**Содержание**

[ВВЕДЕНИЕ 3](#_Toc501697476)

[1. СОЗДАНИЕ ИНФОРМАЦИОННО-ПОИСКОВОГО СПРАВОЧНИКА 4](#_Toc501697477)

[1. Постановка задачи 4](#_Toc501697478)

[2. Структура данных с указанием типов и глобальных констант 4](#_Toc501697479)

[3. Общая структурная схема программы с указанием прототипов отдельных функций и объяснением их назначения, и указанием названия модулей их хранения 7](#_Toc501697480)

[4. Описание специальных алгоритмов 8](#_Toc501697481)

[5. Описание назначения отдельных функций 12](#_Toc501697482)

[6. Результаты работы программы на примере конкретных данных 37](#_Toc501697483)

[ЗАКЛЮЧЕНИЕ 47](#_Toc501697484)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 48](#_Toc501697485)

## ВВЕДЕНИЕ

Цель, которая ставится перед нами в ходе выполнения курсовой работы по дисциплине «Информатика» - это закрепление объёма знаний, как теоретических, так и практических, которые мы получить во время лекций и проводимых тематических лабораторных работ. За три семестра мы получили достаточно большое количество информации и данная курсовая работа призвана помочь нам систематизировать его, достичь не только поставленных целей, но и изучить новые способы решения прикладных задач. Этого можно достичь только совмещая получение теоретических знаний с развитием практических навыков. Поэтому для достижения поставленной цели необходимо будет обратиться и к дополнительной литературе.

Как уже говорилось выше, главной целью курсовой работы является закрепление знаний полученных при изучении дисциплины. Для достижения необходимого результата необходимо решить ряд поставленных перед нами задач, которые вошли в одну программу, вобравшую в себя всю полноту полученных ранее знаний. При разработке приложения были использованы языки программирования *С*. Задачей курсовой работы является разработка и реализация на языке *С* информационно-поискового справочника резисторов

## СОЗДАНИЕ ИНФОРМАЦИОННО-ПОИСКОВОГО СПРАВОЧНИКА

### Постановка задачи

В данной работе нужно разработать и реализовать на языке *C* информационно-поисковый справочник резисторов.

При запуске программы на выполнение, проанализировать командную строку и если в ней указано имя существующего файла данных, то его и обрабатывать в справочнике. В программе должно появляться числовое или буквенное меню (в зависимости от аргументов в командной строке), посредством которого должны вызываться следующие пункты, реализованные в программе как отдельные функции:

1. Создание(формирование) справочника данных (вывод данных) и сохранение справочника данных в файле данных с запросом создаваемого имени.

2. Вывод справочника на экран (чтение из файла по одной записи и ее вывод).

3. Добавление данных в конец справочника.

4. (\*) Добавление данных после записи с указанным номером.

5. (\*) Удаление данных с указанным номером.

6. Корректировка данных с указанным номером.

7. Сортировка методом пузырька по полю строкового типа (в определенном порядке).

8. Сортировка методом выбора или вставок по числовому полю (в любом порядке).

9. (\*) Сортировка методом вставок по текстовому полю.

10. (\*) Сортировка методом вставки по числовому полю (в любом порядке).

11. (\*) Сортировка методом Шелла по числовому полю(в любом порядке).

12. (\*) Сортировка методом быстрой сортировки по текстовому полю(по убыванию).

13. (\*\*)Реализация быстрой сортировки по любому указанному пользователем полю.

14. Выход из программы.

Хранение данных должно быть реализовано с помощью связных списков. Все данные передавать как параметры в отдельные функции. В функциях сортировки запрашивать направление сортировки. Создать собственный заголовочный файл, включающий директивы препроцессора, определение структуры данных , прототипы используемых функций.

Каждая запись должна состоять из 4 полей: наименование (строковый тип); остальные 4 поля выбираются самостоятельно из справочников и они представляют реальные параметры радиодеталей(компараторов). При работе с программой также ввести реальные марки и соответствующие им параметры деталей.

1. **Структура данных с указанием типов и глобальных констант**

Структура - это совокупность нескольких переменных, как правило, разных типов, сгруппированных под единым именем.

Структура в данной работе объявляется следующим образом:

typedef struct resistor {

char name[40];

float power;

int resistance;

double admission;

struct resistor \*next;

struct resistor \*prev;

} resistor;

typedef struct \_DblLinkedList {

size\_t size;

resistor \*head;

resistor \*tail;

}DblLinkedList;

Указатель prev хранит адрес предыдущего узла, если его нет (значит, это первый узел) то переменная равна NULL. Аналогично, указатель next хранит адрес следующего узла. **Значение полей структуры**:

name - название резистора, power- номинальная мощность(Вт), resistance - номинальное сопротивление(Ом), admission – допуск(%), next - храни адрес следующего узла, prev - хранит адрес предыдущего узла, если его нет (значит, это первый узел) то переменная равна NULL.

Структура "Двусвязный Список" будет хранить свой размер (чтобы не пересчитывать количество элементов каждый раз), а также указатель head, ссылающийся на первый элемент, и указатель tail, ссылающийся на последний элемент.

Эти части структуры называются полями. В общем случае структура объявляется следующим образом:

*struct* тег\_структуры {

тип\_поля\_1 имя\_поля\_1 ;

тип\_поля\_*n* имя\_поля\_*n*;

};

Файл *my.h* является отдельным заголовочным файлом, в который записываются структуры, прототипы функций и директивы препроцессора которые будут использованы в программе. Пример заголовочного файла который будет использоваться в моей программе представлен ниже, в листинге заголовочного.

**Листинг заголовочного файла**

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

typedef struct resistor {

char name[40];

float power;

int resistance;

double admission;

struct resistor \*next;

struct resistor \*prev;

} resistor;

typedef struct \_DblLinkedList {

size\_t size;

resistor \*head;

resistor \*tail;

}DblLinkedList;

resistor\* decompression(DblLinkedList \*list);

void compression(DblLinkedList \*list);

void pushBack(DblLinkedList \*list, char\* fileName);

void load(DblLinkedList \*list, char\* fileName);

void sort1(DblLinkedList \*list, char\* fileName);

resistor\* selectionSort(resistor \*head);

void ShellSort(DblLinkedList \*list, char\* fileName);

void save(DblLinkedList \*list, char\* fileName);

void deleteNth1(DblLinkedList \*list, size\_t index);

void update(DblLinkedList \*list, char\* fileName);

resistor\* getNth(DblLinkedList \*list, size\_t index);

int pointer\_compare(void \*location1, void \*location2);

void bubbleSort(DblLinkedList \*list, char\* fileName);

void insertionSort(DblLinkedList \*list, char\* fileName);

void\* popFront(DblLinkedList \*list);

void makeNewDataDirectory(DblLinkedList \*list, char\* fileName);

DblLinkedList\* createDblLinkedList();

void removeList(DblLinkedList \*\*list, char\* fileName);

void insert(DblLinkedList \*list, char \*fileName);

void deleteNth(DblLinkedList \*list, char\* fileName);

void pop(resistor \*\*head);

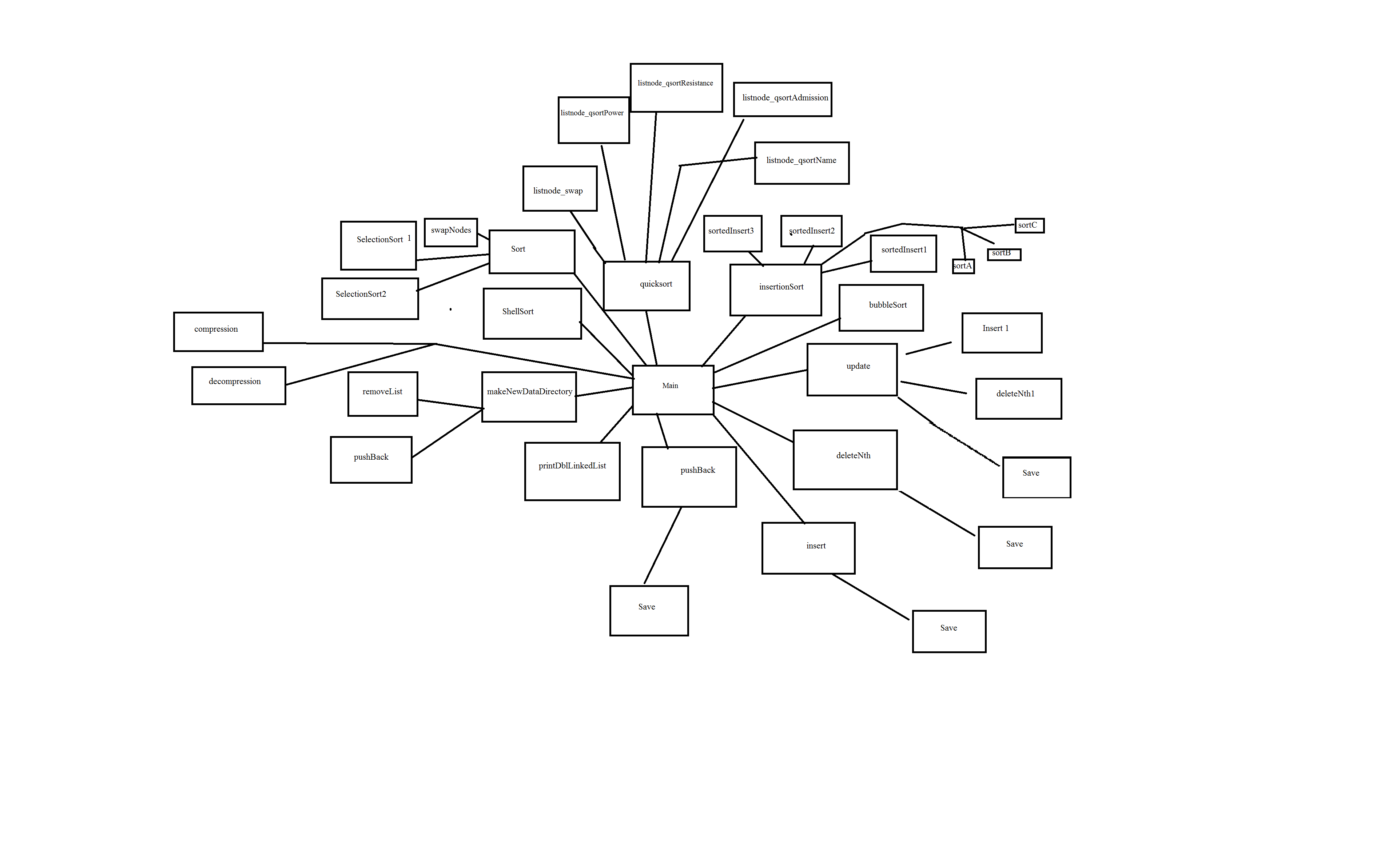
void printDblLinkedList(DblLinkedList \*list, char\* fileName);

void fastRandom(DblLinkedList \*list, char\* fileName);

void quicksort(resistor \* left, resistor \* right);

void swap(resistor \*a, resistor \*b);

1. **Общая структурная схема программы с указанием прототипов отдельных функций и объяснением их назначения, и указанием названия модулей их хранения**



void makeNewDataDirectory(DblLinkedList \*list, char\* fileName) - функция для создания (формирования) справочника данных (Вывод данных) и сохранение справочника данных в файле данных в конец справочника, хранится в модуле makeNewDataDirectory.c

void printDblLinkedList(DblLinkedList \*list, char\* fileName) - функция для вывода справочника на экран (чтение из файла по одной записи и ее вывод), хранится в модуле printDblLinkedList.c

void pushBack(DblLinkedList \*list, char\* fileName) - функция для добавление данных в конец справочника, хранится в модуле pushBack.c

void insert(DblLinkedList \*list, char \*fileName) - функция для добавления данных после записи с указанным номером, хранится в файле insert.c

void deleteNth(DblLinkedList \*list, char\* fileName) - функция для удаления данных с указанным номером, хранится в файле deleteNth.с.

void update(DblLinkedList \*list, char\* fileName) - функция для корректировки данных с указанным номером, хранится в модуле update.c.

void bubbleSort(DblLinkedList \*list, char\* fileName) - функция для сортировки методом пузырька по полю строкового типа (в определенном порядке), хранится в файле bubbleSort.c

void insertionSort(DblLinkedList \*list, char\* fileName) - сортировка методом вставки по числовому полю (в любом порядке), хранится в модуле insertionSort.c

void quicksort(resistor \* left, resistor \* right) - функция для сортировки методом быстрой сортировки по текстовому полю(по убыванию), хранится в модуле quicksort.c.

void sort1(DblLinkedList \*list, char\* fileName) - функция сортировки методом выбора по числовому полю (в любом порядке), selectionSort.c

void ShellSort(DblLinkedList \*list, char\* fileName) - функция сортировки методом Шелла по числовому полю(в любом порядке), хранится в модуле ShellSort.c

DblLinkedList\* createDblLinkedList() - функция создает экземляр структуры DblLinkedList, хранится в модуле createDblLinkedList.с

void removeList(DblLinkedList \*\*list, char\* fileName) - функция, которая удаляет список, находится в модуле DeleteDoubleLinkedList.c

void compression(DblLinkedList \*start) – функция, предназначенная для компресосования информацию методом арифмитического кодирования хранится в модуле compression.c;

resistor \*decompression(DblLinkedList \*start) – функция, предназначенная для декодирования информацию методом арифмитического кодирования хранится в модуле decompression.c;

### Описание специальных алгоритмов

В данной программе используются следующие специальные алгоритмы: сортировка методом выбора, сортировка пузырьком, сортировка методом вставки, сортировка методом Шелла.

* + 1. **Сортировка методом выбора**

**Метод выбора** - это наиболее естественный алгоритм упорядочивания. При данной сортировке из массива выбирается элемент с наименьшим значением и обменивается с первым элементом. Затем из оставшихся n - 1 элементов снова выбирается элемент с наименьшим ключом и обменивается со вторым элементом, и т.д.

**Алгоритм метода**

Алгоритм сортировки методом выбора на основе извлечения минимального элемента (сортировка по убыванию):

1. найдем минимальный элемент среди всех элементов массива , , …, и определим его номер. Пусть это будет элемент ;

2. поменяем местами элементы и . Таким образом, минимальный элемент массива окажется на своем окончательном месте;

3. теперь найдем минимальный элемент среди элементов массива, начиная со второго , , …, и определим его номер. Опять обозначим этот элемент ;

4. поменяем местами элементы и . В результате два первых элемента массива окажутся на своих окончательных местах;

5. на заключительном этапе этого процесса надо выбрать минимальный элемент из двух последних элементов массива и . После чего этот элемент должен быть поставлен на предпоследнее место.

**Пример**

void Sort\_vibor(float\* Mass, int N)

{

float temp;

int i, k, i\_min;

for (k=0; k<N-1; k++)

{i\_min=k;

for (i=k+1; i<N; i++)

if(Mass[i]<Mass[i\_min]) i\_min=i;

// обмен

temp=Mass[k];

Mass[k]=Mass[i\_min];

Mass[i\_min]=temp;

}

return ;

}

* + 1. **Сортировка методом пузырька**

**Сортировка пузырьком** ( bubble sort) — простой алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: O(n²).

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим наибольшим элементом, а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма).

**Алгоритм метода**

Упорядочим заданный целочисленный массив по неубыванию на основе алгоритма "пузырька", относящегося к алгоритмам обменов.

1. сравним элементы и . Если не выполняется условие ,то меняем местами эти элементы и сравниваем элементы и . Так сравниваем и при необходимости меняем местами все пары исходного массива. Последней рассматривается пара и . (всплыл первый по величине пузырек);

2. второй просмотр массива начинается со сравнения элементов и . Последней рассматривается пара и . В результате на месте элемента и окажется второй по величине элемент массива (всплыл второй по величине пузырек);

3. третий просмотр массива начинается с проверки пары и , заканчивается проверкой пары и , на месте элемента окажется третий по величине элемент массива;

4. при последнем просмотре сравниваются элементы и

**Пример**

void Sort\_obmen\_puz(float\* Mass, int N)

{

float temp;

int i,k;

for (k=N-1; k>0 ;k--) // цикл проходов

{for(i=0; i<k; i++) // цикл для пузырька

{if (Mass[i]>Mass[i+1])

{temp=Mass[i];

Mass[i]=Mass[i+1];

Mass[i+1]=temp;

}

}

}

return ;

}

**1.4.3 Сортировка методом вставок**

Сортировка вставками — простой алгоритм сортировки. Уступает в эффективности более сложным (таким как быстрая сортировка), но имеет ряд преимуществ:

- эффективен на небольших наборах данных, на наборах данных до десятков элементов может оказаться лучшим;

- эффективен на наборах данных, которые уже частично отсортированы;

- это устойчивый алгоритм сортировки (не меняет порядок элементов, которые уже отсортированы);

- может сортировать список по мере его получения;

- использует O(1) временной памяти, включая стек.

Недостаток - высокая сложность алгоритма: O(n²).

**Пример**

void Sort\_insert(float\* Mass, int N)

{ float temp;

int i,j;

for (i = 1; i < N; i++) {

j = i;

while (j > 0 && Mass[j - 1] > Mass[j]) {

temp = Mass[j];

Mass[j] = Mass[j - 1];

Mass[j - 1] = temp;

j--;

}

}

return ;

}

**1.4.4 Сортировка методом Шелла**

Данный алгоритм сортировки, являющийся усовершенствованным вариантом сортировки методом вставок. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.

**Алгоритм**

Сначала в исходной последовательности сортируются между собой элементы, отстоящие друг от друга на расстоянии n/2 элементов, затем на расстоянии n/4 и т.д. до тех пор пока не получим 2 последовательности, элементы которых отстоят друг от друга на расстоянии 1-го элемента. После этого делаем сортировку этой полученной последовательсти выбранным методом и на выходе имеем уже полностью отсортированную последовательность.

**Пример**

void shell (float \*Mass, int N) {

register int i, j, k; float h, x;

char a[5]={9,5,3,2,1};

for( k = 0; k < 5; k++) {

h = a[k];

for( i = h; i < N; ++i ) {

x = Mass [ i ];

for(j = i-h; (x < Mass [j]) && (j >= 0); j = j-h)

Mass [ j+h ] = Mass [ j ];

Mass[ j+h ] = x;

}

}

}

**1.4.5 Быстрая сортировка**

Быстрая сортировка (англ. quicksort)— широко известный алгоритм сортировки, разработанный английским информатиком Чарльзом Хоаром в 1960 году. Один из быстрых известных универсальных алгоритмов сортировки массивов (в среднем O(n log n) обменов при упорядочении n элементов), хотя и имеющий ряд недостатков.

**Алгоритм**

1. Два индекса — l и r, приравниваются к минимальному и максимальному индексу разделяемого массива соответственно.

2. Вычисляется индекс опорного элемента (компаранд) m.

3. Индекс l последовательно увеличивается до m до тех пор, пока l-й элемент не превысит компаранд.

4. Индекс r последовательно уменьшается до m до тех пор, пока r-й элемент не окажется меньше либо равен компаранду.

5. Если r = l — найдена середина массива — операция разделения закончена, оба индекса указывают на компаранд.

6. Если l < r — найденную пару элементов нужно обменять местами и продолжить операцию разделения с тех значений l и r, которые были достигнуты. Следует учесть, что если какая-либо граница (l или r) дошла до опорного элемента, то при обмене значение m изменяется на r-й или l-й элемент соответственно.

7. Рекурсивно упорядочиваем подмассивы, лежащие слева и справа от опорного элемента.

Базой рекурсии являются наборы, состоящие из одного или двух элементов. Первый возвращается в исходном виде, во втором, при необходимости, сортировка сводится к перестановке двух элементов. Все такие отрезки уже упорядочены в процессе разделения.

**Пример**

void quick(float \*items, int count) {

qs(items, 0, count-1);

}

/\* Быстрая сортировка. \*/

void qs(float \*items, int left, int right{

register int i, j;

float x, y; i = left; j = right;

x = items[(left+right)/2]; /\* выбор компаранда \*/

do {

while((items[i] < x) && (i < right)) i++;

while((x < items[j]) && (j > left)) j--;

if(i <= j) {

y = items[i];

items[i] = items[j];

items[j] = y;

i++; j--;

}

}

while(i <= j);

if(left < j) qs(items, left, j);

if(i < right) qs(items, i, right);

}

### Описание назначения отдельных функций

В программе используются сдедующие файлы: deleteNth.c, selectionSort.c, bubbleSort.c, getNth.c, pop.c, shellSort.c, compare.c, insert.o printList.c, someRandomFunction.c, insertionSort.c, pushBack.c, swap.c, main.c, push.c, update.c, qsort.c, create.c, read.c, write.c.

main.с – главный модуль, к которому подключаются пользовательские функции. При вызове программы возможно указать максимум 2 аргумента ( 1 – вызов самой программы, 2 – имя файла в который будут записаны данные после вызова функции сохранения ). При указании более 2 аргументов, программа завершит свою работу. При указании 1 аргумента, программа запусти по умолчанию числовое меню.

#include "my.h"

int cycle = 1;

int i = 1;

int main(int argc, char \*argv[]) {

char fileName[24];//name of database

DblLinkedList \*list = createDblLinkedList();

if (argc == 1) {

printf("Enter the filename with database: ");

scanf("%s", fileName);

while(getchar() != '\n') continue;

load(list, fileName);

} else if(argc > 2) {

printf(“Error!\n”);

return 0;

} else {

strcpy(fileName, argv[1]);

load(list, fileName);

}

while(cycle == 1) { //start

int a1 = 124;

int a2 = 45;

int mainNumber = 0; //choice number of menu

for(int i = 0; i < 56; i++)printf("%c",a2);printf("\n");

printf("%c Reference resistor: %c\n",a1,a1);

printf("%c %c\n",a1,a1);

printf("%c 1 - Make new data director %c\n",a1,a1);

printf("%c 2 - Print all %c\n",a1,a1);

printf("%c 3 - Append resistor %c\n",a1,a1);

printf("%c 4 - Insert resistor %c\n",a1,a1);

printf("%c 5 - Delete from data base %c\n",a1,a1);

printf("%c 6 - Update some resistor %c\n",a1,a1);

printf("%c 7 - Sorting a bubble over a field of a string type %c\n",a1,a1);

printf("%c 8 - InsertionSort on a num and str field%c\n",a1,a1);

printf("%c 9 - Quick sort by any field %c\n",a1,a1);

printf("%c 10 - Selection sort to an num (in the certain order) %c\n",a1,a1);

printf("%c 11 - ShellSort an a numeric field (in any order). %c\n",a1,a1);

printf("%c 12 - Compression %c\n",a1,a1);

printf("%c 13 - Decompression %c\n",a1,a1);

printf("%c 0 - Exit %c\n",a1,a1);

for(int i = 0; i < 54; i++)printf("%c",a2);printf("\n");

scanf("%d", &mainNumber);

switch(mainNumber) {

case 1: {

makeNewDataDirectory(list, fileName);

break;

} case 2: {

printDblLinkedList(list, fileName);

break;

} case 3: {

pushBack(list, fileName);

break;

} case 4: {

insert(list, fileName);

break;

} case 5: {

deleteNth(list, fileName);

break;

} case 6: {

update(list, fileName);

break;

} case 7: {

bubbleSort(list, fileName);

break;

} case 8: {

insertionSort(list, fileName);

break;

} case 9 :{

quicksort(list->head, list -> tail);

break;

} case 10 :{

sort1(list, fileName);

break;

} case 11: {

ShellSort(list, fileName);

break;

} case 12: {

compression(list);

break;

} case 13: {

decompression(list);

break;

} case 0: {

cycle = 0;

break;

} default: {

printf("Error!\n\n");

break;

}

}

}

return 0;

}

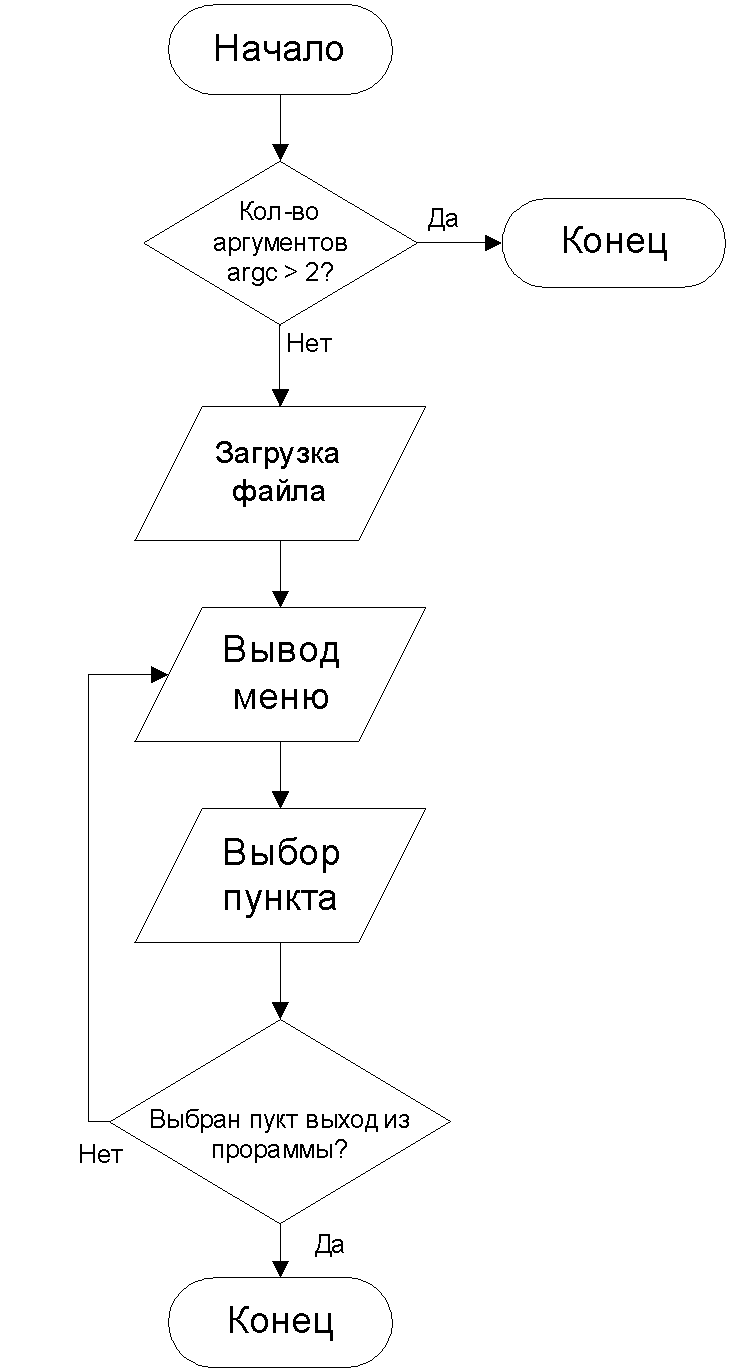


Рисунок 1 – Блок – схема функции “*main.c”.*

makeNewDataDirectory.c – модуль, в котором содержится функция, реализующая ввод данных в базу данных справочника, необходимых для дальнейшей работы с программой.

Если база данных не пуста, то выводит слудующее меню: 1 – ввод новых данных, 2 – добавление данных, 0 – назад. При выборе пункта 1, старая база данных удалиться и будет вводиться заново. При выборе пункта 2, будет добавляться в конец старой базы данных. При выборе пункта 0, программа врнется в главное меню.

При вводе значений параметров компараторов имеется защита от некорректных данных в виде символов так как вводимые значения должны являться числовыми.

#include "my.h"

void makeNewDataDirectory(DblLinkedList \*list, char\* fileName) {

int k;

if(list->head != NULL) {

while(1) {

printf("You already have a database! Are you sure you want to overwrite it? \n1. remove database\n2. Continue add");

while(scanf("%d", &k) != 1)

{

printf("Must be non-negative integer!\n");

while(getchar() != '/n') continue;

}

if(k == 1)

{

removeList(&list, fileName);

break;

}

else if(k == 2){ break; }

else

{

printf("The number can't be less than 1 and greater than 2\n");

continue;

}

}

}

pushBack(list, fileName);

}

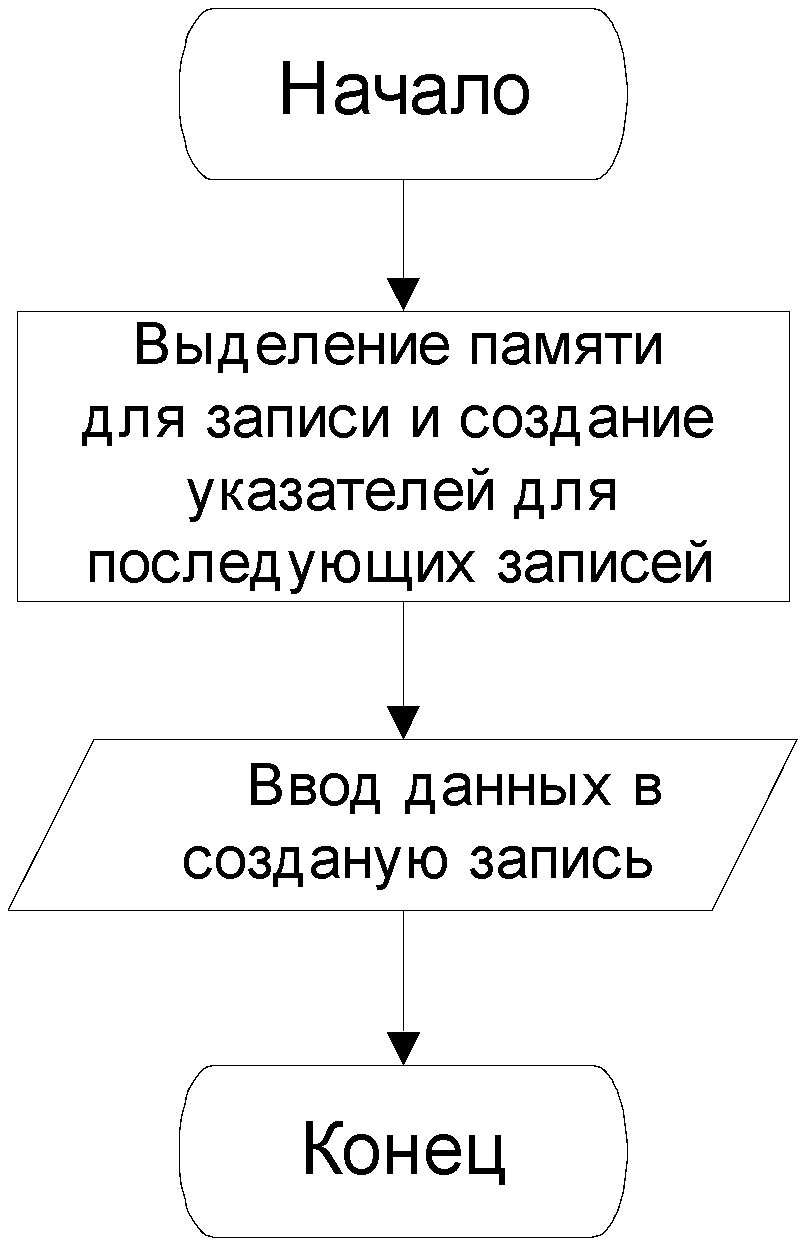


Рисунок 2 – Блок – схема функции “makeNewDataDirectory.c”

printList.c – модуль, содержащий функцию которая позволяет выводить текущее состояние базы данных на экран.

#include "my.h"

void printDblLinkedList(DblLinkedList \*list, char\* fileName) {

read(list, fileName);

int a1 = 124;

int a2 = 45;

int i = 0;

resistor \*tmp = list->head;

if(tmp == NULL) printf(“Database is empty/n”);

printf("%c",a1);for(int a = 0; a < 103; a++)printf("%c",a2);printf("%c\n",a1);

printf("%c\t№\t%c\tName\t%c\tPower(Vt)\t%c\tResistance(hOm)\t%c\tAdmission(%)\t%c\n",a1,a1,a1,a1,a1,a1,a1);

printf("%c",a1);for(int a = 0; a < 103; a++)printf("%c",a2);printf("%c\n",a1);

while (tmp) {

printf("|%-15d",i++);

printf("|%-15s",tmp->name);

printf("|%-23f",tmp->power );

printf("|%-23d",tmp->resistance);

printf("|%-23lf|",tmp->admission);

printf("\n%c",a1);for(int a = 0; a < 103; a++)printf("%c",a2);printf("%c\n",a1);

tmp = tmp->next;

}

printf("\n\n");

}

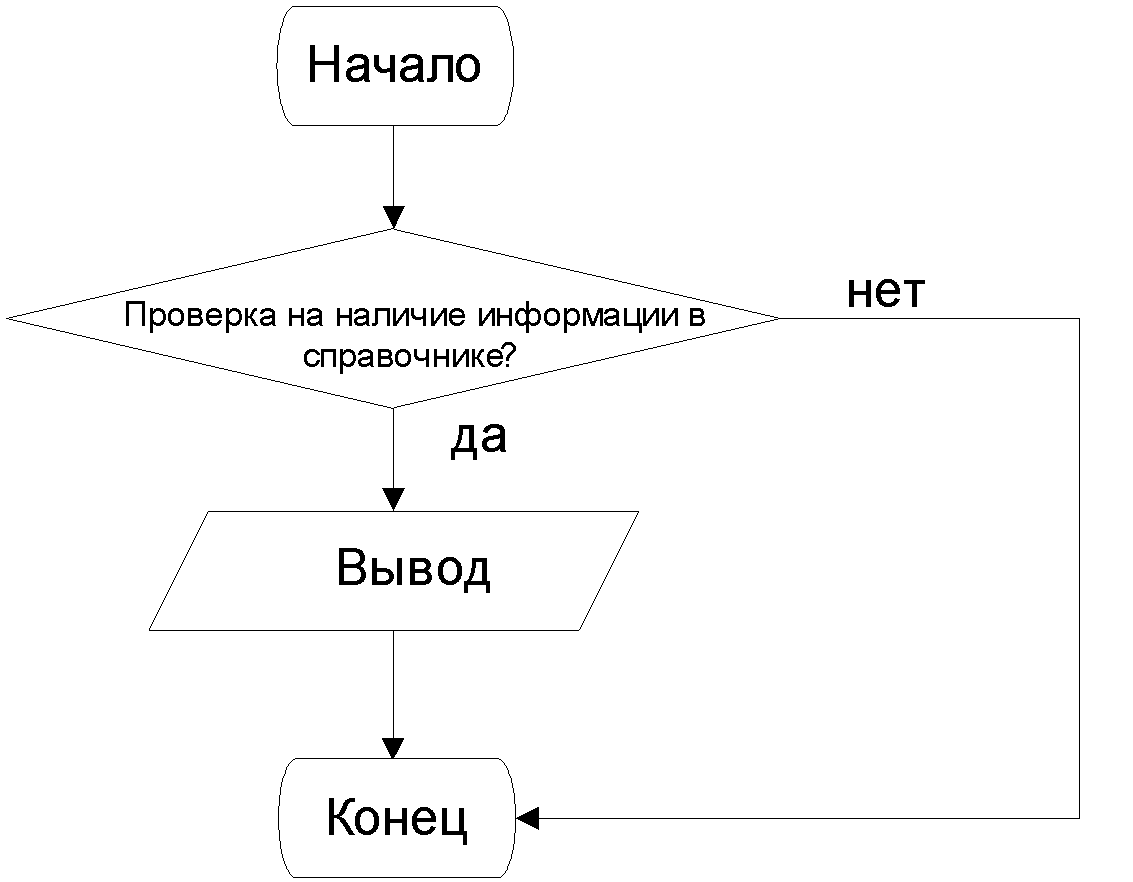


Рисунок 3 – Блок – схема функции “printList.c”

pushBack.c – модуль, который содержит функцию вставки в конец списка.

#include "my.h"

void pushBack(DblLinkedList \*list, char\* fileName) {

int stop = 2;

while(stop == 2){

resistor \*tmp = (resistor\*) malloc(sizeof(resistor));

if (tmp == NULL) {

break;

}

printf("Resistor %d:\n",list->size);

printf("resistance: ");

while (scanf("%d", &tmp->resistance) != 1) {

printf("Incorrect value!\n");

while(getchar() != '\n')continue;

continue;

}

printf("name: ");

while (scanf("%s",&tmp->name) != 1) {

printf("Incorrect value!\n");

while(getchar() != '\n')continue;

continue;

}

printf("addmission: ");

while (scanf("%lf", &tmp->admission) != 1) {

printf("Incorrect value!\n");

while(getchar() != '\n')continue;

continue;

}

printf("power: ");

while (scanf("%f", &tmp->power) != 1) {

printf("Incorrect value!\n");

while(getchar() != '\n')continue;

continue;

}

tmp->next = NULL;

tmp->prev = list->tail;

if (list->tail) {

list->tail->next = tmp;

}

list->tail = tmp;

if (list->head == NULL) {

list -> head = tmp;

}

list->size++;

save(list, fileName);

while(1) {

printf("Do you want to stop ?(1-y, 2-n)");

scanf("%d", &stop);

while(getchar() != '\n'){

continue;

}

if(stop==1 || stop==2){

break;

}else{

printf("Wrong!\n");

continue;

}

}

printf("\n");

}

}

insert.c – модуль, содержащий пользовательскую функцию, реализующую добавление данных в базу данных после записи с указанным номером.

#include "my.h"

void insert(DblLinkedList \*list, char \*fileName) {

int index;

resistor \*elm = NULL;

resistor \*ins = NULL;

printf("Enter the index of element:");

scanf("%d", &index);

elm = getNth(list, index);

if (elm == NULL) {

exit(5);

}

ins = (resistor\*) malloc(sizeof(resistor));

printf("Resistor %d:\n", index);

printf("resistance: ");

scanf("%d", &ins->resistance);

printf("name: ");

scanf("%s", &ins->name);

printf("addmission: ");

scanf("%lf", &ins->admission);

printf("power: ");

scanf("%f", &ins->power);

ins->prev = elm;

ins->next = elm->next;

if (elm->next) {

elm->next->prev = ins;

}

elm->next = ins;

if (!elm->prev) {

list->head = elm;

}

if (!elm->next) {

list->tail = elm;

}

list->size++;

save(list, fileName);

}

} 

Рисунок 4 – Блок – схема функции «insert.c»

deleteNth.c – модуль, в котором есть функция, которая позволяет пользователю удалить определенную структуру в базе данных. При случае ввода номера, которого нет, или при вводе символов, вместо чисел, сработает защита, от ввода некорректных данных.

#include "my.h"

void deleteNth(DblLinkedList \*list, char\* fileName) {

resistor \*elm = NULL;

int index = 0;

printf("Enter the index of element:");

scanf("%d", &index);

elm = getNth(list, index);

if (elm == NULL) {

exit(5);

}

if (elm->prev) {

elm->prev->next = elm->next;

}

if (elm->next) {

elm->next->prev = elm->prev;

}

if (!elm->prev) {

list->head = elm->next;

}

if (!elm->next) {

list->tail = elm->prev;

}

free(elm);

list->size--;

save(list, fileName);

}

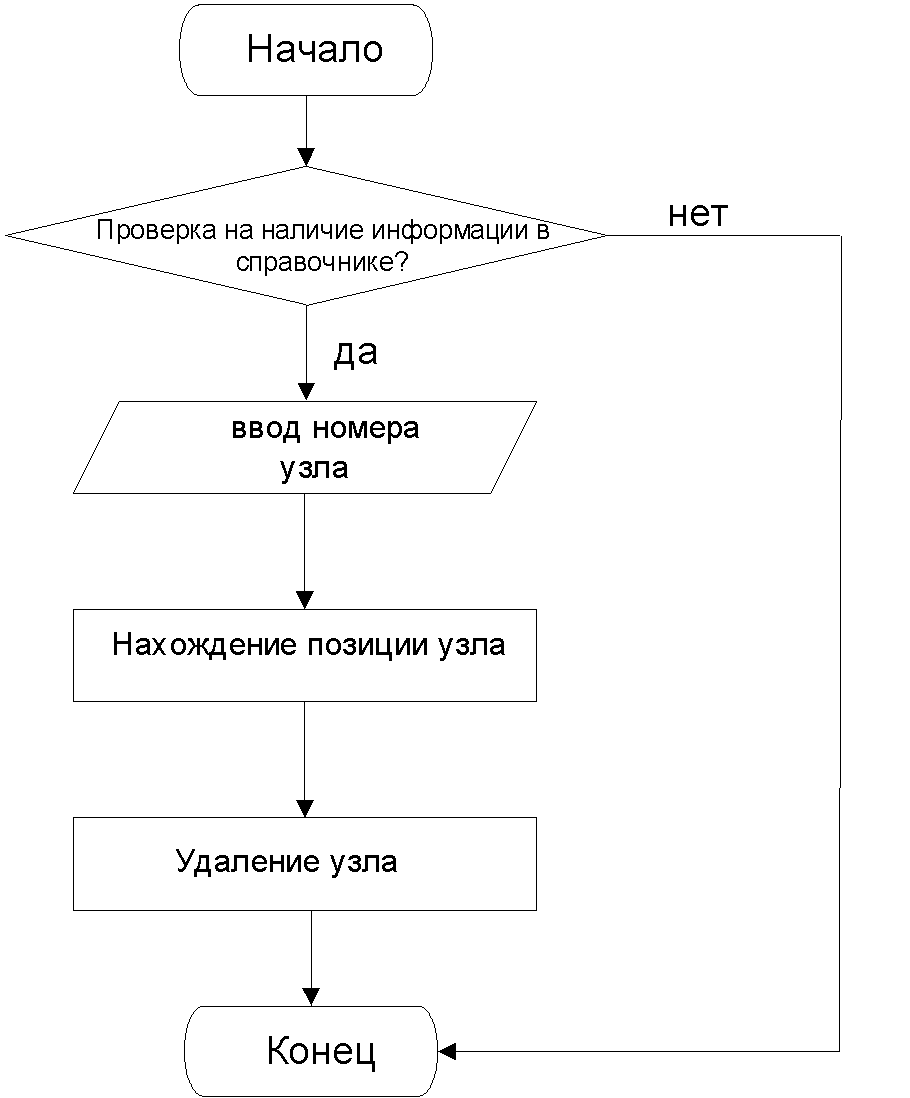


Рисунок 5 – Блок – схема функции “deleteNth.c”

update.c –модуль, в котором есть пользовательская функция, которая служит для того чтобы изменить текущие данных лежащие в определенной структуре.

#include "my.h"

void insert1(DblLinkedList \*list, size\_t index) {

resistor \*elm = NULL;

resistor \*ins = NULL;

elm = getNth(list, index);

if (elm == NULL) {

exit(5);

}

ins = (resistor\*) malloc(sizeof(resistor));

printf("Resistor %d:\n", index+1);

printf("resistance: ");

scanf("%d", &ins->resistance);

printf("name: ");

scanf("%s", &ins->name);

printf("addmission: ");

scanf("%lf", &ins->admission);

printf("power: ");

scanf("%f", &ins->power);

ins->prev = elm;

ins->next = elm->next;

if (elm->next) {

elm->next->prev = ins;

}

elm->next = ins;

if (!elm->prev) {

list->head = elm;

}

if (!elm->next) {

list->tail = elm;

}

list->size++;

}

void deleteNth1(DblLinkedList \*list, size\_t index) {

resistor \*elm = NULL;

elm = getNth(list, index);

if (elm == NULL) {

exit(5);

}

if (elm->prev) {

elm->prev->next = elm->next;

}

if (elm->next) {

elm->next->prev = elm->prev;

}

if (!elm->prev) {

list->head = elm->next;

}

if (!elm->next) {

list->tail = elm->prev;

}

free(elm);

list->size--;

index--;

insert1(list, index);

}

void update(DblLinkedList \*list, char\* fileName) {

int n = 0;

printf("Enter number:");

scanf("%d", &n);

deleteNth1(list, n);//Функция для удаления списка

save(list, fileName);

}

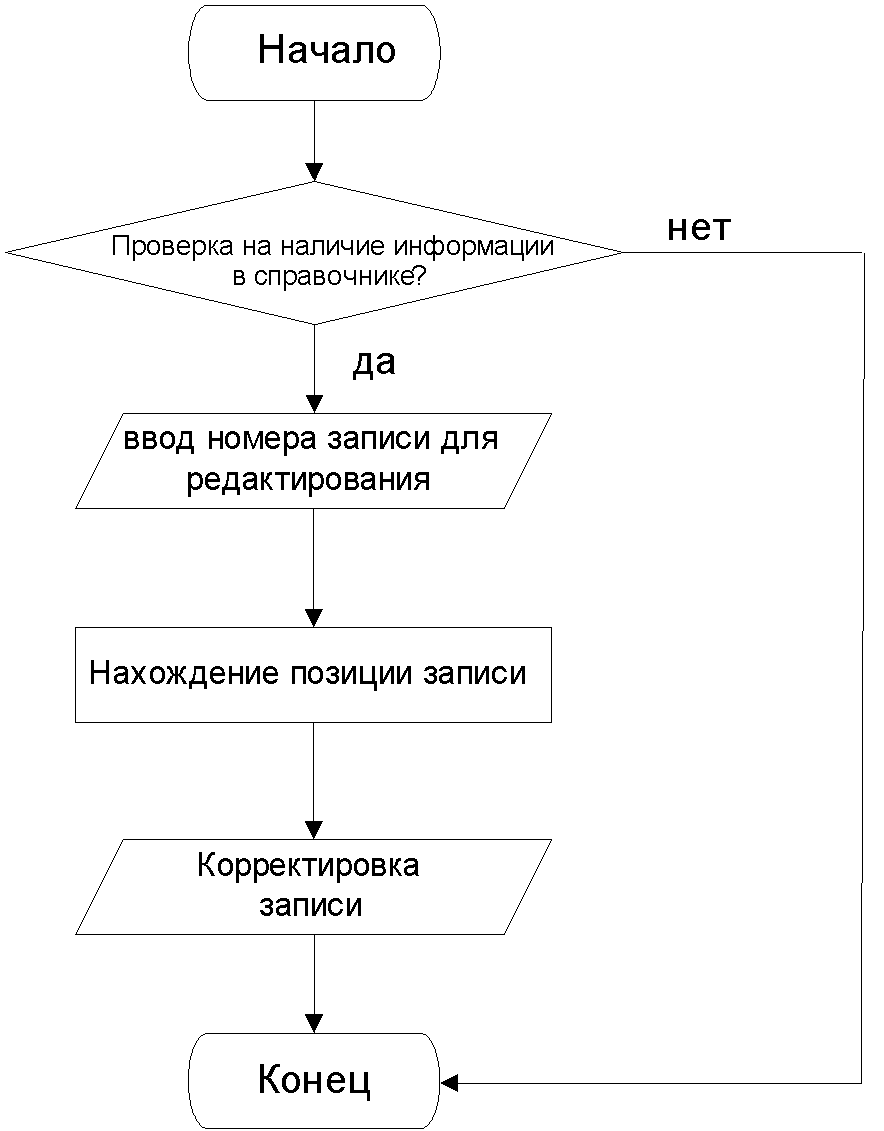


Рисунок 6 – Блок – схема функции “update.c”

write.c – модуль в содержащий функцию, которая записывает данные, находящиеся в базе данных в бинарный файл, имя которого задается пользователем при вызове самой программы либо при вызове этой функции.

#include "my.h"

void save(DblLinkedList \*list, char\* fileName){

FILE \*out = fopen(fileName,"wb");

resistor\* tmp = list->head;

while(tmp) {

fwrite(tmp, sizeof(resistor), 1, out);

tmp = tmp->next;

printf("Serialization was succsess!\n");

}

fclose(out);

}

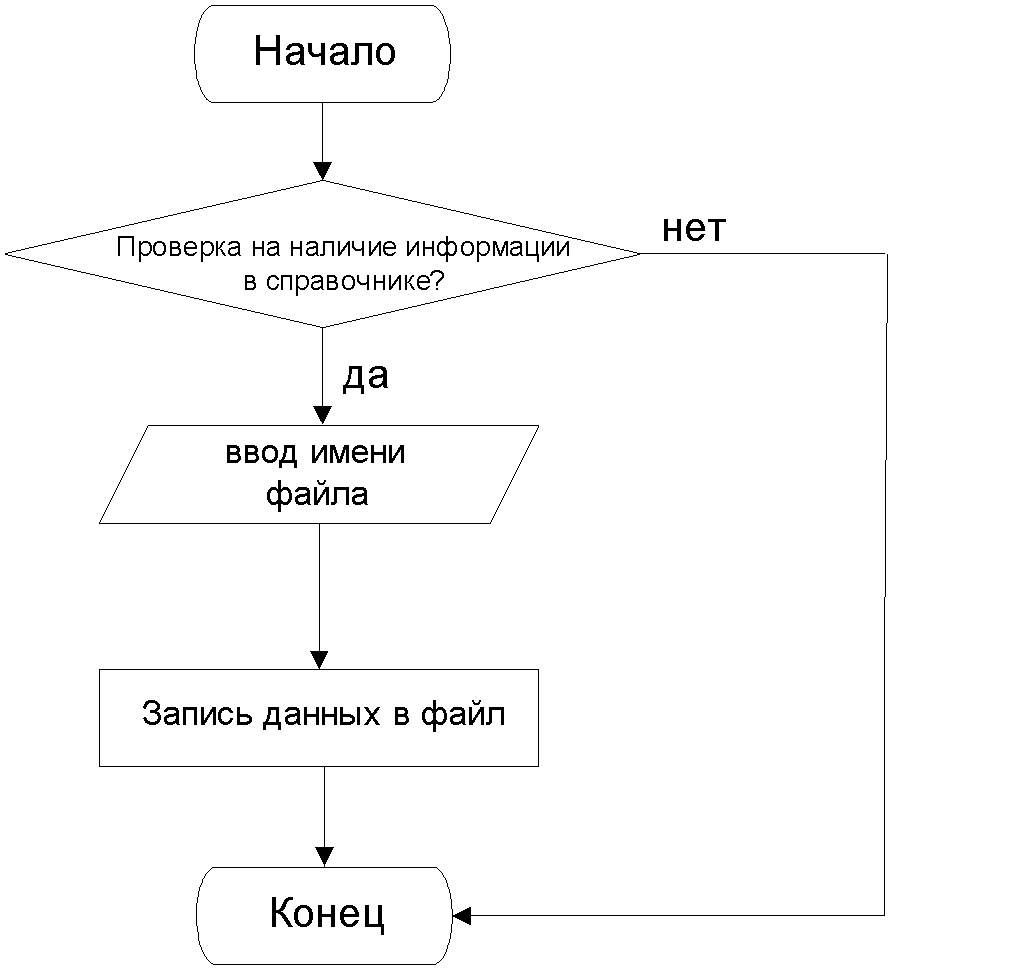


Рисунок 7 – Блок – схема функции “write.c”

read.c – модуль который содержит функцию, которая считывает данные из файла, имя которого задается пользователем при вызове самой функции.

#include "my.h"

void load(DblLinkedList \*list, char\* fileName){

double N\_prov, size;

long long N, i;

FILE \*in = fopen(fileName, "rb");

int countOfScanneredSymbols = 0;

fseek(in, 0, SEEK\_END);

size = ftell(in);

fseek(in, 0, SEEK\_SET);

N\_prov = size/sizeof(resistor);

N = N\_prov;

if(((double)N) != N\_prov) {

fclose(in);

system("clear");

printf("Data on file are not correct inputted!\n");

return;

}

if (N == 0) {

fclose(in);

system("clear");

printf("File is empty!\n");

return;

}

for(int i = 1; i <= N; i++) {

resistor \*tmp = (resistor\*) malloc(sizeof(resistor));

fread(tmp, sizeof(resistor), 1, in);

if (tmp == NULL) {

break;

}

tmp -> next = list -> head;

tmp -> prev = NULL;

if (list -> head) {

list -> head->prev = tmp;

}

list -> head = tmp;

if (list -> tail == NULL) {

list -> tail = tmp;

}

list -> size++;

}

fclose(in);

}

SelectionSort.c –модуль, в котором содержится сортировка методом выбора или вставок по числовому полю (в любом порядке).

#include "my.h"

void swapNodes(resistor\*\* head\_ref, resistor\* currX, resistor\* currY, resistor\* prevY) {

\*head\_ref = currY;

prevY->next = currX;

resistor\* temp = currY->next;

currY->next = currX->next;

currX->next = temp;

}

resistor\* selectionSort1(resistor \*head) { //сортировка по возрастанию

if (head->next == NULL)

return head;

resistor\* min = head;

resistor\* beforeMin = NULL;

resistor\* ptr;

for (ptr = head; ptr->next != NULL; ptr = ptr->next) {

if (ptr->next->power < min->power) {

min = ptr->next;

beforeMin = ptr;

}

}

if (min != head)

swapNodes(&head, head, min, beforeMin);//обмен узлами в списке

head->next = selectionSort1(head->next);

return head;

}resistor\* selectionSort2(resistor \*head) {//сортировка по убыванию

if (head->next == NULL)

return head;

resistor\* min = head;

resistor\* beforeMin = NULL;

resistor\* ptr;

for (ptr = head; ptr->next != NULL; ptr = ptr->next) {

if (ptr->next->power > min->power) {

min = ptr->next;

beforeMin = ptr;

}

}

if (min != head)

swapNodes(&head, head, min, beforeMin); //обмен узлами в списке

head->next = selectionSort2(head->next);

return head;

}

void sort1(DblLinkedList \*list, char\* fileName) { // функция для определения поля для сортировки выбором

int choice = 0;

resistor \*head\_ref = list->head;

if ((head\_ref) == NULL)

return;

while(1) {

printf("1.Sorts in ascending order\n2.Sorts in descending order!\n");

while(scanf("%d", &choice) != 1) {

printf("Must be non-negative integer!\n");

while(getchar() != '/n') continue;

}

if(choice < 1 || choice > 2){

printf("The number can't be less than 1 and greater than 2\n");

continue;

}

break;

}

switch(choice) {

case 1: {

head\_ref = selectionSort1(head\_ref);

break;

}

case 2: {

head\_ref = selectionSort2(head\_ref);

}

}

}

BubbleSort.с – модуль, в котором содержится сортировка методом пузырька по полю строкового типа (в определенном порядке).

#include "my.h"

void bubbleSort(DblLinkedList \*list, char\* fileName) {

int swapped, i;

resistor \*courrent;

resistor \*nextLeft = NULL;

if (courrent == NULL) return;

do {

swapped = 0;

courrent = list -> head;

while (courrent->next != nextLeft) {

if (strcmp(courrent->name, courrent->next->name) > 0) {

swap(courrent,courrent->next); //меняем местами узлы

swapped = 1;

}

courrent = courrent -> next;

}

nextLeft = courrent;

} while (swapped);

save(list, fileName);

}

insertionSort.c – модуль, в котором содержится сортировка методом вставок по текстовому и числовому полю(В любом порядке).

#include "my.h"

void sortedInsert1(resistor\*\* head\_ref, resistor\* new\_node) {

resistor\* current;

if (\*head\_ref == NULL || strcmp((\*head\_ref)->name, new\_node->name) >= 0) {

new\_node -> next = \*head\_ref;

\*head\_ref = new\_node;

} else {

current = \*head\_ref;

while (current -> next!=NULL && strcmp(current->next->name, new\_node->name) < 0) {

current = current -> next;

}

new\_node -> next = current->next;

current -> next = new\_node;

}

}

void sortedInsert2(resistor\*\* head\_ref, resistor\* new\_node) {

resistor\* current;

if (\*head\_ref == NULL || (\*head\_ref)->power >= new\_node->power) {

new\_node->next = \*head\_ref;

\*head\_ref = new\_node;

} else {

current = \*head\_ref;

while (current->next!=NULL && current->next -> power > new\_node->power) {

current = current -> next;

}

new\_node -> next = current->next;

current -> next = new\_node;

}

}

void sortedInsert3(resistor\*\* head\_ref, resistor\* new\_node) {

resistor\* current;

if (\*head\_ref == NULL || (\*head\_ref)->power <= new\_node->power) {

new\_node -> next = \*head\_ref;

\*head\_ref = new\_node;

} else {

current = \*head\_ref;

while (current->next!=NULL && current->next->power > new\_node->power) {

current = current->next;

}

new\_node -> next = current->next;

current -> next = new\_node;

}

}

void SortA(DblLinkedList \*list, char\* fileName) {//Сортирвка по названию

resistor \*head\_ref = list->head;

resistor \*sorted = NULL;

resistor \*current = head\_ref;

while (current != NULL) {

resistor \*next = current->next;

sortedInsert1(&sorted, current);

current = next;

}

head\_ref = sorted;

save(list, fileName);

}

void SortB(DblLinkedList \*list, char\* fileName){//сортировка по сопротивлению

resistor \*head\_ref = list->head;

resistor \*sorted = NULL;

resistor \*current = head\_ref;

while (current != NULL) {

resistor \*next = current->next;

sortedInsert2(&sorted, current);

current = next;

}

head\_ref = sorted;

save(list, fileName);

}

void SortC(DblLinkedList \*list, char\* fileName){//сортировка сопротивления по убыванию

resistor \*head\_ref = list->head;

resistor \*sorted = NULL;

resistor \*current = head\_ref;

while (current != NULL) {

resistor \*next = current->next;

sortedInsert3(&sorted, current);

current = next;

}

head\_ref = sorted;

save(list, fileName);

}

void insertionSort(DblLinkedList \*list, char\* fileName) {

int choice = 0;

while(1) {

printf("1.Sort by name\n2.Sort by resistance (ascending)\n3.Sort by resistance (descending)\n");

while(scanf("%d", &choice) != 1) {

printf("Must be non-negative integer!\n");

while(getchar() != '/n') continue;

}

if(choice < 1 || choice > 3){

printf("The number can't be less than 1 and greater than 2\n");

continue;

}

break;

}

switch(choice) {

case 1: {

SortA(list, fileName);

break;

} case 2: {

SortB(list, fileName);

break;

} case 3: {

SortC(list, fileName);

break;

}

}

}

ShellSort.c – модуль, в котором содержится сортировка методом Шелла по числовому полю (в любом порядке).

#include "my.h"

void ShellSort(DblLinkedList \*list, char\* fileName) {

int choice = 0;

resistor \*head\_ref = list->head;

// if list is empty

if ((head\_ref) == NULL)

return;

while(1) {

printf("1.Sorts in ascending order\n2.Sorts in descending order!\n");

while(scanf("%d", &choice) != 1) {

printf("Must be non-negative integer!\n");

while(getchar() != '/n') continue;

}

if(choice < 1 || choice > 2){

printf("The number can't be less than 1 and greater than 2\n");

continue;

}

break;

}

switch(choice) {

case 1: {

if(list->head) {

int step=0;

int lenght=0;

resistor \*p = list->head;

while(p) {

lenght++;

p=p->next;

}

while(2\*(3\*step+1) <= lenght) step=3\*step+1;

for(step; step>0 ;step/=3)

for(int i=step;i>0;i--)

for(int j=step-i; j<lenght; j+=step) {

p=list->head;

int k;

for(k=0;k<j;k++) p=p->next;

resistor\* c=p;

int temp=k+step;

while(c) {

for(k;k<temp;)

if(c) {

k++;

c=c->next;

}

else break;

if(c)

if(p->resistance>c->resistance) {

int t=p->resistance;

p->resistance=c->resistance;

c->resistance=t;

}

temp+=step;

}

}

}

save(list, fileName);

break;

}

case 2: {

if(list->head) {

int step=0;

int lenght=0;

resistor \*p = list->head;

while(p) {

lenght++;

p=p->next;

}

while(2\*(3\*step+1) <= lenght) step=3\*step+1;

for(step; step>0 ;step/=3)

for(int i=step;i>0;i--)

for(int j=step-i; j<lenght; j+=step) {

p=list->head;

int k;

for(k=0;k<j;k++) p=p->next;

resistor\* c=p;

int temp=k+step;

while(c) {

for(k;k<temp;)

if(c) {

k++;

c=c->next;

}

else break;

if(c)

if(p->resistance < c->resistance) {

int t=p->resistance;

p->resistance=c->resistance;

c->resistance=t;

}

temp+=step;

}

}

}

save(list, fileName);

break;

}

}

}

qsort.c – модуль, в котором реализована быстрая сортировка по любому указанному пользователем полю.

#include "my.h"

void listnode\_swap(resistor \* a, resistor \* b) { //обмен узлами

float temp = a->power;

a->power = b->power;

b->power = temp;

double temp1 = a->admission;

a->admission = b->admission;

b->admission = temp1;

int temp2 = a->resistance;

a->resistance = b->resistance;

b->resistance = temp2;

char tmp[40];

strcpy(tmp, a->name);

strcpy(a->name,b->name);

strcpy(b->name,tmp);

}

void listnode\_qsortName(resistor \* left, resistor \* right) { //быстрая сортировка по имени

if ( left == right ) return;

else if ( left->next == right ) {

if ( strcmp(left->name, right->name) > 0)

listnode\_swap(left, right);

}

else {

resistor \* last = left;

resistor \* current = left;

do {

current = current->next;

if ( strcmp(current->name, left->name) < 0) {

last = last->next;

listnode\_swap(last, current);

}

} while ( current != right );

listnode\_swap(left, last);

listnode\_qsortName(left, last);

if ( last != right )

listnode\_qsortName(last->next, right);

}

}

void listnode\_qsortPower(resistor \* left, resistor \* right) {//быстрая сортировка по мощности

if ( left == right )

return;

else if ( left->next == right ) {

if ( left->power > right->power )

listnode\_swap(left, right);

}

else {

resistor \* last = left;

resistor \* current = left;

do {

current = current->next;

if ( current->power < left->power ) {

last = last->next;

listnode\_swap(last, current);

}

} while ( current != right );

listnode\_swap(left, last);

listnode\_qsortPower(left, last);

if ( last != right )

listnode\_qsortPower(last->next, right);

}

}

void listnode\_qsortResistance(resistor \* left, resistor \* right) { //быстрая сортировка по сопротивлению

if ( left == right )

return;

else if ( left->next == right ) {

if ( left->resistance > right->resistance )

listnode\_swap(left, right);

}

else {

resistor \* last = left;

resistor \* current = left;

do {

current = current->next;

if ( current->resistance < left->resistance ) {

last = last->next;

listnode\_swap(last, current);

}

} while ( current != right );

listnode\_swap(left, last);

listnode\_qsortResistance(left, last);

if ( last != right )

listnode\_qsortResistance(last->next, right);

}

}

void listnode\_qsortAdmission(resistor \* left, resistor \* right) {

if ( left == right )

return;

else if ( left->next == right ) {

if ( left->admission > right->admission )

listnode\_swap(left, right);

}

else {

resistor \* last = left;

resistor \* current = left;

do {

current = current->next;

if ( current->admission < left->admission ) {

last = last->next;

listnode\_swap(last, current);

}

} while ( current != right );

listnode\_swap(left, last);

listnode\_qsortAdmission(left, last);

if ( last != right )

listnode\_qsortAdmission(last->next, right);

}

}

void quicksort(resistor \* left, resistor \* right) { //функция для определения поля сортировки

int choice = 0;

while(1) {

printf("What do you want to sort?\n");

printf("1.Name\n2.Power\n3.Resistance\n4.Admission\n");

while(scanf("%d", &choice) != 1) {

printf("Must be non-negative integer!\n");

while(getchar() != '/n') continue;

}

if(choice < 1 || choice > 4){

printf("The number can't be less than 1 and greater than 2\n");

continue;

}

break;

}

switch(choice){

case 1: {

if ( left == right ) return;

else if ( left->next == right ) {

if (strcmp(left->name, right->name) > 0)

listnode\_swap(left, right);

} else {

resistor \* last = left;

resistor \* current = left;

do {

current = current->next;

if (strcmp(left->name, right->name) < 0) {

last = last->next;

listnode\_swap(last, current);

}

} while ( current != right );

listnode\_swap(left, last);

listnode\_qsortName(left, last);

if ( last != right )

listnode\_qsortName(last->next, right);

}

break;

}

case 2:{

if ( left == right ) return;

else if ( left->next == right ) {

if ( left->power > right->power )

listnode\_swap(left, right);

} else {

resistor \* last = left;

resistor \* current = left;

do {

current = current->next;

if ( current->power < left->power ) {

last = last->next;

listnode\_swap(last, current);

}

} while ( current != right );

listnode\_swap(left, last);

listnode\_qsortPower(left, last);

if ( last != right )

listnode\_qsortPower(last->next, right);

}break;

}

case 3: {

if ( left == right ) return;

else if ( left->next == right ) {

if ( left->resistance > right->resistance )

listnode\_swap(left, right);

} else {

resistor \* last = left;

resistor \* current = left;

do {

current = current->next;

if ( current->resistance < left->resistance ) {

last = last->next;

listnode\_swap(last, current);

}

} while ( current != right );

listnode\_swap(left, last);

listnode\_qsortResistance(left, last);

if ( last != right )

listnode\_qsortResistance(last->next, right);

}break;

}

case 4: {

if ( left == right ) return;

else if ( left->next == right ) {

if ( left->admission > right->admission )

listnode\_swap(left, right);

} else {

resistor \* last = left;

resistor \* current = left;

do {

current = current->next;

if ( current->admission < left->admission ) {

last = last->next;

listnode\_swap(last, current);

}

} while ( current != right );

listnode\_swap(left, last);

listnode\_qsortAdmission(left, last);

if ( last != right )

listnode\_qsortAdmission(last->next, right);

}break;

}

}

*Make*-Файл

*resistor: deleteNth.o selectionSort.o bubbleSort.o getNth.o pop.o shellSort.o compare.o insert.o printList.o someRandomFunction.o insertionSort.o pushBack.o swap.o main.o push.o update.o qsort.o create.o read.o write.o*

*gcc deleteNth.o selectionSort.o bubbleSort.o getNth.o pop.o shellSort.o compare.o insert.o printList.o someRandomFunction.o insertionSort.o pushBack.o swap.o main.o push.o update.o qsort.o create.o read.o write.o -o resistor*

*deleteNth.o: deleteNth.c my.h*

*gcc -c deleteNth.c*

*selectionSort.o: selectionSort.c my.h*

*gcc -c selectionSort.c*

*bubbleSort.o: bubbleSort.c my.h*

*gcc -c bubbleSort.c*

*getNth.o: getNth.c my.h*

*gcc -c getNth.c*

*pop.o: pop.c my.h*

*gcc -c pop.c*

*shellSort.o: shellSort.c my.h*

*gcc -c shellSort.c*

*compare.o: compare.c my.h*

*gcc -c compare.c*

*insert.o: insert.c my.h*

*gcc -c insert.c*

*printList.o: printList.c my.h*

*gcc -c printList.c*

*someRandomFunction.o: someRandomFunction.c my.h*

*gcc -c someRandomFunction.c*

*insertionSort.o: insertionSort.c my.h*

*gcc -c insertionSort.c*

*pushBack.o: pushBack.c my.h*

*gcc -c pushBack.c*

*swap.o: swap.c my.h*

*gcc -c swap.c*

*main.o: main.c my.h*

*gcc -c main.c*

*push.o: push.c my.h*

*gcc -c push.c*

*update.o: update.c my.h*

*gcc -c update.c*

*qsort.o: qsort.c my.h*

*gcc -c qsort.c*

*create.o: create.c my.h*

*gcc -c create.c*

*read.o: read.c my.h*

*gcc -c read.c*

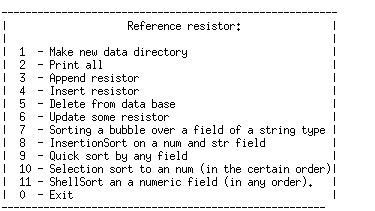
*write.o: write.c my.h*

*gcc -c write.c*

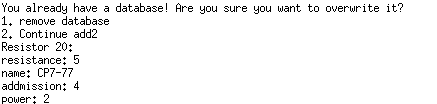
### 6. Результаты работы программы на примере конкретных данных

**Выбор вида главного меню программы:**

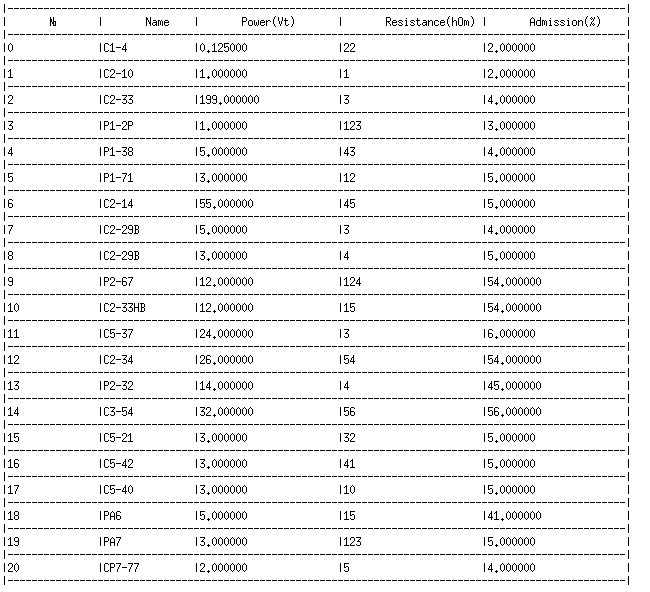
**Главное меню программы:**



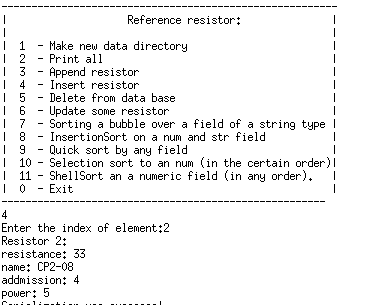
**Создание базы данных:**

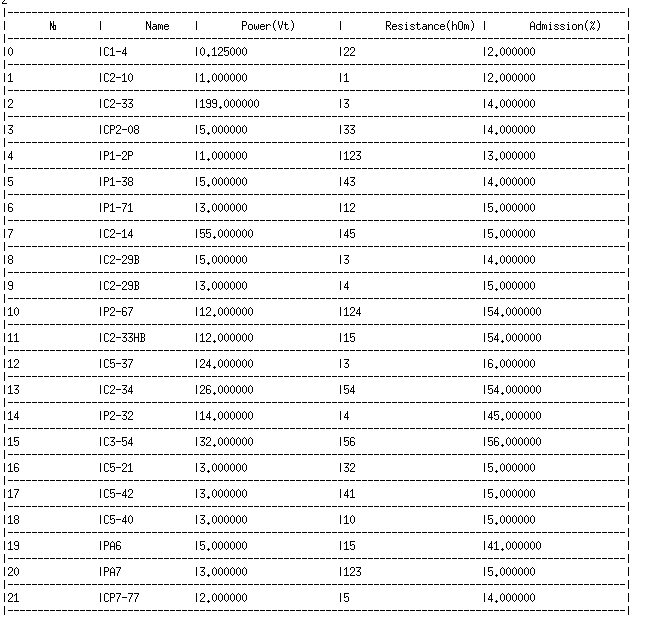


**Вывод базы данных на экран:**

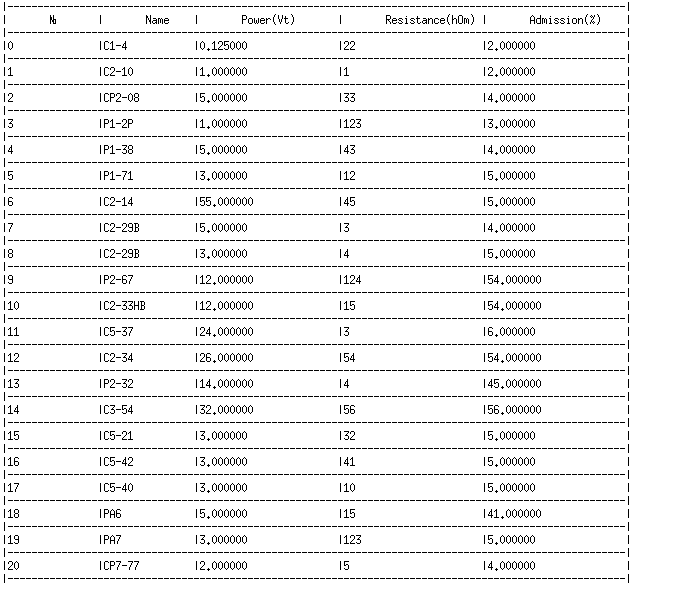


**Вставка в конец списка**:

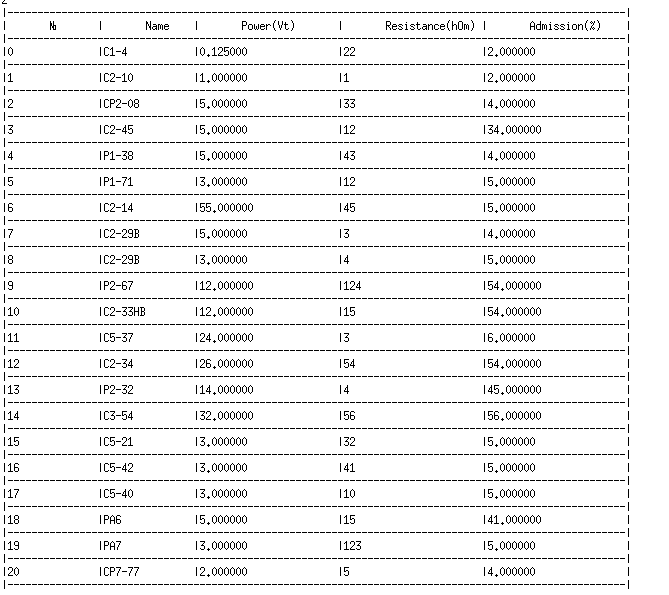
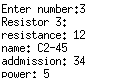


**Результат: **

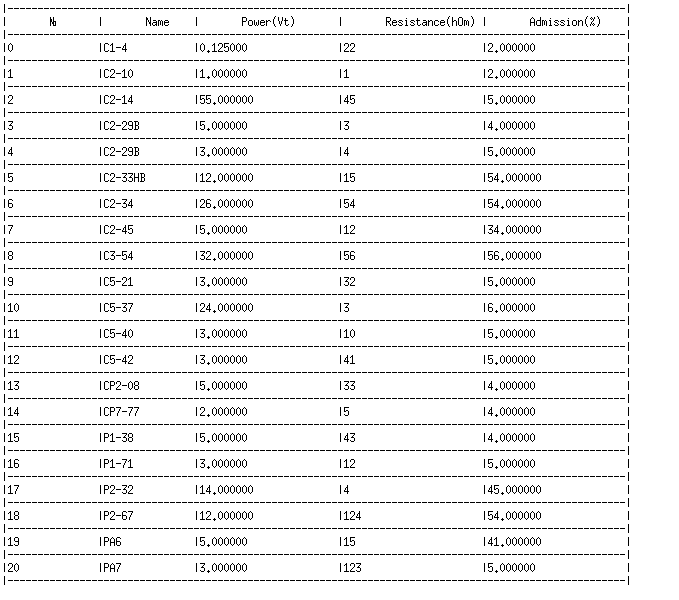
**Удаление резистора из списка**



**Корректировка данных в списке:**

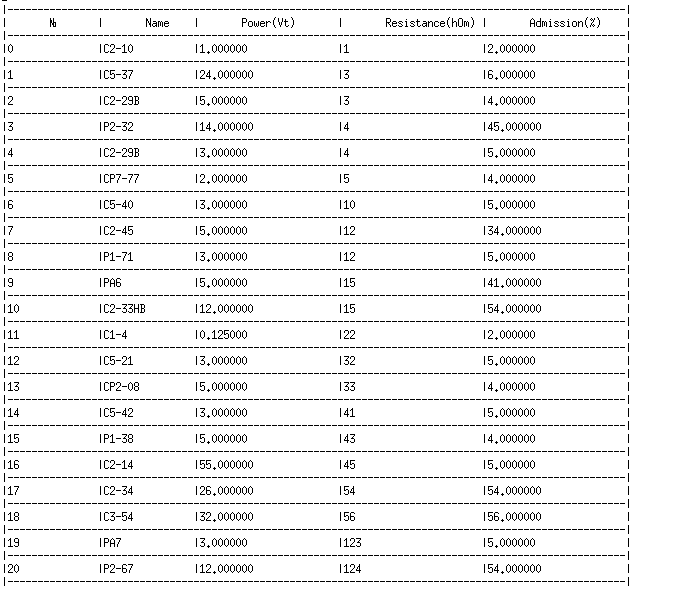


**Результат Пузырьковой сортировки:**



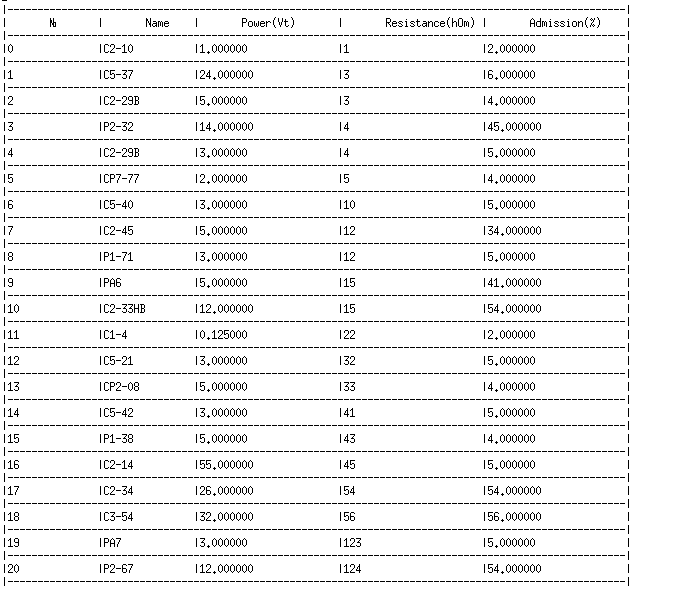
**Сортировка вставкой и ее результат:**



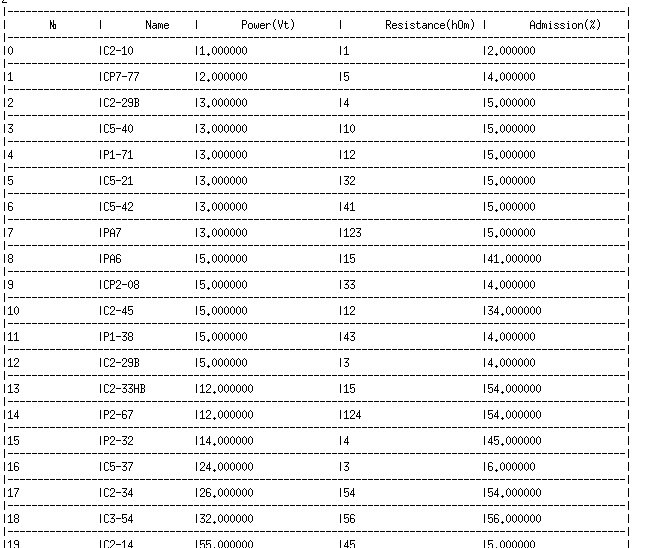


**Быстрая сортировка по любому полю:**

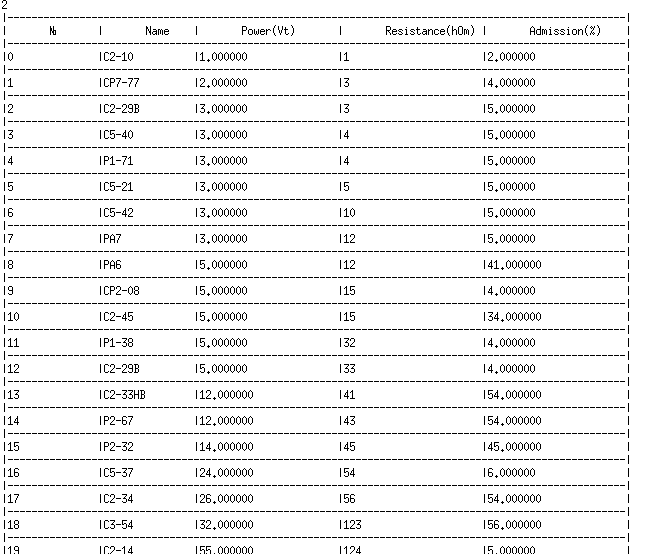




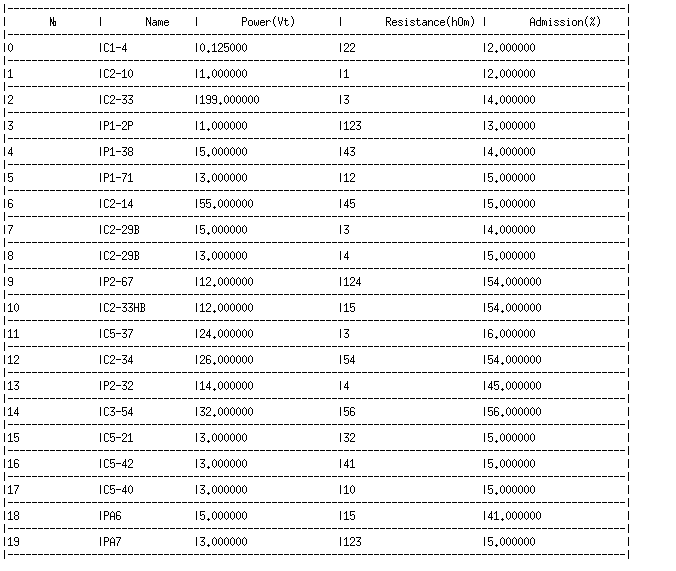
**Сортировка выбором:**



**Сортировка Шелла:**



**Приложение**



## ЗАКЛЮЧЕНИЕ

Была проделана работа по поиску реальной информации по заданной теме, а также составление пробного файла. Произведена отладка программы.

В ходе разработки программы приобрели навыки составления законченной программы с пользовательским интерфейсом и коррекцией ошибок. В ходе работы был проанализирован и систематизирован учебный материал. Получены навыки практического применения знаний по программированию на языке *C*

Так как язык *C* – довольно удобный инструмент для получения быстрых и компактных программ, использующих эффективно доступные ресурсы машины, он налаживает определённые требования на разработчика: в отличие от «классических» языков инженерного программирования (Фортран и т.д.) , язык *С* способен на базовом уровне выполнять только простейшие операции, характерные для большинства процессорных платформ (собственно с помощью чего достигается его эффективность). Вследствие этого инженер-программист, использующий для решения вычислительной проблемы язык *C*, должен обладать чёткой культурой программирования и ясно понимать процессы, происходящие в машине, при выполнении составленной им программы.

## СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Стивен Прата./ Язык программирования С. – М. Издательство “Диасофт”, 2002. – 896 с.

2. Р. Лафоре./ Объектно-ориентированное программирование в С++. – Спб. Издательство “Питер”, 2004. – 928 с.

3. П. Дейтел, Х. Дейтел./ Как программировать на С++. – М. Издательство “Бином” 2001. – 1152 с.

4. О. П. Гораев, Ю. П. Лыч. Текстовый процессор Microsoft Office Word. Практикум по компьютерным технологиям. БелГУТ, 2003. – 68 с.

5. Резисторы (справочник) / под ред. И. И. Четверткова — М.: Энергоиздат, 1991