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: Рекурсия

Студентка гр. 9303 _____ Отмахова М.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы

Ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных функций на языке программирования C++.

Основные теоретические положения

Рассмотрим пример, который присутствует, по-видимому, во всех учебниках по программированию. Функция факториал натурального аргумента n обозначается как n! и определяется соотношением

$$n!=1\cdot2\cdot3...(n-1)\cdot n\ . \qquad (2.1)$$

Удобно доопределить 0!=1 и считать, что n – целое неотрицательное число.

Некоторым недостатком определения (2.1) является наличие в нём многоточия «...», передающего речевой оборот «и так далее» и имеющего интуитивно понятный читателю смысл. Можно дать точное, так называемое рекурсивное определение функции n!, лишенное этого недостатка, т. е. не апеллирующее к нашей интуиции. Определим:

$$\text{а)}\quad 0! = 1\,, \qquad (2.2)$$

$$\text{б)}\quad n! = (n-1)!\cdot n \quad \text{при } n > 0.$$

Соотношения (2.2) можно рассматривать как свойства ранее определенной функции, а можно (как в данном случае) использовать их для определения этой функции.

Далее для функции n! будем использовать привычное «функциональное» (префиксное) обозначение fact (n) , указывая имя функции и за ним в скобках – аргумент. Тогда (2.2) можно записать в виде

$$\text{fact}(n) = \begin{cases} 1, & \text{если } n = 0; \\ \end{cases} \qquad (2.3)$$

$$\lfloor \quad \text{fact}\,(n-1)\cdot n, \qquad \text{если } n > 0;$$

или в другой форме записи

$$\text{fact}\,(n) \equiv \text{if } n = 0 \text{ then } 1 \text{ else fact}\,(n-1)\cdot n, \tag{2.4}$$

где использовано условное выражение if b then e1 else e2, означающее, что в том месте, где оно записано, следует читать e1, если выполняется условие b, и следует читать e2, если условие b не выполняется.

Функция, определяемая таким образом, единственна. Действительно, пусть есть две функции, например: fact1 (n) и fact2 (n), удовлетворяющие соотношениям (2.2) или их эквивалентам (2.3), (2.4). Рассмотрим разность dfact (n) = fact1 (n) – fact2 (n). Очевидно, что, во-первых, в силу соотношения «а» из (2.2) имеем dfact (0) = 0, а, во-вторых, для функции dfact (n) также справедливо соотношение «б». Действительно,

dfact (n) = fact1 (n) – fact2 (n) = fact1 (n – 1) · n – fact2 (n – 1) · n =

= (fact1 (n – 1) – fact2 (n – 1)) · n = dfact (n – 1) · n.

По индукции легко доказывается, что из соотношений dfact (0) = 0 и dfact (n) = dfact (n – 1) · n следует, что dfact (n) = 0 для любого n > 0.

Как конструктивно могут быть использованы эти определения (как они «работают»)? Например, вычислим fact(4):

fact(4) = (fact(3)·4) = ((fact(2)·3)·4) = (((fact(1)·2)·3)·4) = ((((fact(0)·1)·2)·3)·4) =

= ((((1·1)·2)·3)·4) = (((1·2)·3)·4) = ((2·3)·4) = (6·4) = 24.

Здесь каждое новое использование рекурсивного определения заключается в скобки, а затем, когда ссылки на новые значения функции исчерпаны, последовательно каждое произведение двух сомножителей, заключенное в скобки, заменяется на результат умножения. Очевидно, что такое рекурсивное определение может рассматриваться как рецепт вычисления функции.

Задание

Вариант 19.

Функция Ф преобразования целочисленного вектора α определена следующим образом:

$$\Phi(\alpha) = \begin{cases} \alpha, \text{если } \|\alpha\| \le 2, \\ \Phi(\beta)\Phi(\gamma), \text{если } \alpha = \beta\gamma, \|\beta\| = \|\gamma\|, \|\alpha\| > 2, \\ \Phi(\beta\, a)\ \Phi\ (a\ \gamma), \text{ если } \alpha = \beta a \gamma, \|\beta\| = \|\gamma\|, \ \|\alpha\| > 2, \ \|a\| = 1. \end{cases}$$

Например: Ф(1,2,3,4,5) = 1,2,2,3,3,4,4,5; Ф(1,2,3,4,5,6) = 1,2,2,3,4,5,5,6; Ф(1,2,3,4,5,6,7) = 1,2,3,4,4,5,6,7; Ф(1,2,3,4,5,6,7,8) = 1,2,3,4,5,6,7,8. Отметим, что функция Ф может удлинять вектор. Реализовать функцию Ф рекурсивно.

Выполнение работы

В ходе выполнения лабораторной работы была написана рекурсивная функция phi, которая производит обработку целочисленного вектора alpha, в зависимости от его длины. На вход функция принимает такие параметры: вектор и глубину рекурсии. Обработку различных случаев реализуем с помощью цикла if. Возвращает наша функция вектор типа int. Если длина вектора <=2, тогда функция возвращает вектор неизменным. Если длина вектора четная – мы делим вектор пополам и для каждого из получившихся векторов рекурсивно вызываем функцию phi. Если длина вектора нечетная – то вектор делится пополам следующим образом: первый вектор – все элементы, расположенные до элемента, стоящего посередине, включая средний элемент, а второй вектор – элемент, стоящий посередине + все оставшиеся элементы. Данная функция выводит промежуточные действия, а именно для какого вектора применяется функция phi каждый раз.

Также были написаны вспомогательные функции, такие как: concat (функция, склеивающая два вектора) и print_vector (функция, выводящая вектор на экран).

В функции main производится считывание длины вектора, после чего и считывание координат вектора с помощью цикла for. Далее обрабатываем считанный вектор с помощью написанной нами функции phi и выводим получившийся вектор. Исходный код программы можно найти в приложении А. Тестирование программы представлено в приложении Б.

Выводы.

В ходе выполнения данной лабораторной работы были изучены основные понятия и приёмы рекурсивного программирования, получены навыки программирования рекурсивных функций на языке программирования C++.

Была написана программа на языке C++, производящая обработку вектора целых чисел, в зависимости от длины данного вектора. Программа прошла тестирование, результаты работы программы удовлетворяют ожидаемым результатам.

При решении данной задачи очень удобно, эффективно и целесообразно использовать рекурсию.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>

using namespace std;

vector<int> concat(vector<int> a, vector<int> b) {
    vector<int> c;
    for (int i = 0; i < a.size(); i++) {
        c.push_back(a[i]);
    }

    for (int i = 0; i < b.size(); i++) {
        c.push_back(b[i]);
    }
    return c;
}

vector<int> print_vector(vector<int> a) {
    for ( int i = 0; i < a.size(); i++) {
        cout << a[i] << ' ';
    }
}

vector<int> phi(vector<int> alpha, int depth) {
    for (int i = 0; i < depth; i++) {
        cout << " ";
    }
    cout << "Phi called for vector: ";
    print_vector(alpha);
    cout << endl;

    if (alpha.size() <= 2) {
        return alpha;
```

```
    } else if (alpha.size() % 2 == 0) {
        vector<int> b;
        for (int i = 0; i < alpha.size() / 2; i++) {
            b.push_back(alpha[i]);
        }

        vector<int> y;
        for (int i = alpha.size() / 2; i < alpha.size(); i++) {
            y.push_back(alpha[i]);
        }

        return concat(phi(b, depth + 1), phi(y, depth + 1));
    } else {
        vector<int> b;
        for (int i = 0; i < alpha.size() / 2; i++) {
            b.push_back(alpha[i]);
        }

        vector<int> a = {alpha[alpha.size() / 2]};

        vector<int> y;
        for (int i = alpha.size() / 2 + 1; i < alpha.size(); i++)
{
            y.push_back(alpha[i]);
        }

        return concat(
                phi(concat(b, a), depth + 1),
                phi(concat(a, y), depth + 1)
        );
    }
}


int main() {
```

```
    int n;
    cin >> n;

    vector<int> alpha;
    alpha.resize(n);

    for (int i = 0; i < n; i++) {
        cin >> alpha[i];
    }

    vector<int> res = phi(alpha, 1);

    cout << "\nResult is: ";
    print_vector(res);

    return 0;
}
```

ПРИЛОЖЕНИЕ В

ТЕСТИРОВАНИЕ ПРОГРАММЫ

Входные данные	Результат работы программы