АКМИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»
Тема: Программирование рекурсивных алгоритмов

Студент гр. 9303 _____ Халилов Ш.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Реализовать рекурсивный алгоритм для нахождения самой длинной подстрока палиндрома

Задание.

Реализовать рекурсивно функцию, определяющую для заданной строки длину самой длинной подстроки, являющейся палиндромом.

Основные теоретические положения.

Рекурсивный - это объект, который содержит сам себя или определяется с помощью самого себя. Сила рекурсии заключается в том, что она позволяет вам определять бесконечное количество объектов с помощью конечного оператора. Точно так же бесконечные вычисления можно описать с помощью конечной рекурсивной программы. Рекурсивные алгоритмы лучше всего использовать, когда решаемая проблема, вычисляемая функция или обрабатываемая структура данных определяются с помощью рекурсии. Если процедура (функция) P содержит явный вызов самой себя, она называется непосредственно рекурсивной. Если P содержит вызов процедуры (функции) Q, которая содержит (прямо или косвенно) вызов P, то P называется косвенно рекурсивным. Многие известные функции можно определить рекурсивно. Например, факториал, который присутствует почти во всех учебниках программирования, а также наибольший общий делитель, числа Фибоначчи, степенная функция и т. Д.

Выполнение работы.

Для реализации поиска по длине полииндрома под строкой были реализованы следующие функции:

void checkForPalindrome (строка и слово, int и слева, int и right, int couter) и void LongestPalindrome (строка и слово, индекс int, палиндром PalPos). Файл будет

указан при запуске программы в виде ключей, если ключей нет, программа предлагает прочитать слова из консоли. И он начинает читать слова из консоли или файла. после этого вызывается функция LongestPalindrome (). все аргументы передаются в виде ссылки, потому что функция ничего не возвращает, а просто записывает в память из ссылки.

Функция checkForPalindrome () получает ссылку на слова и два числа. Во-первых, эти числа являются буквенным индексом. функция сравнивает две соседние буквы, и если они одинаковые, то левая уменьшается, а правая увеличивается на 1 и вызывает себя, пока не найдет неподходящие буквы.

Функция LongestPalindrome () принимает ссылки на слова и структуры для сохранения положения полииндрома. Функция для каждой буквы из слова вызывает функцию checkForPalindrome () со ссылкой на слово и два числа, равных индексу. После завершения функций checkForPalindrome разница между двумя числами будет сравниваться с размером предыдущего полииндрома, если он больше, новая позиция будет сохранена на месте старой.

Пример работи.

Таблица №1

№	ввод	выход
1	banana	banana [1:5] ---> b|anana|
2	anna	anna [0:3] ---> |anna|
3	ananas	ananas [0:4] ---> |anana|s
4	rotatoradars	rotatoradars [0:6] ---> |rotator|adars
5	polindrom	polindrom [-:-] ---> Palindrome not found!!!
6	anavolimilovana	anavolimilovana [0:14] ---> |anavolimilovana|
7	signalwow	signalwow [6:8] ---> signal|wow|

Выводы.

Реализована программа, включающая рекурсивные функции для нахождения длины полииндрома под строкой. Программа тоже была протестирована. Были выполнены следующие требования: сообщения о начале и конце вызова функции отображаются с отступами, соответствующими глубине рекурсии, результату работы программы, всем сообщениям в консоль и результату в файл, если файл есть.

Приложения А

```
#include <iostream>
#include <fstream>

using namespace std;

typedef struct Palindrome_Poistion {

  int start, end, length;
} PalPos ;

void checkForPalindrome (string & word, int &left, int &right, int couter) {
    cout << "   [";
    for (int i = 0; i <= couter; i++)
        cout << "!";

    if (left > 0 && right < (int)word.length() && word[left - 1] == word[right + 1])
    {
        left--;
```

```
        right++;

        cout << "]  call  CP( ... , " << left << ", " << right << " ) --> [ ";
        for (int i = left; i <= right; i++)
                cout << word[i];

        cout<< " ]\n";
        checkForPalindrome (word, left, right, couter+1);
    }
    else {

        cout << "]  call  CP( ... , " << left << ", " << right << " ) --> [ ";
            for (int i = left-1; i <= right+1; i++)
                    cout << word[i];
            cout<< " ]\n";
    }



    cout << "   [";
    for (int i = 0; i <= couter; i++)
        cout << "!";
    cout << "]  end\n";


  return;
}


void LongestPalindrome (string &word, int index, PalPos &palindrome)
{
  if (index < (int)word.length() )
    {
        int left = index, right = index;
```

```
if (word[left] == word[right + 1]){

   cout << "\n";
         right++;
         checkForPalindrome (word, left, right, 0);
}
else {

   cout << "\n";
   checkForPalindrome (word, left, right, 0);
}

if (right - left > palindrome.length) {

         palindrome.start = left;
         palindrome.end = right;
         palindrome.length = palindrome.end - palindrome.start;
       }


cout << "\n[";
for (int i = 0; i <= index; i++)
   cout << "!";


cout << "]  call  PL( ... , "
    << index
    <<", { "
    << palindrome.start
    << ", "
    << palindrome.end
    << ", "
```

```
            << palindrome.length
            << "} )\n";

        LongestPalindrome (word, index+1, palindrome);
    }

    cout << "[";
    for (int i = 0; i <= index; i++)
        cout << "!";

    cout << "]   end\n";
    return;
}


int main (int argc, char *argv[] ) {



    string input_word;
    PalPos palindrome;

    if( argc > 1 ) {

        ifstream input_file(argv[1]);
        ofstream output_file( argv[2] );

        while (input_file >> input_word){

            cout << "\n[=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=]\n|\t\t"
                << input_word
                <<"\n[=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=]\n";
            palindrome = { 0, 0, 0 };
            LongestPalindrome (input_word, 0, palindrome);
```

```
                if( palindrome.length >1 ) {
output_file << input_word << " ["<< palindrome.start << ":" << palindrome.end <<"] ---> ";
cout <<"\n[ Result ]\n\n    "<< input_word << " ["<< palindrome.start << ":" << palindrome.end <<"]
---> ";
                    for (int i = 0; i < palindrome.start; i++)
                    {
                       output_file << input_word[i];
                       cout << input_word[i];
                    }
                    output_file <<"|";
                    cout <<"|";
                    for (int i =palindrome.start; i <= palindrome.end; i++ ) {

                       output_file << input_word[i];
                       cout << input_word[i];
                    }

                    output_file <<"|";
                    cout <<"|";

                    for (int i = palindrome.end+1; i < (int)input_word.length(); i++)
                    {
                       output_file << input_word[i];
                       cout << input_word[i];
                    }
                    output_file << "\n\n";
                    cout << "\n\n[ Search completed ]\n\n";
                 }
                 else
                 {
                    output_file << input_word << " [-:-] ---> Palindrome not found!!!\n\n";
                    cout << input_word << " [-:-] ---> Palindrome not found !!!\n\n";
                 }
```

```
        }
        output_file.close();
        input_file.close();
    }
    else
    {
       palindrome = { 0, 0, 0 };
        cin >> input_word;

        LongestPalindrome (input_word, 0, palindrome);

            for (int i =palindrome.start; i <=palindrome.end; i++)
            {
                cout << input_word[i];
            }
            cout << "\n";
    }

    cout << "[ Prosses Finshed ] \n> Goodbye!\n";
  return 0;
}
```