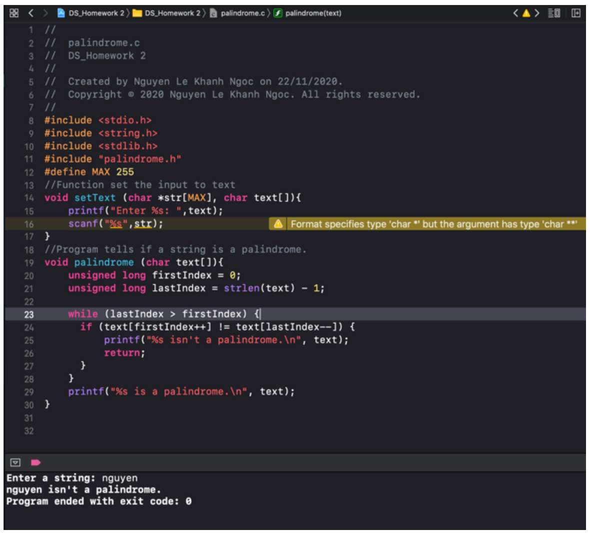
Ngoc Nguyen NITK17K - 1700062

Final job

Internal logic of the program
The Final job is done in C programming language. The logic will be explained step by step based on the order of requirements in the question.

Firstly, a simple program was made to merely solve the requirement:
- *Write a program that determines whether an input string is a palindrome; that is, whether it can be read the same way forward and backward*

```c
//Program tells if a string is a palindrome.
void palindrome (char text[]){
    unsigned long firstIndex = 0;
    unsigned long lastIndex = strlen(text) - 1;

    while (lastIndex > firstIndex) {
      if (text[firstIndex++] != text[lastIndex--]) {
          printf("%s isn't a palindrome.\n", text);
          return;
      }
    }
    printf("%s is a palindrome.\n", text);
```


DS_Homework 2 〉 DS_Homework 2 〉 palindrome.c 〉 palindrome(text)

```c
//
//  palindrome.c
//  DS_Homework 2
//
//  Created by Nguyen Le Khanh Ngoc on 22/11/2020.
//  Copyright © 2020 Nguyen Le Khanh Ngoc. All rights reserved.
//
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "palindrome.h"
#define MAX 255
//Function set the input to text
void setText (char *str[MAX], char text[]){
    printf("Enter %s: ",text);
    scanf("%s",str);
}
//Program tells if a string is a palindrome.
void palindrome (char text[]){
    unsigned long firstIndex = 0;
    unsigned long lastIndex = strlen(text) - 1;

    while (lastIndex > firstIndex) {
      if (text[firstIndex++] != text[lastIndex--]) {
          printf("%s isn't a palindrome.\n", text);
          return;
      }
    }
    printf("%s is a palindrome.\n", text);
}
```

Format specifies type 'char *' but the argument has type 'char **'

```
Enter a string: nguyen
nguyen isn't a palindrome.
Program ended with exit code: 0
```

Secondly, in order to apply stack theory, at each point, the program can read only one character of the input string; do not use an array to first store this string and then analyse it (except, possibly, in a stack implementation). Hence, I came up with the idea of using pointer variable dynamically allocate memory using malloc(), in which this pointer will hold the base address of the block created.

The program applies multiple stacks implementation with push and pop operation in stack to check whether the character is similar or not beside other function to check whether the stack is full or not.

```c
//  Copyright © 2020 Nguyen Le Khanh Ngoc. All rights reserved.
//
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifndef palindrome_h
#define palindrome_h
void readChar (char *str);        //read only one character of the input string
struct Stack;                     //structure defining Stack data structure
struct Stack* createStack(unsigned capacity);
int isFull(struct Stack* stack); //stack is full when top is equal to the last index
int isEmpty(struct Stack* stack);//stack is empty when top is equal to -1
void push(struct Stack* stack, char ele);//add an element to stack and then increment top index
char pop(struct Stack* stack);    //remove top element from stack and decrement top index
int isPalindrome(char *str);      //determine whether an input string is a palindrome
void printPalindrome (int i, char *str); //print the result
#endif /* palindrome_h */
```

By following asymptotic notations, I will determine the time complexity and space complexity of my algorithms and my overall program as below. In this program, I will use big O- notation to denote the asymptotic upper bound because it gives the worst-case running time of an algorithm.

```
O(g(n)) = { f(n): there exist positive constants c and n₀
            such that 0 ≤ f(n) ≤ cg(n) for all n ≥ n₀ }
```

- In main.c

```
int main(int argc, const char * argv[]) {
    char *str;                      //Pointer variable, this pointer will hold the
                                    //base address of the block created

    unsigned long size = 0;         //Assign to 0 to fix the compile warning

    str = (char *) malloc (size);   //Dynamically allocate memory using malloc()

    readChar(str);                  //Read only one character of the input string

    printPalindrome(isPalindrome(str), str); //Determines whether an input string is a palindrome
    return 0;
}
```

Space complexity	**Time complexity**
Variable **str* is char type and *size* is unsigned long type, hence they will take up 1 + 4 = 5 bytes. → Constant space complexity	Constant time c_1 so we have O(1).

- In palindrome.c

```
void readChar (char *str){
    int i;
    printf("Enter a string: ");
    i = 0;
    char ch = getchar ();
    while(ch!='\n') {               //get the string from the user by
        str[i] = ch;                //using only single char variable in a loop
        i++;
        ch = getchar();
    }
    printf("You entered the following string: %s.\n",str);
}
```

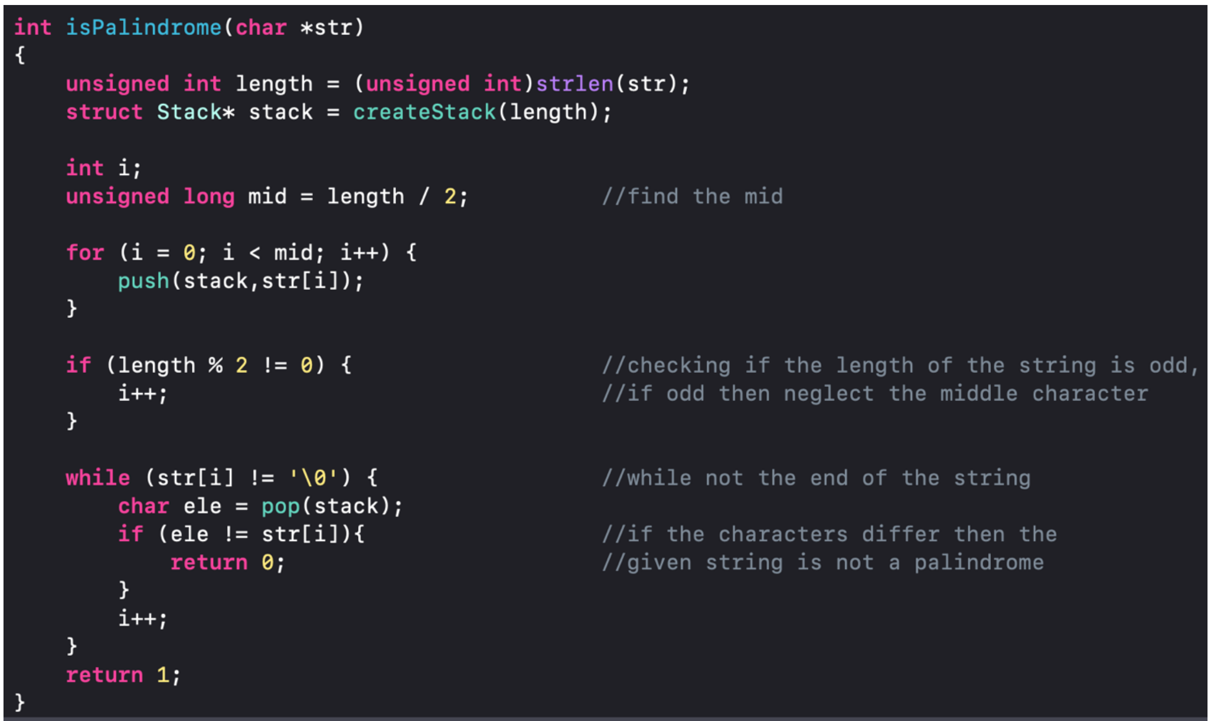
Space complexity	**Time complexity**
Variable *ch* is char type and *i* is integer type, hence they will take up 1 + 4 = 5 bytes. 1*n bytes of space is required for the pointer *str elements. Hence the total memory requirement will be (n + 5), which is increasing linearly with the increase in the input value n. → Linear space complexity	Statements outside the while loop: $c_1 + c_2$ Statements inside the while loop take $c_3n + c_4n + c_5n$ Hence, we have $c_1 + c_2 + c_3n + c_4n + c_5n$. → O (n)

```c
struct Stack                        //structure defining Stack data structure
{
    int top;
    unsigned capacity;
    char *ptr;
};

struct Stack* createStack(unsigned capacity)
{
    struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;                //initializes the top index to -1
    stack->ptr = (char*) malloc(stack->capacity * sizeof(char));
    return stack;
}
```

Space complexity	**Time complexity**
Variable **ptr* is char type and *top, capacity* is integer and unsigned type, hence they will take up 1 + 4 + 4 = 9 bytes. Hence the total memory requirement will be (9 + 9) = 18 bytes, this additional 9 bytes is for return value.	O(1)

```c
int isFull(struct Stack* stack){    //stack is full when top is equal to the last index
    if(stack->top == stack->capacity - 1){
        return 1;
    } else {
        return 0;
    }
}

int isEmpty(struct Stack* stack){   //stack is empty when top is equal to -1
    if(stack->top == -1){
        return 1;
    } else {
        return 0;
    }
}
```

Space complexity	**Time complexity**
The return values are all integer types, hence they will take up 4 bytes each, so total memory requirement will be (4 + 4) = 8 bytes.	O(1) + O(1) = O(2)



```c
int isPalindrome(char *str)
{
    unsigned int length = (unsigned int)strlen(str);
    struct Stack* stack = createStack(length);

    int i;
    unsigned long mid = length / 2;          //find the mid

    for (i = 0; i < mid; i++) {
        push(stack,str[i]);
    }

    if (length % 2 != 0) {                   //checking if the length of the string is odd,
        i++;                                 //if odd then neglect the middle character
    }

    while (str[i] != '\0') {                 //while not the end of the string
        char ele = pop(stack);
        if (ele != str[i]){                  //if the characters differ then the
            return 0;                        //given string is not a palindrome
        }
        i++;
    }
    return 1;
}
```

Space complexity	**Time complexity**
Variable *ele* is char type and *length, i, mid* is integer and unsigned long/int type. Hence the total memory requirement will be 1 + 4 + 4 + 4 + 4 = 13 bytes, this additional 4 bytes is for return value.	Statement *length, stack*: c + c Statement *mid*: c *For loop*: n/2*c *If*: c *While loop*: n*c + c + c Whole function: c Hence, we have O(n)

Total space complexity = O(n)
Total time complexity = O(n)