**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. Бинарное коромысло»**

Студент гр. 7382 Дрозд А. С.

Преподаватель Фирсов М. А.

Санкт-Петербург

2018

**Задание.**

Подсчитать общую длину всех плеч заданного бинарного коромысла bk.

Для этого ввести рекурсивную функцию

**short** Length (const БинКор bk).

**Пояснение задания.**

На вход программе подаётся последовательность символов. Итогом завершения программы должно быть выражение, дающее понять, что выражение не является бинарным коромыслом, или длину всех плеч коромысла, если оно токовым является.

**Описание алгоритма.**

Программа считывает строку, затем функция **are\_correct** начинает посимвольную проверку строки, на то, является ли она бинарным коромыслом или нет.

Если выражение не является бинарным коромыслом, то выводится предложение, позволяющее это понять, а также информация в какой части бинарного коромысла сделана ошибка. Если предложение является бинарным коромыслом, программа выведет предложение об этом и функция **maker** начнет создавать и заполнять список структур бинарных коромысел.

После этого функция **Length** подсчитывает длину всех плеч бинарного коромысла и выводит результат.

В конце программы функция **destroy** очищает динамическую память, выделенную на список структур бинарных коромысел.

**Описание структур**

В программе использовалось 2 структуры:

* struct binkor

В структуре находится 3 указателя на структуры. Указатель struct plecho\* left указывает на структуру левого плеча. Указатель struct plecho\* right указывает на структуру правого плеча. Указатель struct binkor\* head указывает на структуру, которая является предыдущей в списке структур.

* struct plecho

В структуре заходятся две переменные типа int и один указатель на структуру. Int length используется для хранения значения длины данного плеча. Int weight используется для хранения значения веса плеча. Указатель на структуру struct binkor\* wei указывает на структуру бинарного коромысла, повешенного на плечо. При заполненной структуре wei и weight не могут одновременно иметь положительные значения. Если на плечо подвешена гирька в struct binkor\* wei находится NULL. Если на плечо повешено бинарное коромысло, то wei равняется -1.

**Описание функций**

В программе использовалось 6 функций:

* **int are\_correct(char\* str)**

Принимает указатель на строку и проверяет, является ли строка бинкарным коромыслом. Если строка — это бинарное коромысло возвращает 0, в обратном случае возвращает число от 1 до 12. Каждое из чисел показывает каком месте бинарного коромысла была допущена ошибка.

* **void maker(char\* str, struct binkor\* a)**

Принимает указатель на строку и указатель на структуру бинарного коромысла. В строке должно находится корректное бинарное коромысло, поэтому функцию **maker** можно запускать только после проверки строки на корректность другими средствами. Сама функция заполняет структуру данными из строки попутно создавая новые структуры и заполняя их при необходимости. Все структуры, созданные функцией, составляют разветвленный двунаправленный список. Функция не имеет возвращаемого значения.

* **short Length(struct binkor a)**

Принимает указатель на структуру бинарного коромысла. Функция подсчитывает длину плеч бинарного коромысла, а также длину плеч всех бинарных коромысел, которые подвешены к нему. Возвращает подсчитанную длину.

* **int how\_many\_recurs(struct binkor\* a)**

Принимает указатель на структуру бинарного коромысла. Функция подсчитывает на каком месте от головы списка находится данная структура. Это число необходимо для красивого вывода дополнительной информации в функции **Length**. Возвращает найденное число.

* **int main()**

Главная функция программы. Выводит предложения, поясняющие работу программы. Считывает строку. Вызывает функцию для проверки полученной строки на корректность (является ли строка бинарным коромыслом). Исходя из возвращаемого значения функции проверки выдает предложение об ошибке ввода (возвращаемое значение принадлежит промежутку [1;12]) и завершает программу, или выводит предложение о том, что выражение прошло проверку на корректность (возвращаемое значение равно 0) и продолжает работу. Если выражение корректно программа создает структуру бинарного коромысла, которая будет являться головой списка. Структура создается динамически, так же, как и некоторые ее части(плечи), которые тоже являющееся структурами. После этого вызывается функция, заполняющая структуру бинарного коромысла данными из строки. По сути создается двунаправленный список структур отображающий бинарное коромысло. Потом происходит вызов функции подсчета длины полечь бинарного коромысла и вывод этой длины. Конечным шагом функции **main** является высвобождение взятой динамической памяти.

**Тестирование.**

|  |  |
| --- | --- |
| Вводимые данные | Результат: |
| ((4 7)(8 7)) | Выражение является Бинарным Коромыслом.  Длинна левого Плеча БинКор 0-ого по глубине равна 4.  Длинна правого Плеча БинКор 0-ого по глубине равна 8.  Длина плеч Бинарного коромысла 12. |
| ((4 7i)(8 7)) | Выражение не является Бинарным Коромыслом.  Левый груз не БинКор и не Гирька(или не закрыто левое плечо). |
| ((4 7)(8 ((8 7)(5 6))) | Выражение не является Бинарным Коромыслом.  БинКор не закрыт. |
| ((4and7)(8 7)) | Выражение не является Бинарным Коромыслом.  Между Длиной и Грузом левого Плеча не пробел(или груза нет вообще). |
| ({4 7](8 7)) | Выражение не является Бинарным Коромыслом.  Левое плечо не открыто. |

**Вывод.**

В процессе выполнения лабораторной работы были продуманы, созданы и реализованы на практике алгоритмы и методы работы по построению иерархического списка, а также по работе с ним. С помощью построения списка в памяти и вывода его на стандартный поток мы смогли подсчитать общую длину всех плеч бинарного коромысла.

**Приложение 1. Код программы.**

* [**test.sh**](https://github.com/makometr/AiSD/pull/37/files#_blank)

#!/bin/bash

gcc ./Sourse/main.c -o Lab2

echo -e '\_\_\_\_\_\_\_\nTest 1:'

cat ./Tests/test1.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test1.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 2:'

cat ./Tests/test2.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test2.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 3:'

cat ./Tests/test3.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test3.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 4:'

cat ./Tests/test4.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test4.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 5:'

cat ./Tests/test5.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test5.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 6:'

cat ./Tests/test6.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test6.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 7:'

cat ./Tests/test7.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test7.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 8:'

cat ./Tests/test8.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test8.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 9:'

cat ./Tests/test9.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test9.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 10:'

cat ./Tests/test10.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test10.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 11:'

cat ./Tests/test11.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test11.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 12:'

cat ./Tests/test12.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Lab2 < ./Tests/test12.txt

* **main**[**.**](https://github.com/makometr/AiSD/pull/37/files#_blank)**c**

#include <stdio.h>

#include <stdlib.h>

struct binkor //Структура бинарного коромысла(Далее БинКор)

{

struct plecho\* left;

struct plecho\* right;

struct binkor\* head;

};

struct plecho //Структура плеча

{

int length;

int weight;

struct binkor\* wei;

};

void maker(char\* str,struct binkor\* a) // Создает список бинарных коромысел

{

char\* simvol=str; //Счетчик, с помощью которого мы понимаем на каком символе сейчас находимся

short count1=0; //Переменная для считывания Длины/Груза

int exit=0; //Переменная для прохода БинКор

while(\*simvol=='(') //Доходим до Длины

simvol++;

while(\*simvol!=' ') //Посимвольно считаем Длину

{

count1=count1\*10+\*simvol-'0';

simvol++;

}

a->left->length=count1; //Записываем данные в структуру

simvol++;

if(\*simvol=='(') //Если Груз это БинКор

{

struct binkor \*c=(struct binkor \*)calloc(1,sizeof(struct binkor)); //Создаем новую структуру БинКор

c->head=a;

struct plecho\* l=(struct plecho \*)calloc(1,sizeof(struct plecho));

struct plecho\* r=(struct plecho \*)calloc(1,sizeof(struct plecho));

c->left=l;

c->right=r;

maker(simvol,c); //Заполняем созданную структуру

a->left->weight=-1; //Так как Груз является БинКор а не Гирькой, впишем в вес невозможное число веса Гирьки

a->left->wei=c; //Т.к weight не обозначает вес Гирьки, то считаем вес через БинКор(вписываем в wei адрес этого БинКор)

while(exit!=-1) //Цикл прохода БинКор

{

if(\*simvol=='(')

exit++;

if(\*simvol==')')

exit--;

simvol++;

}

}

else //Если груз это Гирька

{

count1=0;

while(\*simvol!=')') //Пока не посчитаем Груз

{

count1=count1\*10+\*simvol-'0';

simvol++;

}

a->left->weight=count1; //Записываем данные в структуру

a->left->wei=NULL; //Т.к. Груз не БинКор адрес делаем пустым

}

simvol++;

while(\*simvol=='(') //Доходим до Длины

simvol++;

count1=0;

while(\*simvol!=' ') //Посимвольно считаем Длину

{

count1=count1\*10+\*simvol-'0';

simvol++;

}

simvol++;

a->right->length=count1; //Записываем данные в структуру

if(\*simvol=='(') //Если Груз это БинКор

{

struct binkor \*c=(struct binkor \*)calloc(1,sizeof(struct binkor)); //Создаем новую структуру БинКор

c->head=a;

struct plecho\* l=(struct plecho \*)calloc(1,sizeof(struct plecho));

struct plecho\* r=(struct plecho \*)calloc(1,sizeof(struct plecho));

c->left=l;

c->right=r;

maker(simvol,c); //Заполняем созданную структуру

a->right->weight=-1; //Так как Груз является БинКор а не Гирькой, впишем в вес невозможное число веса Гирьки

a->right->wei=c; //Т.к weight не обозначает вес Гирьки, то считаем вес через БинКор(вписываем в wei адрес этого БинКор)

//Во второй раз не пишем цикл прохода БинКор т.к. уверены в правильности Введенного Бинарного коромысла и после Груза правого плеча не может быть ничего нужного

}

else //Если груз это Гирька

{

count1=0;

while(\*simvol!=')') //Пока не посчитаем Груз

{

count1=count1\*10+\*simvol-'0';

simvol++;

}

a->right->weight=count1; //Записываем данные в структуру

a->right->wei=NULL; //Т.к. Груз не БинКор адрес делаем пустым

}

}

int are\_correct(char\* str)// Проверяет на корректность БинКор

{

char\* simvol=str; //Счетчик, с помощью которого мы понимаем на каком символе сейчас находимся

int count=0; //Переменная для прохода Длины/Груза

int exit=0; //Переменная для прохода БинКор

if(\*(simvol+1)=='(' && \*simvol=='(') //если БинКор

{

simvol=str+2;

while(\*simvol>='0' && \*simvol<='9') //Проверяем есть ли Длина

{

count++;

simvol++;

}

if(count==0) //Если нет Длины возвращаем ошибку

return 1;

if(\*simvol!=' ') //Если между Длинной и Грузом нет пробела возвращаем ошибку

return 2;

simvol++;

if((\*(simvol+1)!='(' || \*simvol!='(') && (\*simvol<'0' || \*simvol>'9')) //Если Груз не БинКор и не Гирька возвращаем ошибку

return 3;

if(\*(simvol+1)=='(' || \*simvol=='(') //Если груз это БинКор,рекурсивно проверяем на коректность этот БинКор

{

count=are\_correct(simvol);

if(count!=0) //Если в рекурсивно проверенном БинКор есть ошибка возвращаем ошибку

return count;

}

count=0;

if(\*simvol>='0' && \*simvol<='9')

{

char\* sim=simvol; //Переменная для возвращения к началу Груза

while(\*simvol>='0' && \*simvol<='9') //Проверяем есть ли Длина

{

count++;

simvol++;

}

if(\*simvol!=')')

return 3;

simvol=sim;

}

while(exit!=-1) //Цикл прохода Груза

{

if(\*simvol=='(')

exit++;

if(\*simvol==')')

exit--;

simvol++;

}

if(\*simvol!='(') //Если не открыто Плечо возвращаем ошибку

return 5;

simvol++;

count=0;

while(\*simvol>='0' && \*simvol<='9') //Проверяем есть ли Длина

{

count++;

simvol++;

}

if(count==0) //Если нет Длины возвращаем ошибку

return 6;

if(\*simvol!=' ') //Если между Длиной и Грузом нет пробела возвращаем ошибку

return 7;

simvol++;

if((\*(simvol+1)!='(' || \*simvol!='(') && (\*simvol<'0' || \*simvol>'9')) //Если Груз не БинКор и не Гирька возвращаем ошибку

return 8;

if(\*(simvol+1)=='(' || \*simvol=='(') //Если груз это БинКор,рекурсивно проверяем на коректность этот БинКор

{

count=are\_correct(simvol);

if(count!=0) //Если в рекурсивно проверенном БинКор есть ошибка возвращаем ошибку

return count;

}

count=0;

if(\*simvol>='0' && \*simvol<='9')

{

char\* sim=simvol; //Переменная для возвращения к началу Груза

while(\*simvol>='0' && \*simvol<='9') //Проверяем есть ли Длина

{

count++;

simvol++;

}

if(\*simvol!=')')

return 3;

simvol=sim;

}

exit=0;

while(exit!=-1) //Цикл прохода Груза

{

if(\*simvol=='(')

exit++;

if(\*simvol==')')

exit--;

simvol++;

}

if(\*simvol!=')') //Если БинКор не закрыт возвращаем ошибку

return 10;

return 0;

}

else

if(\*(simvol)!='(')

return 11;

else

return 12;

}

void destroy(struct binkor\*a) //Высвобождает память всего списка БинКор

{

if(a->left->wei!=NULL) //Если вместо левого Груза использовалось БинКор

destroy(a->left->wei);

free(a->left);

if(a->right->wei!=NULL) //Если вместо правого Груза использовалось БинКор

destroy(a->right->wei);

free(a->right);

free(a);

}

int how\_many\_recurs(struct binkor\* a)

{

int count=0;

while(a->head!=NULL)

{

count++;

a=a->head;

}

return count;

}

short Length(struct binkor a) //Считает Длину всех Плеч БинКор

{

int count\_to\_rec=-1; //Переменная для подсчета глубины рекурсии

int indent; //Счетчик для создания отступов;

short count=0; //Переменная для подсчета Длины

count=count+a.left->length;

count\_to\_rec=how\_many\_recurs(&a); //Посчитаем глубину рекурсии

indent=count\_to\_rec;

while(indent!=0) //Выставим нужное количество отступов

{

printf(" ");

indent--;

}

printf("Длинна левого Плеча БинКор %d-ого по глубине равна %d.\n",count\_to\_rec,count);

if(a.left->wei!=NULL) //Если вместо левого Груза использовалось БинК

{

count=count+Length(\*(a.left->wei));

}

count=count+a.right->length;

indent=count\_to\_rec;

while(indent!=0) //Выставим нужное количество отступов

{

printf(" ");

indent--;

}

printf("Длинна правого Плеча БинКор %d-ого по глубине равна %d.\n",count\_to\_rec,a.right->length);

if(a.right->wei!=NULL) //Если вместо правого Груза использовалось БинКор

{

count=count+Length(\*(a.right->wei));

}

return count;

}

int main()

{

int result; // переменная, в которую выведется результат рекурсии

char str[100]; //строка, для считывания данных

printf("Введите выражение.\nБинКор = (<Плечо><Плечо>)\nПлечо = (<Длина> <Груз>)\nГруз = <Гирька>(число) или <БинКор>\n");

printf("Между Плечьми нет пробелов.\nДлину и Груз отделяет пробе(и ни что другое).\nМаксимальныя длинна БинКор 100 символов.\n");

printf("Если после верного Бинарного Коромысла вы введете мусор, программа его проигнорирует.\n");

fgets(str,100,stdin); //Считываение предпологаемого БинКор

result=are\_correct(str); //Проверка БинКор на корректность

if(result==0)

printf("Выражение является Бинарным Коромыслом.\n\n");

if(result!=0)

printf("Выражение не является Бинарным Коромыслом.\n");

if(result==1)

printf("В левом Плече нет Длины.\n");

if(result==2)

printf("Между Длиной и Грузом левого Плеча не пробел(или груза нет вообще).\n");

if(result==3)

printf("Левый груз не БинКор и не Гирька(или не закрыто левое плечо).\n");

if(result==5)

printf("После левого Плеча не идет правое Плечо.\n");

if(result==6)

printf("В правом плече нет Длины.\n");

if(result==7)

printf("Между Длиной и Грузом правого Плеча не пробел(или груза нет вообще).\n");

if(result==8)

printf("Правый Груз не БинКор и не Гирька.\n");

if(result==10)

printf("БинКор не закрыт.");

if(result==11)

printf("БинКор не открыт.");

if(result==12)

printf("Левое плечо не открыто.");

if(result==0) //Если БинКор корректно

{

struct binkor \*c=(struct binkor \*)calloc(1,sizeof(struct binkor)); //Создание структуры БинКор

c->head=NULL;

struct plecho\* l=(struct plecho \*)calloc(1,sizeof(struct plecho));

struct plecho\* r=(struct plecho \*)calloc(1,sizeof(struct plecho));

c->left=l;

c->right=r;

maker(str,c); //Создание списка БинКор

printf("\nДлина плеч Бинарного коромысла %d.",Length(\*c)); //Вывод результата подсчета Длины Плеч БинКор

destroy(c); //Высвобождение динамической памяти

}

}