

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

**РТУ МИРЭА**

Отчет по выполнению практического задания № 1.7

# Тема:

Рекурсивные алгоритмы и их реализация

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Тринеев П. С.

Фамилия И.О.

Группа: ИКБО-32-22

Номер группы

Москва – 2023

**Занятие 7.** Рекурсивные алгоритмы и их реализация

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

Задание

Разработать и протестировать рекурсивные функции в соответствии с задачами варианта

1) 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.

2) 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

2. Задача 1

2.1. Условие задачи

Перевести число из 10-системы счисления в систему с основанием В(1<В≤10)

2.2. Описание алгоритма – рекуррентная зависимость

Рекурсия — состоит в определении, описании, изображении какого-либо объекта или процесса внутри самого этого объекта или процесса. Это ситуация, когда объект является частью самого себя.

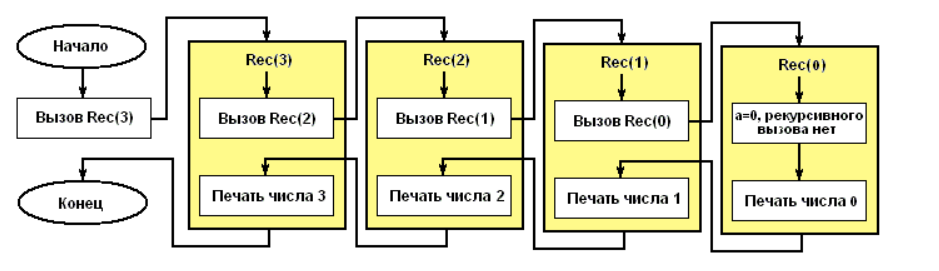


Рис. 1 - Блок схема работы рекурсивной процедуры.

Глубина рекурсии вычисляется примерно по следующей

формуле (из-за того, что не всегда массив делится на две равные

части оценка примерная) ,что подтверждается

практическими данными. График роста глубины рекурсии от количества элементов:

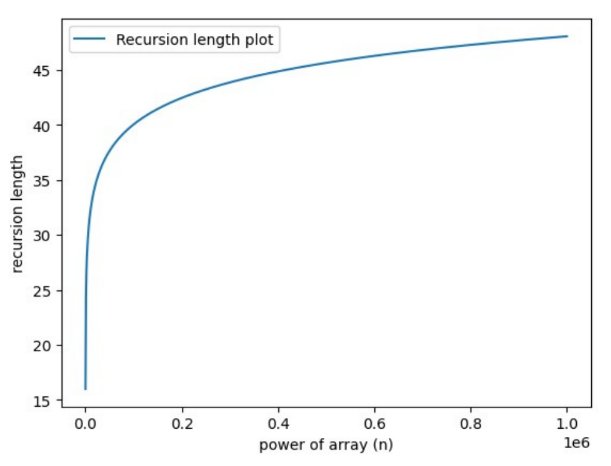


Рис. 2 - График роста глубины рекурсии

2.3. Код программы и скриншоты результатов тестирования

#include <iostream>

#include <clocale>

#include <conio.h>

using namespace std;

void ss(int i, int s) {

if (i == 0)

cout << 0;

else {

if (i / s > 0) {

ss(i / s, s);

}

cout << i % s;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int i, s;

cout << "Введите исходное число в десятичной системе счисления: ";

cin >> i;

cout << "Введите систему счисления < 10: ";

cin >> s;

if (s >= 10) {

cout << "Основание должно быть меньше 10!";

return EXIT\_FAILURE;

}

cout << "Число " << i << " в " << s << " системе счисления: ";

ss(i, s);

\_getch();

}

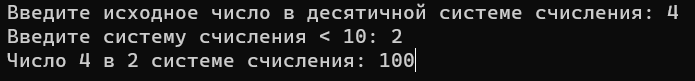


Рис. 3 - скриншот результатов тестирования (1)

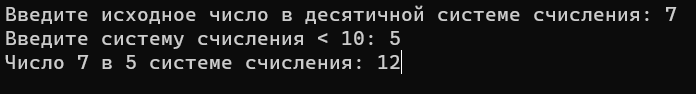


Рис. 4 - скриншот результатов тестирования (2)

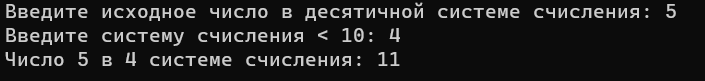


Рис. 5 - скриншот результатов тестирования (3)

3. Задача 2

3.1. Условие задачи

Удаление очереди, реализованной на однонаправленном списке

3.2 Описание алгоритма – рекуррентная зависимость

Очередью называется упорядоченный набор элементов, которые могут удаляться с её начала и помещаться в её конец.

Существует несколько способов реализации очереди:

* с помощью одномерного массива;
* с помощью связанного списка;
* с помощью класса объектно-ориентированного программирования.

Простейшие операции с очередью:

* init() инициализация очереди.
* insert(q, x) — помещение элемента x в конец очереди q (q — указатель на очередь);
* x=remove(q) — удаление элемента x из очереди q;
* isempty(q) — возвращает 1, если очередь пуста и 0 в противном случае;
* print(q) – вывод элементов очереди q.

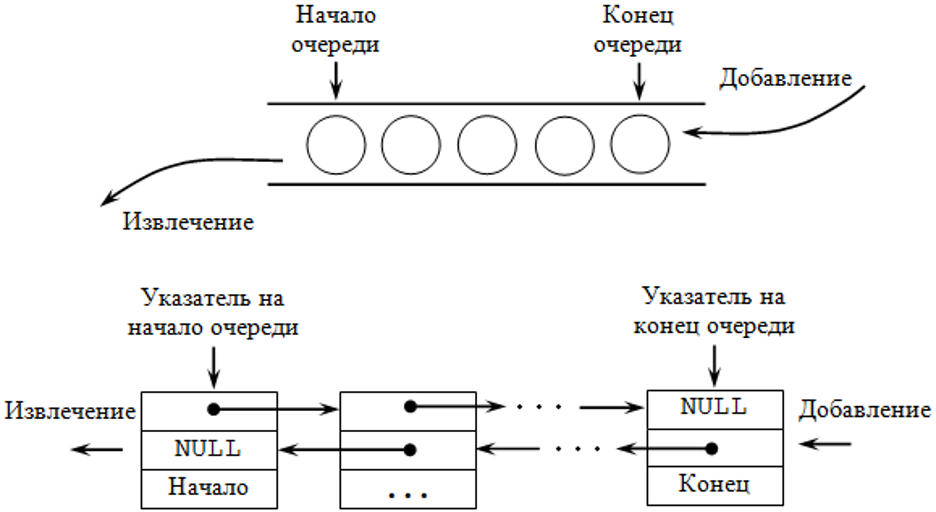


Рис. 6 – Иллюстрация работы очереди

3.3. Код программы и скриншоты результатов тестирования

#define QMAX 100

struct queue {

int qu[QMAX];

int rear, frnt;

};

void init(struct queue\* q) {

q->frnt = 1;

q->rear = 0;

return;

}

void insert(struct queue\* q, int x) {

if (q->rear < QMAX - 1) {

q->rear++;

q->qu[q->rear] = x;

}

else

printf("Очередь полна!\n");

return;

}

void print(struct queue\* q) {

int h;

if (isempty(q) == 1) {

printf("Очередь пуста!\n");

return;

}

for (h = q->frnt; h <= q->rear; h++)

printf("%d ", q->qu[h]);

return;

}

int remove(struct queue\* q) {

int x;

if (isempty(q) == 1) {

printf("Очередь пуста!\n");

return(0);

}

x = q->qu[q->frnt];

q->frnt++;

return x;

}

int isempty(struct queue\* q) {

if (q->rear < q->frnt) return 1;

else return 0;

}

int main() {

struct queue\* q;

int a;

system("chcp 1251");

system("cls");

q = (queue\*)malloc(sizeof(queue\*));

init(q);

print(q);

for (int i = 0; i < 8; i++) {

printf("Введите элемент очереди: ");

scanf("%d", &a);

insert(q, a);

}

printf("\n");

print(q);

while (q->frnt != NULL) {

a = remove(q);

printf("\nУдален элемент %d\n", a);

print(q);

}

getchar(); getchar();

return 0;

}

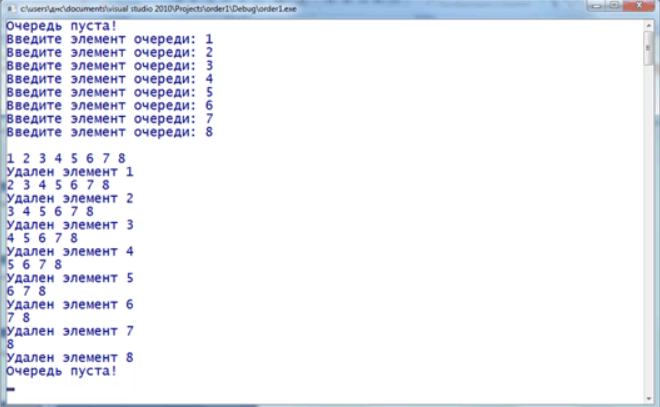


Рис. 7 - скриншот результатов тестирования

Вывод: мы получили знания и практические навыки по

разработке и реализации рекурсивных процессов.