**МИНОБРНАУКИ 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. 8383 |  | Федоров И.А |
| Преподаватель |  | Фирсов М.А |

Санкт-Петербург

2019

**Цель работы.**

Ознакомиться с основными понятиями и приемами рекурсивного программирования, получить навыки программирования рекурсивных функций.

**Основные теоретические положения.**

Самое простое определение рекурсии можно дать как вызов функции из неё же самой непосредственно (функция foo() вызывает функцию foo(), возможно с другими параметрами) или через другие функции. (например, функция foo() вызывает функцию bar(), а функция bar() — функцию foo(). Количество вложенных вызовов функции называется глубиной рекурсии. Важно понимать, что на каждый вызов функции приходятся как временные затраты, так и затраты памяти в области стека, основная часть которых приходится на хранение аргументов функции и ее локальных переменных.

Очень хорошо использование рекурсии ложится на задачи, которые можно разделять на себе подобные. Например для вычисления факториала числа :

(1.1)

Можно дать более удобное определение формулы (1.1), так называемое рекурсивное определение:

а) (1.2)

б) при

Опираясь на определение (1.2) можно написать рекурсивную функцию fact (n), которую кратко можно описать вот так:

fact (n) **if** n = 0 **then** 1 **else** fact , где использовано условное выражение **if** *b* **then** e1 **else** e2, означающее, что в том месте, где оно записано, следует читать *e*1, если выполняется условие *b*, и следует читать *e*2, если условие *b* не выполняется.

При разработке рекурсивного алгоритма, в первую очередь следует продумать условие завершения рекурсии, иначе рекурсия будет продолжаться до аварийного завершения программы. Часто, это выглядит как обработка частных конечных случаев. Продолжая предыдущий пример, условием остановки рекурсивного алгоритма будет . Следует так же подчеркнуть, что любой алгоритм можно построить без использования рекурсии.

**Задание.**

Вариант 18: Пусть определена функция Ф преобразования целочисленного вектора :

(1.3) Например: Ф(1,2,3,4,5) = 1,2,3,4,5; Ф(4,3,2,1) = 3,4,1,2; Ф(4,3,2) = 3,4,2. Отметим что функция Ф преобразует вектор, не меняя его длину. Реализовать функцию Ф рекурсивно.

**Реализация**

class Vector {

 public:

int\* data ;

int size = 0;

};

1) Класс Vector содержит динамический массив *data* , который хранит координаты целочисленного вектора , и целочисленную переменную *size* , которая хранит количество координат вектора. Члены класса объявлены как *public* члены для простоты обращения к ним.

void readFile (const char \* fileName, Vector &obj, int number ) ;

2) Функция *readFile* считывает заданное количество координат из файла (название которого введено пользователем). Функция принимает на вход название файла, ссылку на объект класса *Vector* и количество координат.

void recDepth ( Vector &obj, int pos , int length, int depth);

3) Функция *recDepth* выводит промежуточные данные во время работы рекурсивной функции.

void readConsole(Vector &obj);

4) Функция *readConsole* считывает координаты вектора введенные пользователем вручную.

5) Основная функция программы (main) создает объект класса Vector и осуществляет ввод данных из консоли , сопровождая его соответствующем диалогом. Пользователь вводит количество координат, выбирает один из способов ввода координат (из файла или из консоли). После обработки вектора пользователь может продолжить работу с программой или завершить её.

void transformVector ( Vector &obj, int pos , int length);

6) Функция *transformVector* - рекурсивная функция, выполняющая обработку вектора по формуле (1.3) . В качестве параметров принимает на вход ссылку на экземпляр класса *Vector* , параметр *pos* и *length.* Параметр *pos* - это позиция, с которой будет обрабатываться переданный вектор . Параметр *length* - это длина той части, которую обработает очередной вызов функции. Если переданный параметр , то функция рекурсивно вызывает себя для каждой из половин переданного вектора, если длина является четной. Если же длина нечетна, то функция вызовет себя для левой и правой половин, при этом длина левой части будет больше на 1 элемент . Если то в зависимости от условий, координаты будут поменяны местами. Если , то функция завершит работу (это условие завершения алгоритма).

**Вывод.**

В результате работы ознакомился с основными понятиями и приемами рекурсивного программирования. Была написана программа, которая использует рекурсивную функцию для преобразования вектора.

Протокол

Таблица 1 - тестирование

|  |  |  |
| --- | --- | --- |
| № теста | Входные данные | Выходные данные |
| 1. | 2,1,3,4,6,5,7 | 1,2,3,4,5,6,7 |
| 2. | -1, -1,-1,-1 | -1, -1,-1,-1 |
| 3. | 4,3,2 | 3,4,2 |
| 4. | -6,-9,-13,54,119,708 | -9,-6,-13,-119,-54,-708 |
| 5. | 1,2,3,4,5 | 1,2,3,4,5 |
| 6. | -19 | -19 |
| 7. | 5,3,13,8,9 | 3,5,13,8,9 |
| 8. | 4,3,2,1 | 3,4,2,1 |
| 9. | -5,6,10,-1,7,7,8,-10, 0, 99 | -5,6,10,-1,7,7,8,-10, 0, 99 |
| 10. | -2,1,3,4,6,5,7, -91,54,25 | 1,2,3,4,6,5,7,-91,25,54 |

Таблица 2 - из файла.

|  |  |  |  |
| --- | --- | --- | --- |
| № теста | Содержимое файла | Выходные данные | Комментарий |
| 1. | length=10 4,2,3,9,11 | 2,4,3,9,11 | Считывается либо введенное количество, либо пока не кончится файл. |
| 2. | length=3 2,1,3,4,6,5,7 | 1,2,3 | Считалось только введенное количество. |
| 3. | length=6 4,2,abcdf,dfdf,1 | 2,4 | Считывание прекращается, когда встречается не корректный символ. |

Действия программа при некорректном вводе (см. рис 1):

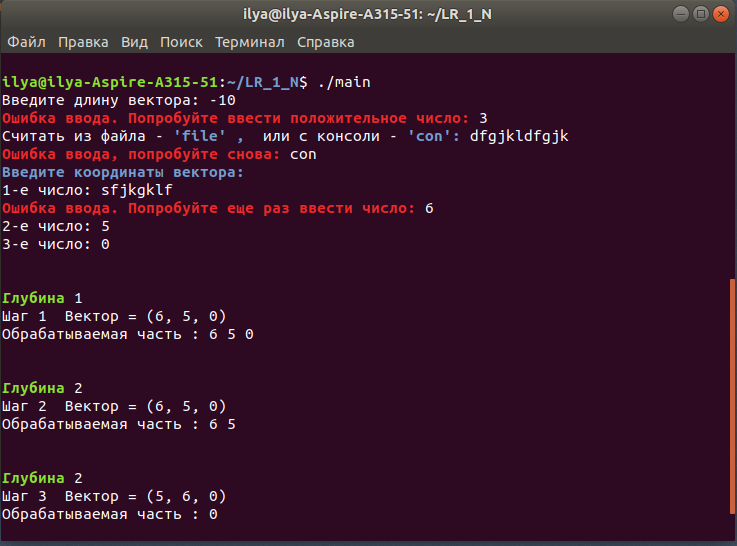


рис.1 (неверный ввод)

Пример полной работы (см. рис 2):

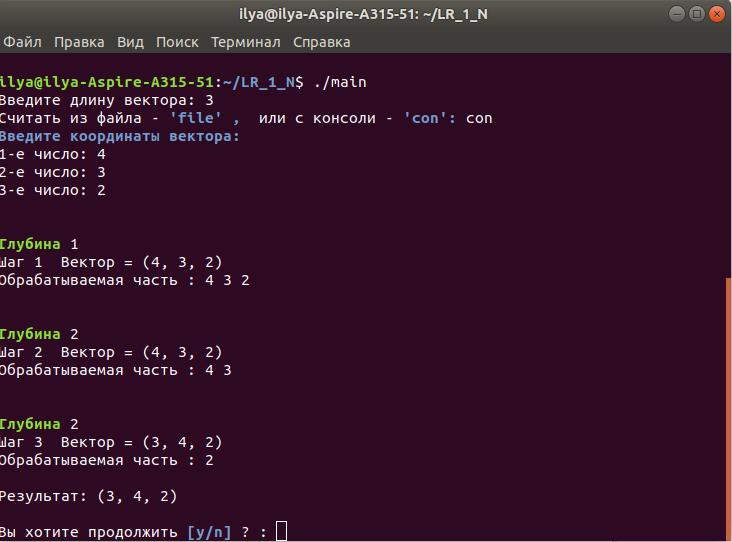


рис. 2

Отладка:

1) Программа аварийно завершалась вследствие ошибки *segmentation fault* (ошибка сегментации) при вводе отрицательного числа в качестве длины вектора , во время выделения динамической памяти, т.к. не была реализована проверка на положительность введенной длины.

2) Программа не воспринимала введенные отрицательные числа в качестве координат, т.к. система считывания не воспринимала любые символы , кроме цифр.

3) При вводе некорректных данных (например набор символов abcd, вместо числа) не реагировала на это, и записывала в координаты число 0 (т.к. при неправильном считывании возвращается значение 0).

Приложение (код программы).

#include <iostream>

#include <cstring>

#include <fstream>

#define QUIT "quit"

#define SIZE 30

using namespace std;

int step = 1; // глобальная переменная для вывода промежуточных данных

class Vector {

 public:

int\* data;

int size = 0;

friend ostream &operator<<(ostream &stream, Vector obj);

};

//перегрузка оператора << для класса Vector

ostream &operator<<(ostream &stream, Vector obj){

int i = 0; stream << "(";

for (i; i < obj.size-1 ; i++)

{

stream << obj.data[i] << ", ";

}

stream << obj.data[i] << ")"<<"\n";

return stream;

}

//функция считывания из файла

void readFile(const char \*fileName, Vector &obj, int number){

char ch; int count = 0;

ifstream in(fileName, ios::in | ios::binary);

while (!in){

cout << "Не удается открыть файл.введи корректное имя: \n";

char buf[SIZE];

cin.get(buf,SIZE, '\n');

ifstream in(fileName, ios::in | ios::binary);

}

char buf[20];

while(in && number != 0 && in.getline(buf, 20, ','))

{

if ((atoi(buf)) == 0 && !isdigit(buf[0])) {

cout << "Некорректная запись в файле";

obj.size = count;

return;

}

obj.data[count++] = atoi(buf);

number--;

}

obj.size = count; in.close();

cout << "Считанный из файла вектор: " << obj << "\n";

}

//функция вывода промежуточных данных

void recDepth(Vector &obj, int pos , int length, int depth){

cout <<"\n\n" << "" << depth;

cout <<"\n" << "Шаг "<< step << " Вектор = "<< obj; step++;

cout << "Обрабатываемая часть : " ;

for (int i = pos; i < pos+length; i++)

cout << obj.data[i] << " ";

cout << "\n";

}

//рекурсивная функция преобразования вектора

void transformVector(Vector &obj, int pos , int length,int depth = 0){

recDepth(obj, pos, length, depth);

if(length > 2)

{

if(length % 2 == 1)

{

transformVector(obj,pos,length/2+1, depth+1);

transformVector(obj,pos + length/2 +1,length/2, depth+1);

}

else

{

transformVector(obj,pos,length/2,depth+1);

transformVector(obj,pos + length/2,length/2, depth+1);

}

}

if(length == 2 && (obj.data[pos+1] < obj.data[pos]))

{

int x = obj.data[pos];

obj.data[pos] = obj.data[pos+1];

obj.data[pos+1] = x;

}

else if (length == 2 && (obj.data[pos+1]) >= obj.data[pos]){

}

if(length == 1) return;

}

void toLower(string &str){

int i = 0;

while (str[i]){

str[i] = tolower(str[i]);

i++;

}

}

//функция ввода с консоли

void readConsole(Vector &obj){

int tmp;

cout << "\033[1;31Введите координаты вектора: \033[0m\n";

for (int i = 0; i < obj.size ;i++)

{

cout << i+1 << "-e число: ";

while (!(cin >> tmp)|| (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "\033[1;31mОшибка ввода. Попробуйте еще раз ввести число: \033[0m";

}

obj.data[i] = tmp;

}

}

string file = "file";

string con = "con";

string stop = "n";

string again = "y";

int main()

{

string str;

do{

Vector obj;

cout << "Введите длину вектора: ";

while (!(cin >> obj.size)|| (cin.peek() != '\n') || obj.size <= 0)

{

cin.clear();

while (cin.get() != '\n');

cout << "\033[1;31mОшибка ввода. Попробуйте ввести положительное число: \033[0m";

}

obj.data = new int[obj.size];

cout << "Считать из файла - \033[1;34m\'file\' , \033[0m или с консоли - \033[1;34m\'con\': \033[0m";

cin >> str; toLower(str);

while (str!=file && str!=con)

{

cout << "\033[1;31mОшибка ввода, попробуйте снова: \033[0m" ;

cin >> str;

}

if (str == file)

{

string fileName;

cout << "Введите имя файла: ";

cin >> fileName;

readFile(fileName.c\_str(),obj,obj.size);

}

else {

readConsole(obj);

}

transformVector(obj, 0, obj.size, 1);

cout << "\n" << "Результат: " << obj;

cout << "\n" << "Вы хотите продолжить \033[1;34m[y/n]\033[0m ? : ";

cin >> str;

toLower(str);

while (str!=stop && str!=again){

cout << "Ошибка, введите [y/n] :";

cin >> str; toLower(str);

}

cout << "\n";

step = 1;

}while(str != "n");

return 0;

}