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

ОТЧЕТ

по лабораторной работе № 2
по дисциплине «Алгоритмы и структуры данных»
Тема: Иерархические списки

Студент гр. 9304 _____ Мохаммед А.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы

Ознакомиться с иерархическими списками, изучить применение иерархических списков на практике.

Основные теоретические положения

Бинарное коромысло устроено так, что у него есть два плеча: левое и правое. Каждое плечо представляет собой невесомый стержень определенной длины, с которого свисает либо гирька, либо еще одно бинарное коромысло, устроенное таким же образом. Возможный способ представления бинарного коромысла представлен на рис. 1.

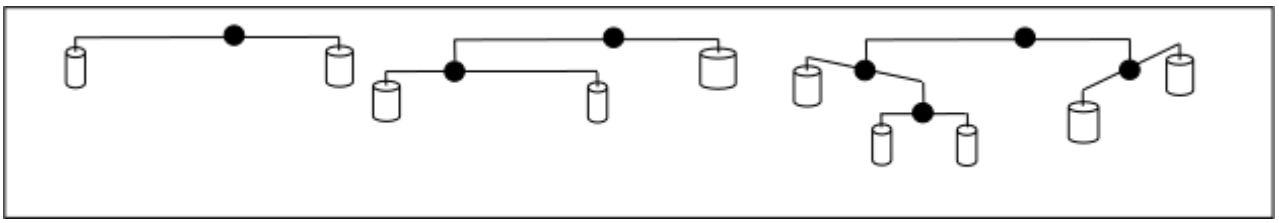


Рисунок 1 — Способ представления бинарного коромыслаВ соответствии с данным определением бинарного коромысла представим бинарное коромысло (БинКор) списком из двух элементов:

БинКор::=(Плечо Плечо),

где первое плечо является левым, а второе — правым. В свою очередь плечо будет представляться списком из двух элементов:

Плечо::=(Длина Груз),

где длина есть натуральное число, а груз представляется вариантами

Груз::=Гирька | БинКор,

где в свою очередь гирька есть натуральное число. Таким образом, бинарное коромысло — это специального вида иерархический список из натуральных чисел.

Постановка задачи

Подсчитать общую длину всех плеч заданного бинарного коромысла bk. Для этого ввести рекурсивную функцию

short length(const БинКор bk).

Выполнение работы

Программа написана на языке программирования C++.

В начале работы происходит ввод данных. Выбор способа ввода данных предоставляется пользователю. Вводится строка, которая является бинарным коромыслом, после чего происходит обработка строки и создание бинарного коромысла, для чего вызывается функция `createBinCor()`. Если введенные данные некорректны, то программа выводит соответствующую ошибку и завершает работу. В случае ввода корректных данных происходит дальнейший вызов функции `length()`, которая выводит на экран общую длину всех плеч бинарного коромысла.

Текст программы представлен в приложении Б.

Описание функций

1. `int createBinCor(char *str, BinCor **binCor)`

Функция принимает указатель на строку, в которую записано бинарное коромысло и указатель на структуру `BinCor` (бинарное коромысло). Функция устанавливает начальный и конечный индексы на первый и последний символы полученной строки, после чего передает эти данные функции `readBinCorElement()`, которая обрабатывает строку и создает бинарное коромысло.

Параметры:

`str:` указатель на строку с бинарным коромыслом;

`binCor:` указатель на указатель на структуру данных `BinCor`, содержащий адрес, куда будет заноситься результат обработки строки `str.`

Возвращаемое значение: 0 — если данные в строке `str` корректны, в ином случае — 1.

2. `int readBinCorElement(char *str, int index_1, int index_2, BinCor **element)`

Функция принимает указатель на строку, в которую записано бинарное коромысло, начальный и конечный индекс на первый и последний символы строки `str,` между которыми записано обрабатываемое на данном шаге

бинарное коромысло и указатель на указатель на структуру BinCor (бинарное коромысло). Функция посимвольно обрабатывает два плеча бинарного коромысла, внося все данные по адресу, содержащемся в указателе, на который указывает element. Если груз плеча является еще одним бинарным коромыслом, то происходит рекурсивный вызов функции. При возникновении в какой-то момент ошибки вызывается функция errorMassage(), которая выводит на экран сообщение об ошибке, после чего функция завершает свою работу.

Параметры:

str: указатель на строку с бинарным коромыслом;

index_1: индекс, с которого надо начать обработку строки str;

index_2: индекс, на котором нужно закончить обработку строки str;

binCor: указатель на указатель на структуру данных BinCor, содержащий адрес, куда будет заносится результат обработки строки str.

Возвращаемое значение: 0 — если данные в строке str корректны, в ином случае — 1.

3. short length(const BinCor binCor, int deep_of_recursion)

Функция принимает структуру данных BinCor и целочисленное значение, указывающее на глубину рекурсии. Функция считает длину всех плеч в бинарном коромысле binCor.

Параметры:

binCor: бинарное коромысло, в котором считается длина плеч;

deep_of_recursion: глубина рекурсии (необходимо для вывода работы алгоритма на экран).

Возвращаемое значение: длина всех плеч в бинарном коромысле.

Тестирование программы

Было создано несколько тестов для проверки работы программы. Помимо тестов, демонстрирующих работу алгоритма, были написаны тесты, содержащие некорректные входные данные, для демонстрации вывода сообщений об ошибках введенных данных. Тестовые случаи представлены в приложении А.

Выводы

В ходе выполнения работы была изучена новая структура данных: иерархические списки. Так же были закреплены навыки работы с рекурсивными функциями.

ПРИЛОЖЕНИЕ А

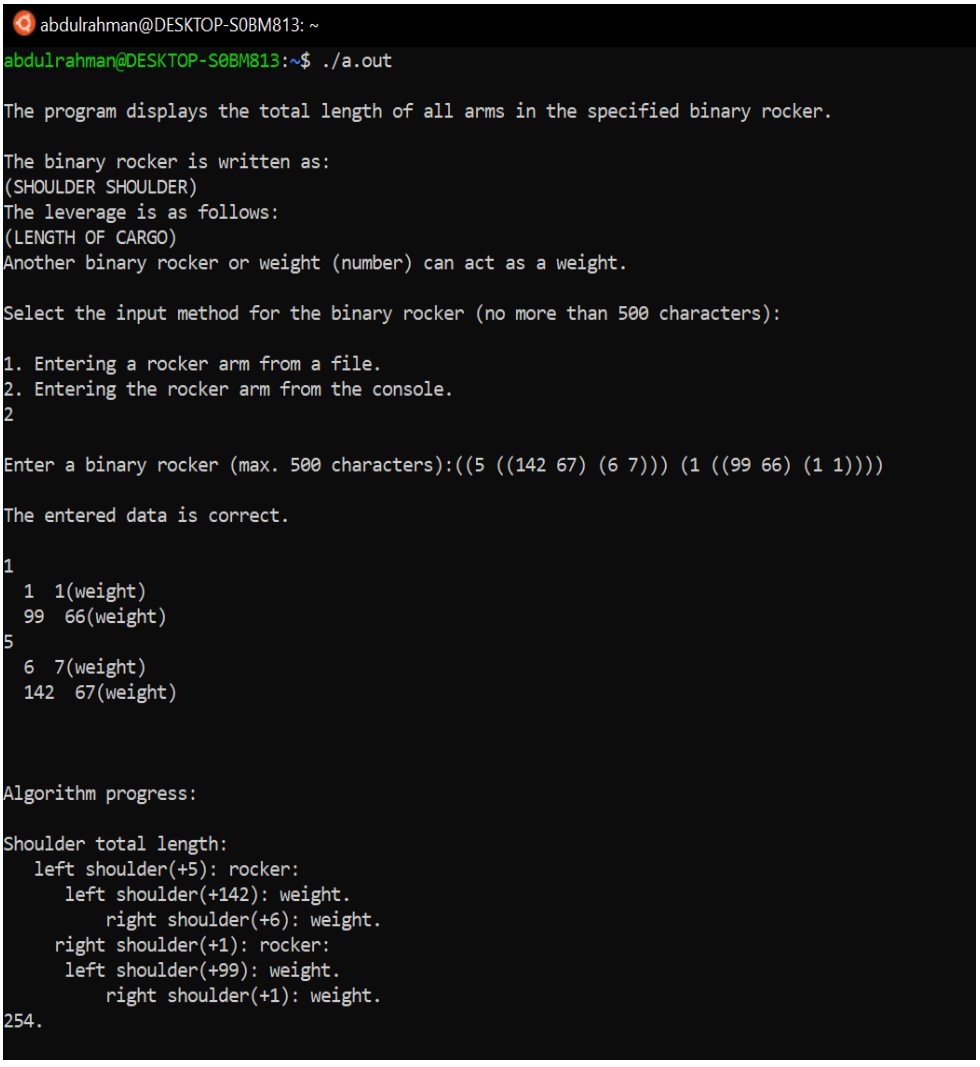
Тестирование

Демонстрация работы:

Входные данные:

((5 ((142 67) (6 7))) (1 ((99 66) (1 1))))

Результат работы программы:

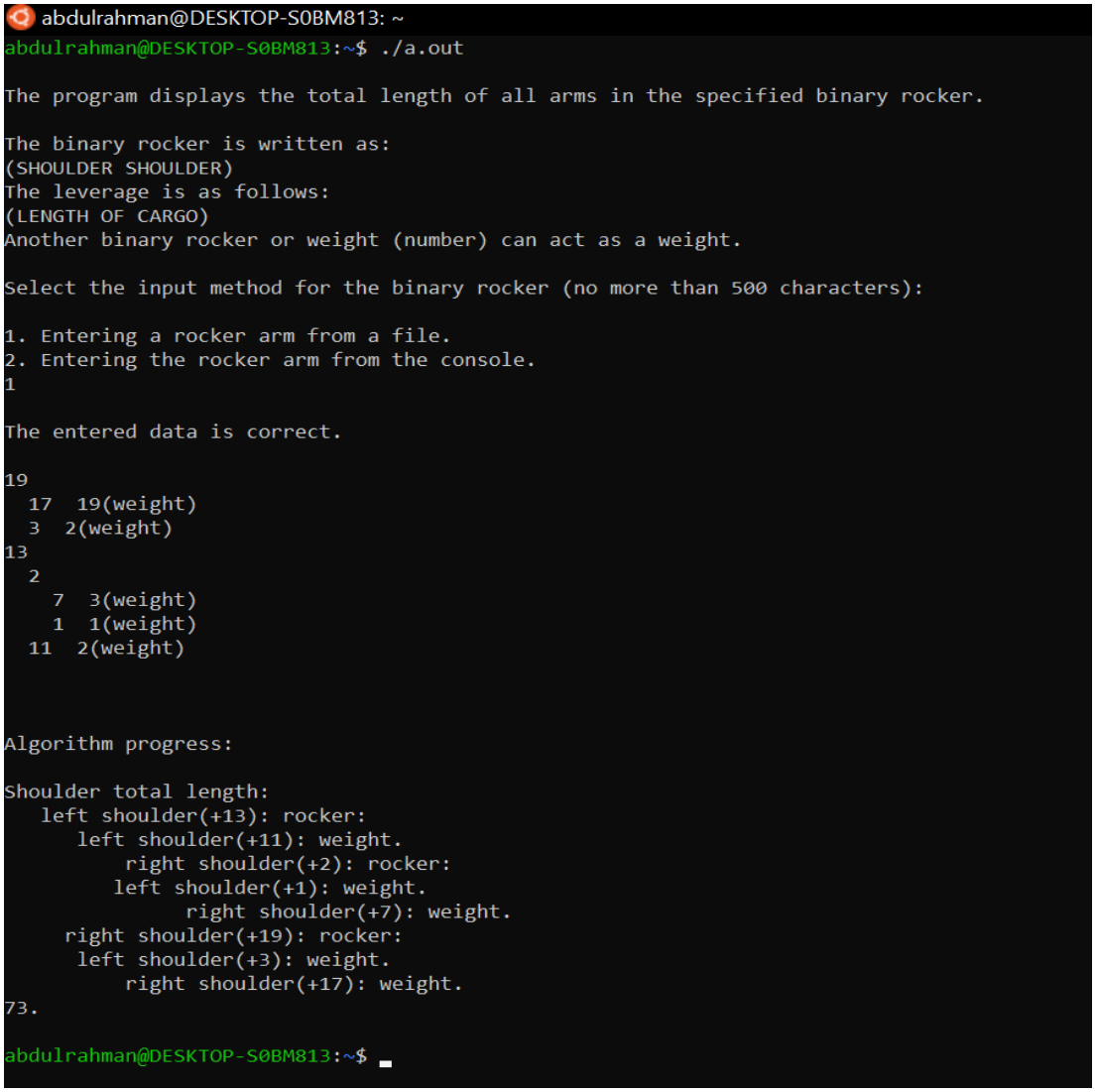


Входные данные:

Input.txt

((13 ((11 2) (2 ((1 1) (7 3))))) (19 ((3 2) (17 19))))

Результат работы программы:



Входные данные	Выходные данные
((5 3) (1 2))	Shoulder total length: 6
((2 ((13 5) (8 1))) (4 5))	Shoulder total length: 27
((5 ((142 67) (6 7))) (1 ((99 66) (1 1))))	Shoulder total length: 254
((5 2) (5 ((2 19) (14 6))))	Shoulder total length: 26
((13 ((11 2) (2 ((1 1) (7 3))))) (19 ((3 2) (17 ((2 2) (4 6))))))	Shoulder total length: 79
((1 1)(1))	Error! You entered incorrect data: Symbol № 7 - '('. Gap expected. The program has ended.
((11adsa 5) (4 3))	Error! You entered incorrect data: Symbol № 5 - 'a'. Gap expected. The program has ended.

ds((1 6) (4 3))	Error! You entered incorrect data: Symbol № 1 - 'd'. Symbol expected - '('. The program has ended.
((1 6) (4 3))))))))))	Error! You entered incorrect data: There are extra characters after the №12 character. The program has ended.
((1 6) (4 3)dasda)	Error! You entered incorrect data: There are extra characters after the №12 character. The program has ended.
((dsa1 6) (4 3))	Error! You entered incorrect data: Symbol № 3 - 'd'. Expected a value between 1 and 9. The program has ended.

ПРИЛОЖЕНИЕ Б
Код программы

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <iostream>
#include <fstream>

#define N 501

using namespace std;
// Binary rocker
typedef struct BinCor
{
   unsigned int length_left;
   unsigned int length_right;
   unsigned int weight_left;
   unsigned int weight_right;
   struct BinCor* cor_1;
   struct BinCor* cor_2;
} BinCor;


// display error messages
void errorMassage(int error_number, char* str, int index)
{
   cout << "\nError! You entered incorrect data:\n";
   switch (error_number)
   {
   case 1:
      cout << "Symbol № " << index + 1 << " - " << "'" << str[index] << "'.\n";
      cout << "Symbol expected - '('.\n";
      break;
   case 2:
      cout << "Symbol № " << index + 1 << " - " << "'" << str[index] << "'.\n";
      cout << "Symbol expected - ')'.\n";
      break;
   case 3:
      cout << "Symbol № " << index + 1 << " - " << "'" << str[index] << "'.\n";
      cout << "Expected a value between 1 and 9.\n";
      break;
   case 4:
      cout << "Symbol № " << index + 1 << " - " << "'" << str[index] << "'.\n";
      cout << "Gap expected.\n";
      break;
   case 5:
      cout << "Symbol № " << index + 1 << " - " << "'" << str[index] << "'.\n";
      cout << "Missing closing parenthesis.\n";
```

```
        break;
    case 6:
        cout << "Symbol № " << index + 1 << " - " << "'" << str[index] << "'.\n";
        cout << "value was expected between 1 and 9 or'('.\n";
        break;
    case 7:
        cout << "There are extra characters after the №" << index + 1 << " character.\n";
        break;
    }
}


BinCor* initBinCorElement()// Initialize the list item
{
    BinCor* element = new BinCor[sizeof(BinCor)];
    element->length_left = 0;
    element->length_right = 0;
    element->weight_left = 0;
    element->weight_right = 0;
    element->cor_1 = NULL;
    element->cor_2 = NULL;
    return element;
}


// read and create a binary rocker
// function returns 1 if an error occurred
int readBinCorElement(char* str, int index_1, int index_2, BinCor** element)
{
    *element = initBinCorElement();

    if (str[index_1++] != '(')
    {
        errorMassage(1, str, index_1 - 1);
        return 1;
    }
    if (str[index_2--] != ')')
    {
        errorMassage(2, str, index_2 + 1);
        return 1;
    }


    if (str[index_1++] != '(')
    {
        errorMassage(1, str, index_1 - 1);
        return 1;
    }

    if (!isdigit(str[index_1]) || str[index_1] == '0')
    {
```

```
    errorMassage(3, str, index_1);
    return 1;
}

while (1)
{
    if (isdigit(str[index_1]))
    {
        (*element)->length_left = (*element)->length_left * 10 + str[index_1] - '0';
        index_1++;
    }
    else
        break;
}

if (str[index_1++] != ' ')
{
    errorMassage(4, str, index_1 - 1);
    return 1;
}

if (isdigit(str[index_1]) && str[index_1] != 0)
{
    while (1)
    {
        if (isdigit(str[index_1]))
        {
            (*element)->weight_left = (*element)->weight_left * 10 + str[index_1] - '0';
            index_1++;
        }
        else
            break;
    }
}
else
    if (str[index_1] == '(')
    {
        int bracket_count = 0;
        int index;

        for (index = index_1; index < index_2; index++)
        {
            if (str[index] == '(')
                bracket_count++;
            if (str[index] == ')')
                bracket_count--;

            if (bracket_count == 0)
            {
                if (readBinCorElement(str, index_1, index, &((*element)->cor_1)))
                    return 1;
```

```
            index_1 = index + 1;
            break;
          }
        }
        if (bracket_count != 0)
        {
          errorMassage(5, str, index_1);
          return 1;
        }
      }
      else
      {
        errorMassage(6, str, index_1);
        return 1;
      }

if (str[index_1++] != ')')
{
  errorMassage(2, str, index_1 - 1);
  return 1;
}


if (str[index_1++] != ' ')
{
  errorMassage(4, str, index_1 - 1);
  return 1;
}



if (str[index_1++] != '(')
{
  errorMassage(1, str, index_1 - 1);
  return 1;
}


if (!isdigit(str[index_1]) || str[index_1] == 0)
{
  errorMassage(3, str, index_1);
  return 1;
}
while (1)
{
  if (isdigit(str[index_1]))
  {
    (*element)->length_right = (*element)->length_right * 10 + str[index_1] - '0';
    index_1++;
  }
  else
```

```
        break;
    }

    if (str[index_1++] != ' ')
    {
        errorMassage(4, str, index_1 - 1);
        return 1;
    }

    if (isdigit(str[index_1]) && str[index_1] != '0')
    {
        while (1)
        {
            if (isdigit(str[index_1]))
            {
                (*element)->weight_right = (*element)->weight_right * 10 + str[index_1] - '0';
                index_1++;
            }
            else
                break;
        }
    }
    else
        if (str[index_1] == '(')
        {
            int bracket_count = 0;
            int index;
            // finding a closing parenthesis
            for (index = index_1; index < index_2; index++)
            {
                if (str[index] == '(')
                    bracket_count++;
                if (str[index] == ')')
                    bracket_count--;

                if (bracket_count == 0) {
                    if (readBinCorElement(str, index_1, index, &((*element)->cor_2)))
                        return 1;
                    index_1 = index + 1;
                    break;
                }
            }

            if (bracket_count != 0)
            {
                errorMassage(5, str, index_1);
                return 1;
            }
        }
        else
        {
```

```
            errorMassage(6, str, index_1);
            return 1;
        }
    if (str[index_1] != ')')
    {
        errorMassage(2, str, index_1);
        return 1;
    }

    if (index_2 != index_1)
    {
        errorMassage(7, str, index_1);
        return 1;
    }

    return 0;
}


// create a binary rocker
// function returns 1 if an error occurred
int createBinCor(char* str, BinCor** binCor)
{
    int index_1 = 0;
    int index_2 = strlen(str) - 2;

    return readBinCorElement(str, index_1, index_2, binCor);
}


void print_bin_kor(BinCor* binCor, int deep_of_recursion)
{
    for (int i = 0; i < deep_of_recursion; i++)
        cout << "\t";
    cout << binCor->length_right << "  ";

    if (binCor->cor_2 == NULL)
    {
        cout << binCor->weight_right << "(weight)\n\n";
    }
    else
    {
        cout << "\t\n";
        print_bin_kor(binCor->cor_2, deep_of_recursion + 1);
    }

    for (int i = 0; i < deep_of_recursion; i++)
        cout << "\t";
    cout << binCor->length_left << "  ";

    if (binCor->cor_1 == NULL)
```

```
    {
        cout << binCor->weight_left << "(weight)\n\n";
    }
    else
    {
        cout << "\t\n";
        print_bin_kor(binCor->cor_1, deep_of_recursion + 1);
    }
}


// The return value is equal to the length of all arms
// in the given binary rocker
short length(const BinCor binCor, int deep_of_recursion)
{
    int result = 0;

    for (int i = 0; i < deep_of_recursion; i++)
        cout << "    ";
    cout << "left shoulder(+" << binCor.length_left << "): ";
    result += binCor.length_left;

    if (binCor.cor_1 == NULL)
    {
        cout << "weight.\n";
    }
    else
    {
        cout << "rocker: \n";
        result += length(*(binCor.cor_1), deep_of_recursion + 1);
    }

    for (int i = 0; i < deep_of_recursion; i++)
        cout << "    ";
    cout << "right shoulder(+" << binCor.length_right << "): ";
    result += binCor.length_right;

    if (binCor.cor_2 == NULL)
    {
        cout << "weight.\n";
    }
    else
    {
        cout << "rocker: \n";
        result += length(*(binCor.cor_2), deep_of_recursion + 1);
    }

    return result;
}
```

```
void free_memory(BinCor* binCor)
{
   if (binCor != NULL)
   {
      delete binCor->cor_1;
      delete binCor->cor_2;
   }
   delete binCor;
}



int main()
{
   char str[N];
   BinCor* binCor = NULL;
   int in = 0;


   // start reading and processing data
   cout << "\nThe program displays the total length of all arms in the specified binary rocker.\n";
   cout << "\nThe binary rocker is written as:\n";
   cout << "(SHOULDER SHOULDER)\n";
   cout << "The leverage is as follows:\n";
   cout << "(LENGTH OF CARGO)\n";
   cout << "Another binary rocker or weight (number) can act as a weight.\n";
   cout << "\nSelect the input method for the binary rocker (no more than 500 characters): \n";
   cout << "\n1. Entering a rocker arm from a file.";
   cout << "\n2. Entering the rocker arm from the console.\n";
   cin >> in;

   if (in == 1)
   {
      FILE* file = fopen("input.txt", "r");
      if (file == NULL)
         perror("File open error");
      fgets(str, N, file);
      fclose(file);
   }
   if (in == 2)
   {
      cout << "\nEnter a binary rocker (max. 500 characters):";
      fgets(str, 2, stdin);
      fgets(str, N, stdin);
   }

   if (createBinCor(str, &binCor))
   {
      cout << "The program has ended.\n\n";
      free_memory(binCor);
      return 0;
```

```
    }
    cout << "\nThe entered data is correct.\n\n";

    print_bin_kor(binCor, 0);
    cout << "\n\n\n";
    cout << "Algorithm progress:\n\n";
    cout << "\nShoulder total length: " << length(*binCor, 1) << ".\n\n";

    free_memory(binCor);

    return 0;
}
```