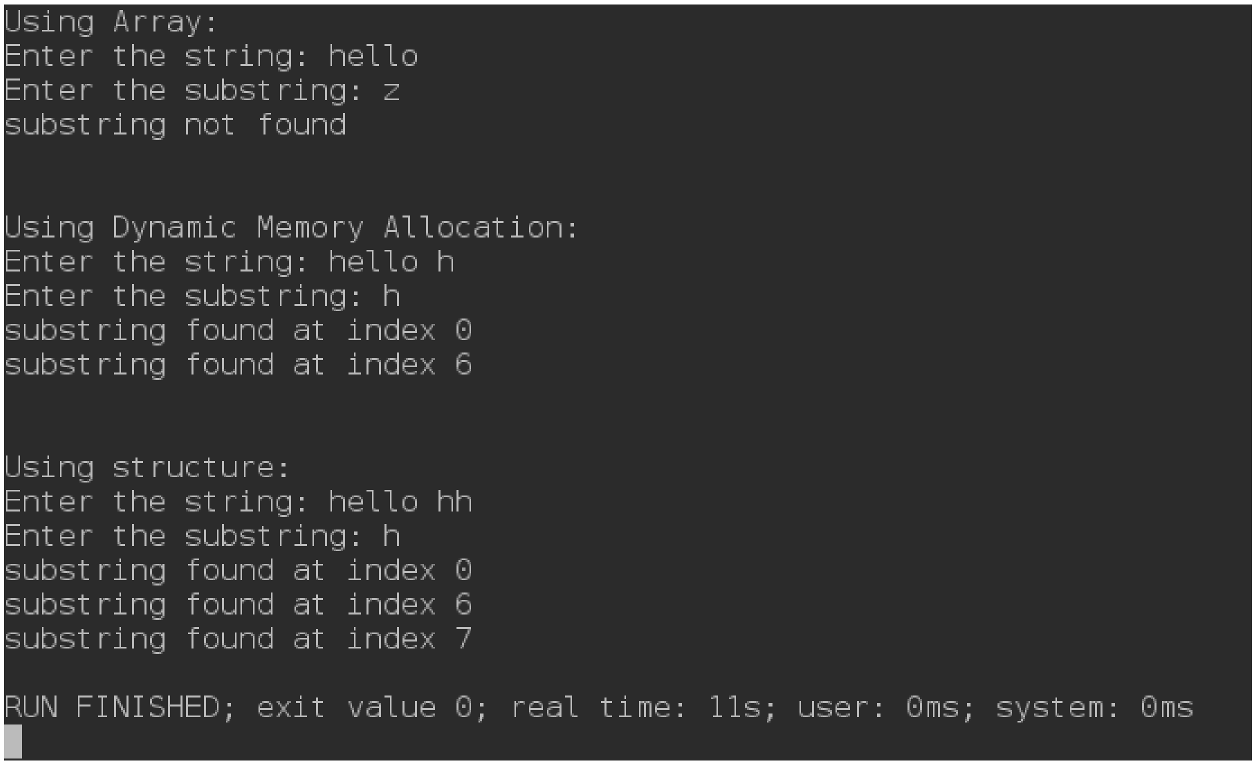
Figure 5: Main.c (3)

5. Presentation of results



```
cd '/home/ouroboros/Documents/College/DSA-lab/netbeans/lab2'
/usr/bin/make -f Makefile CONF=Debug
"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-conf
make[1]: Entering directory '/home/ouroboros/Documents/College/DSA-lab/netbeans/lab2'
"/usr/bin/make"  -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux/lab2
make[2]: Entering directory '/home/ouroboros/Documents/College/DSA-lab/netbeans/lab2'
make[2]: 'dist/Debug/GNU-Linux/lab2' is up to date.
make[2]: Leaving directory '/home/ouroboros/Documents/College/DSA-lab/netbeans/lab2'
make[1]: Leaving directory '/home/ouroboros/Documents/College/DSA-lab/netbeans/lab2'

BUILD SUCCESSFUL (total time: 131ms)
```

Figure 6: Build



```
Using Array:
Enter the string: hello
Enter the substring: z
substring not found


Using Dynamic Memory Allocation:
Enter the string: hello h
Enter the substring: h
substring found at index 0
substring found at index 6


Using structure:
Enter the string: hello hh
Enter the substring: h
substring found at index 0
substring found at index 6
substring found at index 7

RUN FINISHED; exit value 0; real time: 11s; user: 0ms; system: 0ms
```

Figure 7: Execution

6. Conclusion

Using Vector ADTs provide an efficient way in managing data, as it is a data structure with wide range of uses. The limitation of the ADT implemented in this experiment is that most of its uses are focused on strings. Despite this, there are some functions that are general, like the length function. Expanding the flexibility and usability of the ADT is a trivial task, since the new code needs to be written only in the Vector.h and Vector.c files. The implementation of the ADT is independent of the main program where it is used. This allows the ADT to be used in other programs as well. Scalability of the ADT is also no issue, which further highlights the significance of using such structures in programming.

The pattern matching program finds occurrences of a substring within a string. This is contrary to the commonly held notion of finding a word within a string. The implemented way of pattern matching is highly useful in search algorithms, where searching for an occurrence of a substring in a string or list of strings is needed.