|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования |
| **«МИРЭА – Российский технологический университет»** |
| **РТУ МИРЭА** |
|  |

|  |  |
| --- | --- |
| **Отчет по выполнению практического задания № 7** | |
| **Тема:** | |
| **«Рекурсивные алгоритмы и их реализация»** | |
| Дисциплина: «Структуры и алгоритмы обработки данных» | |
|  | Выполнил студент: Зернов Н.А. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ №1 4](#_30j0zll)

[2.1 Формулировка задачи (В списке №13, Вариант 13) 4](#_1fob9te)

[2.2 Итерационный алгоритм 4](#_2et92p0)

2.3 Рекуррентная зависимость 6

[2.4 Рекурсивная функция 7](#_17dp8vu)

[2.5 Объединение программ 9](#_1ci93xb)

[2.6 Вывод по заданию №1 11](#_dslhzqc2xurl)

[3 ЗАДАНИЕ №2 12](#_2bn6wsx)

[3.1 Формулировка задачи 12](#_qsh70q)

[3.2 Рекурсивная функция 12](#_u290pxmzmczz)

[3.3 Вывод по заданию №2 14](#_fc3bwntftrcf)

[4 ВЫВОДЫ 16](#_ihv636)

[5 ЛИТЕРАТУРА 17](#_32hioqz)

# **1 ЦЕЛЬ**

Получить знания и практические навыки по разработке и реализации рекурсивных процессов.

# **2 ЗАДАНИЕ №1**

## **2.1 Формулировка задачи (В списке №13, Вариант 13)**

Разработать и протестировать рекурсивные функции в соответствии с задачами варианта

Требования к выполнению первой задачи варианта:

• приведите итерационный алгоритм решения задачи

• реализуйте алгоритм в виде функции и отладьте его

• определите теоретическую сложность алгоритма

• опишите рекуррентную зависимость в решении задачи

• реализуйте и отладьте рекурсивную функцию решения задачи

• определите глубину рекурсии, изменяя исходные данные

• определите сложность рекурсивного алгоритма, используя метод подстановки и дерево рекурсии

• приведите для одного из значений схему рекурсивных вызовов

• разработайте программу, демонстрирующую выполнение обеих функций и покажите результаты тестирования.

Задание: Дана последовательность из N чисел Х1,Х2,....,ХN. Вычислить значение выражения: Хn(Хn+Xn-1)(Хn+Xn-1+Xn-2)(Хn+Xn-1+Xn2+Xn-3)... (Хn+Xn-1+Xn-2+...+X1). Массив не использовать.

## **2.2 Итерационный алгоритм**

Итерационный алгоритм - это метод, в котором последовательность операций выполняется несколько раз или до выполнения определенного условия с использованием циклов. В отличие от рекурсивных алгоритмов, которые вызывают самих себя для решения подзадач, итерационные алгоритмы используют повторяющиеся итерации для достижения решения.

Проанализируем теоретическую сложность данного алгоритма.

1. Входные данные: Пользователь вводит количество чисел в последовательности n, а затем последовательно вводит n чисел.
2. Для каждого из n чисел выполняются следующие действия:

- Ввод числа: это операция O(1).

- Обновление суммы: операция O(1).

- Вычисление произведения: операция O(1).

1. Всего имеется n итераций, каждая из которых выполняет операции с константной сложностью O(1).

Таким образом, общая теоретическая сложность алгоритма составляет O(n), где n - количество чисел в последовательности. Это потому, что время выполнения алгоритма линейно зависит от количества введенных пользователем чисел.

Реализуем вычисление значения последовательности для некоторого целого числа n с помощью итерационного алгоритма(блок кода 1). Для этого будут использоваться библиотека iostream и пространство имен как std.

iostream: библиотека в C++, является заголовочным файлом с классами, функциями и переменными для предоставления ввода и вывода данных через стандартные потоки cout (для вывода на консоль) и cin (для ввода с консоли).

**#include** <iostream>

**using** **namespace** std;

**long** **long** iterative(**int** n) {

**long** **long** resault **=** 1;

**long** **long** sum **=** 0;

**int** num;

**for** (**int** i **=** 0; i **<** n; i**++**) {

        cout **<<** "X" **<<** (n **-** i) **<<** " = ";

        cin **>>** num;

        sum **+=** num;

        resault **\*=** sum;

    }

**return** resault;

}

**int** main() {

    setlocale(LC\_ALL, "RUS");

**int** n;

    cout **<<** "Введите количество чисел в последовательности: ";

    cin **>>** n;

    cout **<<** "Ответ в итеративной форме: " **<<** iterative(n);

**return** 0;

}

Блок схема 1 - реализация итерационного алгоритма для задачи 1

Проведем тестирование данного кода на разных значениях (рис. 1-3).

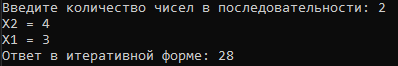


Рисунок 1 - Тестирование алгоритма

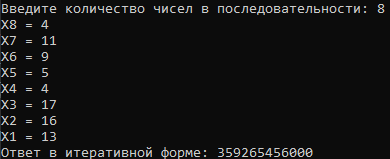


Рисунок 2 - Тестирование алгоритма



Рисунок 3 - Тестирование алгоритма

## **2.3 Рекуррентная зависимость**

В вашем исходном коде присутствует рекурсивная зависимость между функцией function и переменной rez. Вот как это происходит. В функции function происходит рекурсивный вызов с уменьшением значения n на 1. Переменная rez используется для накопления результата умножения суммы sum на предыдущее значение rez. Таким образом, каждый вызов function использует rez, который зависит от предыдущего вызова function, создавая рекурсивную зависимость.

Это означает, что каждый раз, когда function вызывается, она использует rez, который зависит от предыдущего вызова function. Это может привести к непредсказуемым результатам или нежелательному поведению программы.

Для решения этой проблемы можно передавать rez как параметр в функцию function, чтобы избежать использования глобальной переменной и рекурсивной зависимости. Вот как это делается в переработанной версии кода, представленной выше.

## **2.4 Рекурсивная функция**

Рекурсивная функция - это функция, которая вызывает саму себя внутри своего определения. В процессе выполнения рекурсивной функции задача разбивается на более простые подзадачи, и каждая из этих подзадач решается с использованием той же функции.

Глубина рекурсии - это количество вызовов рекурсивной функции, которые происходят в процессе ее выполнения, прежде чем будет достигнут базовый случай, который завершает рекурсию.

Функция function вызывает саму себя с аргументом n - 1. Каждый раз, когда n уменьшается на 1, мы уменьшаем общую глубину рекурсии на 1. Когда n становится равным 0, рекурсия завершается.

1. Если n = 5, то рекурсивная функция будет вызвана 5 раз: function(5) -> function(4) -> function(3) -> function(2) -> function(1) -> function(0). Глубина рекурсии составит 5.

2. Если n = 1, то рекурсивная функция будет вызвана 1 раз: function(1) -> function(0). Глубина рекурсии также будет равна 1.

Определим сложность рекурсивного алгоритма с помощью метода подстановки и дерева рекурсии.

Метод подстановки:

Пусть T(n) - время работы алгоритма для входного значения n.

По коду видно, что каждый вызов function выполняет постоянное количество операций (ввод числа, вычисление суммы, умножение на rez). Пусть это число операций равно c.

Тогда время работы T(n) может быть описано следующим образом:

- При n > 0: T(n) = c + T(n-1)

- При n = 0: T(n) = c

Решая рекуррентное соотношение, получаем:

- T(n) = c\*n + c

Следовательно, сложность алгоритма - O(n).

Теперь представим схему рекурсивных вызовов для случая n = 3:

function(3) -> function(2) -> function(1) -> function(0)

Это представление демонстрирует, как каждый вызов function порождает другие вызовы до тех пор, пока не достигнет базового случая при n = 0.

Реализуем вычисление значения последовательности для некоторого целого числа n с помощью итерационного алгоритма(блок кода 2). Для этого будут использоваться библиотека iostream.

**#include** <iostream>

**using** **namespace** std;

**static** **long** **long** **int** rez **=** 1;

**static** **long** **long** **int** cont **=** 1;

**long** **long** **int** function(**long** **long** **int** n) {

**long** **long** **int** x, sum **=** 0;

**if** (n **==** 0)

**return** 0;

**else** **if** (n **>** 0) {

        cout **<<** "Введите X" **<<** cont **<<** " = ";

        cin **>>** x;

**++**cont;

        sum **=** x **+** function(n **-** 1);

        rez **=** rez **\*** sum;

**return** sum;

    }

}

**int** main() {

    setlocale(LC\_ALL, "RUS");

**int** n;

    cout **<<** "Введите количество чисел в последовательности: ";

    cin **>>** n;

    function(n);

    cout **<<** "Ответ в рекурсивной форме: " **<<** rez **<<** endl;

**return** 0;

}

Блок кода 2 - Рекурсивная функция для задачи 1

Проведем тестирование данного кода на разных значениях (рис. 1-3).

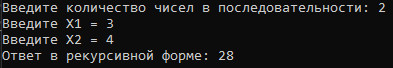


Рисунок 4 - Тестирование адаптированной программы

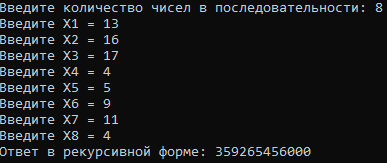


Рисунок 5 - Тестирование адаптированной программы



Рисунок 6 - Тестирование адаптированной программы

## **2.5 Объединение программ**

Объединим программы итерационного алгоритма и рекурсивной функции в блок коде 3 с выполнением обоих алгоритмов на заданном числе. Продемонстрируем результаты работы программы на рисунках 8-10.

**#include** <iostream>

**using** **namespace** std;

**static** **long** **long** **int** rez **=** 1;

**static** **long** **long** **int** cont **=** 1;

**long** **long** **int** function(**long** **long** **int** n) {

**long** **long** **int** x, sum **=** 0;

**if** (n **==** 0)

**return** 0;

**else** **if** (n **>** 0) {

        cout **<<** "Введите X" **<<** cont **<<** " = ";

        cin **>>** x;

**++**cont;

        sum **=** x **+** function(n **-** 1);

        rez **=** rez **\*** sum;

**return** sum;

    }

}

**long** **long** iterative(**int** n) {

**long** **long** resault **=** 1;

**long** **long** sum **=** 0;

**int** num;

**for** (**int** i **=** 0; i **<** n; i**++**) {

        cout **<<** "X" **<<** (n **-** i) **<<** " = ";

        cin **>>** num;

        sum **+=** num;

        resault **\*=** sum;

    }

**return** resault;

}

**int** main() {

    setlocale(LC\_ALL, "RUS");

**int** n;

    cout **<<** "Введите количество чисел в последовательности: ";

    cin **>>** n;

    function(n);

    cout **<<** "Ответ в рекурсивной форме: " **<<** rez **<<** endl;

    cout **<<** "Ответ в итеративной форме: " **<<** iterative(n);

**return** 0;

}

Блок кода 3 - Объединение программ

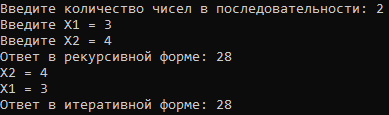


Рисунок 8 - Тестирование программы

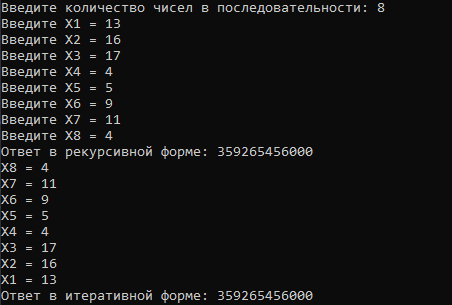


Рисунок 9 - Тестирование программы

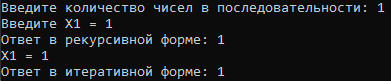


Рисунок 10 - Тестирование программы

## **2.6 Вывод по заданию №1**

Рекурсивный алгоритм, основанный на вызове самого себя для решения подзадач, обычно представляет собой компактный и интуитивно понятный подход. Его преимущества заключаются в простоте и естественности рекурсивных вызовов. Однако он может иметь более высокую временную сложность и потреблять больше памяти из-за создания дополнительных вызовов стека. Существует также риск переполнения стека при обработке больших объемов данных или глубокой рекурсии.

С другой стороны, итерационный алгоритм использует циклы для выполнения задачи и обычно является более эффективным с точки зрения времени и памяти, так как не создает дополнительных вызовов стека. Однако он может быть менее интуитивным для понимания, особенно в случае сложных задач. Итерационные методы обычно предпочтительны для обработки больших объемов данных или при работе с ограниченными ресурсами.

# **3 ЗАДАНИЕ №2**

## **3.1 Формулировка задачи**

Требования к выполнению второй задачи варианта:

• рекурсивную функцию для обработки списковой структуры согласно варианту. Информационная часть узла – простого типа – целого;

• для создания списка может быть разработана простая или рекурсивная функция по желанию (в тех вариантах, где не требуется рекурсивное создание списка);

• определите глубину рекурсии

• определите теоретическую сложность алгоритма

• разработайте программу, демонстрирующую работу функций и покажите результаты тестов.

Задание: Найти в двунаправленном списке количество четных элементов.

## **3.2 Рекурсивная функция**

Реализуем удалить нули из однонаправленного списка с помощью рекурсивной функции(блок кода 4). Создание списка будет разработан простой функцией. Для этого будут использоваться библиотеки iostream.

**#include** <iostream>

**using** **namespace** std;

**struct** Node {

**int** data;

    Node**\*** next;

    Node(**int** d) : data(d), next(**nullptr**) {}

};

**void** push\_front\_Noode(Node**\*&** head,**int** newData) {

    Node**\*** newNode **=** **new** Node(newData);

    newNode->next **=** head;

    head **=** newNode;

}

**void** output\_Node(Node**\*&** head) {

    Node**\*** temp **=** head;

**while** (temp **!=** **nullptr**) {

        cout **<<** temp->data **<<** " ";

        temp **=** temp->next;

    }

    cout **<<** endl;

}

// Рекурсивная функуия удаления

**void** delete\_Node(Node**\*&** head) {

**if** (head **==** **nullptr**)

**return**;

    delete\_Node(head->next);

**if** (head->data **==** 0) {

        Node**\*** temp **=** head;

        head **=** head->next;

**delete** temp;

    }

}

**int** main() {

    setlocale(LC\_ALL, "Rus");

    Node**\*** head **=** **nullptr**;

    push\_front\_Noode(head, **-**1);

    push\_front\_Noode(head, 1);

    push\_front\_Noode(head, 0);

    push\_front\_Noode(head, 4);

    push\_front\_Noode(head, 0);

    push\_front\_Noode(head, 1);

    push\_front\_Noode(head, 11);

    cout **<<** "Список: ";

    output\_Node(head);

    delete\_Node(head); // Удаление нулей

    cout **<<** "Список после удаления: ";

    output\_Node(head);

**while** (head **!=** **nullptr**) {

        Node**\*** temp **=** head;

        head **=** head->next;

**delete** temp;

    }

**return** 0;

}

Блок кода 4 - Программа для задания 2 с рекурсивной функцией

Глубина рекурсии в методе delete\_Node определяется количеством узлов в списке с нулевыми значениями. Каждый раз, когда функция вызывает саму себя, она перемещается на один узел вперёд по списку. Глубина рекурсии будет равна количеству нулей в списке. Если в списке нет нулей, то глубина рекурсии будет равна нулю.

К примеру, если у нас есть список, содержащий 3 нуля, то функция delete\_Node будет вызвана 3 раза, и глубина рекурсии будет равна трем. Аналогично и с другим количеством нулей в списке.

Теоретическая сложность алгоритма рекурсии delete\_Node зависит от количества узлов в списке и количества узлов с нулевыми значениями. Предположим, у нас есть список из n элементов, и m из них содержат нулевые значения. Каждый вызов функции delete\_Node просматривает один узел списка и, при необходимости, удаляет его. Поскольку список проходится рекурсивно, общая сложность будет состоять из O(n) операций. Однако, если каждый узел списка содержит нулевое значение, и функция должна удалить все такие узлы, в этом случае будет O(n) операций на просмотр каждого узла и еще O(n) операций на удаление нулевых узлов.

Таким образом, общая теоретическая сложность алгоритма рекурсии delete\_Node равна O(n) в худшем случае, где n - количество узлов в списке.

Проведем тестирование программы и предоставим результаты на рисунке 11.



Рисунок 11 - Тестирование программы

## **3.3 Вывод по заданию №2**

Взаимодействие рекурсии с однонаправленным списком предоставляет эффективный способ обработки данных в этой структуре, где каждый элемент имеет ссылку только на следующий узел. Рекурсивные алгоритмы обхода и списками позволяют реализовать удаление элементов из списка.

Рекурсивная функция, используемая для удаления нулей из списка, отражает структуру списка и естественным образом его обрабатывает. Каждый рекурсивный вызов перемещается к следующему узлу списка до достижения его конца. При этом проверяется значение данных текущего узла: если оно равно нулю, узел удаляется из списка. Этот процесс повторяется до тех пор, пока не будет просмотрен весь список.

Таким образом, взаимодействие рекурсии с однонаправленным списком обеспечивает реализацию гибких и эффективных алгоритмов обработки данных, при этом обеспечивая простоту и понятность кода.

# **4 ВЫВОДЫ**

В процессе выполнения практической работы были достигнуты следующие результаты. Получены теоретические знания о создании и применении рекурсивных процессов. Накоплены практические навыки в разработке и реализации рекурсивных алгоритмов. Разработаны и реализованы программы, использующие как рекурсивные функции, так и итеративные алгоритмы. Проведено сравнение эффективности программ, основанных на рекурсии и итерациях. Создана программа, в которой использована рекурсивная функция для работы с однонаправленным списком. Проведена оценка сложности и глубины рекурсивной функции. Проведено тестирование всех разработанных программ.

Таким образом, основная цель практической работы - получение знаний и практических навыков по работе с рекурсивными процессами - была успешно достигнута.

# **5 ЛИТЕРАТУРА**

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. – СПб: Питер, 2017. – 288 с.

2. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с.

3. Кнут Д.Э. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 832 с.

4. Кораблин Ю.П. Структуры и алгоритмы обработки данных: учебно-методическое пособие / Ю.П. Кораблин, В.П. Сыромятников, Л.А. Скворцова. – М.: РТУ МИРЭА, 2020. — 219 с.

5. Кормен Т.Х. и др. Алгоритмы: построение и анализ, 3-е изд. – М.: ООО «И.Д.Вильямс», 2013. – 1328 с.

6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., - М.: Техносфера, 2018. – 416 с.

7. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.

8. Скиена С. Алгоритмы. Руководство по разработке, - 2-е изд. – СПб: БХВ-Петербург, 2011. – 720 с.

9. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2-е изд. – СПб: ООО «Альфа-книга», 2017. – 432 с.

10. AlgoList – алгоритмы, методы, исходники [Электронный ресурс]. URL: http://algolist.manual.ru/ (дата обращения 15.03.2022).

11. Алгоритмы – всё об алгоритмах / Хабр [Электронный ресурс]. URL: https://habr.com/ru/hub/algorithms/ (дата обращения 15.03.2022).

12. НОУ ИНТУИТ | Технопарк Mail.ru Group: Алгоритмы и структуры данных [Электронный ресурс]. URL: https://intuit.ru/studies/courses/3496/738/info (дата обращения 15.03.2022).