**Московский авиационный институт**

(национальный исследовательский университет)

**Факультет № 8 «Информационные технологии и прикладная математика»**

**Кафедра 806 «Вычислительная математика и программирование»**

КУРСОВОЙ ПРОЕКТ

по курсу

«Практикум на ЭВМ»

на тему:

«Сортировка и поиск»

Студент: Пермяков Н.А.

Год приема: 2020

Группа: М8О - 108Б - 19

Руководитель: Поповкин А.В.

\_\_.\_\_.\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2020

#### Содержание:

Оглавление

[Цель работы 3](#_Toc36182801)

[Задание 4](#_Toc36182802)

[Общий метод работы 4](#_Toc36182803)

[Общие сведения о программе 5](#_Toc36182804)

[Описание программы 6](#_Toc36182805)

[Используемые структуры 6](#_Toc36182806)

[Описание функций программы 7](#_Toc36182807)

[Используемые переменные 7](#_Toc36182808)

[Входные и выходные данные 8](#_Toc36182809)

[Протокол 10](#_Toc36182810)

[Заключение 25](#_Toc36182811)

[Список использованных ресурсов 25](#_Toc36182812)

## Цель работы

Составить программу на языке Си с использованием процедур и функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице.

Программа должна вводить значения элементов неупорядоченной таблицы и проверять работу процедуры сортировки в трех случаях:

1) элементы таблицы с самого начала упорядочены;

2) элементы таблицы расставлены в обратном порядке;

3) элементы таблицы не упорядочены.

В последнем случае использовать встроенные процедуры генерации псевдослучайных чисел.

Для каждого вызова процедуры сортировки необходимо печатать исходное состояние таблицы и результаты сортировки.

После выполнения сортировки программа должна вводить ключи и для каждого из них выполнять поиск в упорядоченной таблице с помощью процедуры двоичного поиска и печатать найденные элементы, если они присутствуют в таблице.

В процессе отладки и тестирования использовать команды обработки текстовых файлов ОС UNIX и переадресацию ввода-вывода. Тестовые данные заранее поместить в текстовые файлы.

В качестве текста для записей таблицы взять изображение ASCII-графики. Каждый элемент таблицы, содержащий ключ и текст записи, распечатывать в отдельной строке.

## Задание

Вариант 18

Содержимое и структура таблицы

**Тип ключа:** комбинированный (целое + литера)

**Длинна ключа в байтах:** 5

**Хранение данных и ключей:** отдельно

**Число элементов таблицы:** 8-12

**Тип данных:** изображения ASCII-графики

Действия

Осуществлять сортировку записей таблицы методом Шелла.

Сортировка Шелла – усовершенствованный алгоритм «сортировки вставками», с заданным расстоянием между сравниваемыми элементами. Если значение расстояния не кратно длине сортируемого массива, то возможны циклические сдвиги сортируемого массива, а также изменения значений расстояния.

Существуют несколько подходов к выбору этих значений, в работе будет использоваться эмпирическая последовательность Марцина Циура (A102549 в OEIS): является одной из лучших для сортировки массива ёмкостью приблизительно до 4000 элементов.

## Общий метод работы

## Общие сведения о программе

**язык и система программирования:** GNU C

**местонахождение файлов:** /mnt/c/users/permi/source/high\_school/course\_works

**способ запуска:**

gcc line\_lists.c

./a.out

**Вывод меню:**

Line Lists

1. Input data to List

2. Print all List

3. Delete node Lists

4. Calcucate length List

5. Special action

6. Quit

Your choice ...

для вывода данных:

2

для записи данных:

1

Указание индекса вставляемого элемента:

Тип Integer – диапазон неограничен, в том числе отрицательные числа

(список циклический)

для очистки данных:

3

Указание индекса удаляемого элемента:

Тип Integer – диапазон неограничен, в том числе отрицательные числа

(список циклический)

для вывода размера:

4

для запуска специального действия:

5

Указание индекса центрального элемента:

Тип Integer – диапазон неограничен, в том числе отрицательные числа

(список циклический)

Для завершения программы:

6

При выполнении курсовой работы использовался ноутбук на базе процессора Intel Core i7 10th generation, под WSL Ubuntu

## 

## Описание программы

В файле line\_lists.h объявляем структуру данных. В файле line\_lists.c принимаем данные с консоли, записываем в двусвязный кольцевой список, выводим информацию и содержимое списка по запросу. В line\_lists.c читаем данные, считаем по алгоритму – выводим результат в консоли.

## Используемые структуры

|  |  |  |
| --- | --- | --- |
| Название | Содержимое | Описание |
| prev\_dump.c | | |
| \_node | int data;  struct \_node \*post;  struct \_node \*prev; | Сам элемент  Указатель на  предыдущий элемент  Указатель на следующий элемент |

## Описание функций программы

|  |  |  |
| --- | --- | --- |
| Название | Аргументы | Описание |
| line\_lists.c | | |
| void get\_choice |  | Вывод информации и меню действий |
| void input\_data |  | Выбор событий – инициализация списка, либо добавление элемента |
| void init\_list | struct \_node \*node, int \*new\_data | Инициализация списка и добавление первого элемента |
| void create\_new\_node | struct \_node \*node, int \*new\_data | Вставка нового элемента на указанное место |
| void shift\_insert | struct \_node \*current\_node, int \*index, int \*new\_data | Сдвиг по списку на указанное количество элементов |
| void print\_list | struct \_node \*node | Вывод списка |
| void delete\_node | struct \_node \*node | Удаление указанного элемента |
| int get\_length\_list | struct \_node \*node | Возвращает количество элементов |
| void roque\_two\_elements |  | Рокировка двух элементов |

## Используемые переменные

|  |  |
| --- | --- |
| Название | Описание |
| int index | Индекс для примениния действий к указанному элементу |
| int choice | Выбор действия |
| struct \_node \*new\_node | Новый элемент |
| int length\_list | Длина списка |
| struct \_node \*current\_node | Текущий перебираемый элемент |

**Используемые функции**

|  |  |
| --- | --- |
| Название | Описание |
| void\* malloc (size\_t size); | Выделяет блок памяти в байтах, возвращая указатель на начало блока. |
| void\* calloc (size\_t num, size\_t size); | Выделяет блок памяти для массива из n элементов, каждый из которых имеет размер в байтах, и инициализирует все его биты нулями. |
| void free (void\* ptr); | Блок памяти, ранее выделенный вызовом malloc, calloc или realloc, освобождается, что делает его снова доступным для дальнейшего распределения. |

## Входные и выходные данные

|  |  |  |
| --- | --- | --- |
|  | входные | выходные |
| из консоли | ./a.out  1  1  2  2  3  6  4  6  5  4  2  3  13245  3  1325  4  4  5  2  1523 | Line Lists  1. Input data to List  2. Print all List  3. Delete node Lists  4. Calcucate length List  5. Special action  6. Quit  Your choice ...  Your new node:  Input index for new node:  Your new node:  Input index for new node:  Your new node:  Input index for new node:  Your new node:  Input index for new node:  Your new node:  Input index for new node:  Now you have such List:  Input index for delete:  Your List has length - %d elements  Enter central index for roque adjacent values round it: |

## Протокол

**Файл line\_lists.h:**

#ifndef \_LINE\_LISTS\_H

#define \_LINE\_LISTS\_H

#include<stdlib.h>

#include<stdio.h>

typedef struct \_node{

int data;

struct \_node \*post;

struct \_node \*prev;

} \_node;

int get\_choice();

void input\_data();

void init\_list(struct \_node \*node, int \*new\_data);

void create\_new\_node(struct \_node \*node, int \*new\_data);

void shift\_insert(struct \_node \*current\_node, int \*index, int \*new\_data);

void print\_list(struct \_node \*node);

void delete\_node(struct \_node \*node);

int get\_length\_list(struct \_node \*node);

void roque\_two\_elements();

#endif

**Файл line\_lists.c:**

#include"line\_lists.h"

int get\_choice(){

int choice;

printf("Line Lists\n");

printf("1. Input data to List\n");

printf("2. Print all List\n");

printf("3. Delete node Lists\n");

printf("4. Calcucate length List\n");

printf("5. Special action\n");

printf("6. Quit\n");

printf("Your choice ...");

scanf("%d", &choice);

printf("\n");

return choice;

}

void init\_list(struct \_node \*node, int \*new\_data){

node->data = \*new\_data;

node->post = node;

node->prev = node;

}

void shift\_insert(struct \_node \*current\_node, int \*index, int \*new\_data){

struct \_node \*new\_node;

new\_node = (struct \_node\*)malloc(sizeof(struct \_node));

int length\_list = get\_length\_list(current\_node);

if(\*index >= 0) {

for(int i = 0; i < (\*index % length\_list); i++){

current\_node = current\_node->prev;

}

new\_node->data = \*new\_data;

new\_node->post = current\_node;

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

current\_node->prev->post = new\_node;

current\_node->prev = new\_node;

} else {

for(int i = 0; i < length\_list - (abs(\*index) % length\_list); i++){

current\_node = current\_node->post;

}

new\_node->data = \*new\_data;

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

new\_node->prev = current\_node;

current\_node->post->prev = new\_node;

current\_node->post = new\_node;

}

}

void create\_new\_node(struct \_node \*node, int \*new\_data){

int index;

printf("Input index for new node: ");

scanf("%d", &index);

struct \_node \*current\_node;

current\_node = (struct \_node\*)malloc(sizeof(struct \_node));

current\_node = node;

shift\_insert(current\_node, &index, new\_data);

}

void input\_data(struct \_node \*node){

int new\_data;

printf("Your new node: ");

scanf("%d", &new\_data);

if (get\_length\_list(node) == 0) {

init\_list(node, &new\_data);

} else {

create\_new\_node(node, &new\_data);

}

}

void print\_list(struct \_node \*node){

if(node == NULL){

printf("\nERROR: List was not initialized.\n");

exit(1);

}

printf("\nNow you have such List: ");

struct \_node \*current\_node;

current\_node = (struct \_node\*)malloc(sizeof(struct \_node));

current\_node = node;

for(int i = 0; i < get\_length\_list(node); i++) {

current\_node = current\_node->prev;

printf("%d ", current\_node->data);

}

printf("\n");

}

void delete\_node(struct \_node \*node){

if(node == NULL){

printf("\nERROR: List was not initialized.\n");

exit(1);

}

struct \_node \*current\_node;

current\_node = (struct \_node\*)malloc(sizeof(struct \_node));

current\_node = node;

int index, length\_list = get\_length\_list(node);

printf("Input index for delete: ");

scanf("%d", &index);

if(index >= 0) {

for(int i = 0; i < (index % length\_list); i++){

current\_node = current\_node->prev;

}

current\_node->prev = current\_node->prev->prev;

current\_node = current\_node->prev->post;

} else {

for(int i = 0; i < length\_list - (abs(index) % length\_list); i++){

current\_node = current\_node->post;

}

current\_node->post = current\_node->post->post;

current\_node = current\_node->post->prev;

}

}

int get\_length\_list(struct \_node \*start\_node){

struct \_node \*current\_node;

if(start\_node->data == 0){

return 0;

}

current\_node = start\_node;

int count\_node = 0;

do {

count\_node++;

current\_node = current\_node->prev;

} while (start\_node != current\_node);

return count\_node;

}

void roque\_two\_elements(struct \_node \*node) {

if(node == NULL){

printf("\nERROR: List was not initialized.\n");

exit(1);

}

struct \_node \*current\_node;

current\_node = (struct \_node\*)malloc(sizeof(struct \_node));

current\_node = node;

int index, temp, length\_list = get\_length\_list(node);

printf("Enter central index for roque adjacent values round it: ");

scanf("%d", &index);

if(index >= 0) {

for(int i = 0; i < (index % length\_list); i++){

current\_node = current\_node->prev;

}

} else {

for(int i = 0; i < length\_list - (abs(index) % length\_list); i++){

current\_node = current\_node->post;

}

}

temp = current\_node->post->data;

current\_node->post->data = current\_node->prev->data;

current\_node->prev->data = temp;

}

int main(){

struct \_node \*node;

node = (struct \_node\*)calloc(1, sizeof(struct \_node));

while(1) {

switch(get\_choice()){

case 1:

input\_data(node);

break;

case 2:

print\_list(node);

break;

case 3:

delete\_node(node);

print\_list(node);

break;

case 4:

printf("Your List has length - %d elements", get\_length\_list(node));

printf("\n");

break;

case 5:

roque\_two\_elements(node);

print\_list(node);

break;

case 6:

exit(1);

break;

default:

break;

}

printf("\n");

}

}

## Заключение

В процессе работы над курсовым проектом были изучены способы разделения данных на следующие структуры списков:

1. кольцевой однонаправленный;
2. линейный однонаправленный;
3. линейный однонаправленный с барьерным элементом;
4. кольцевой двунаправленный;
5. линейный двунаправленный;
6. линейный двунаправленный с барьерным элементом;

Реализованы функции:

1. Печать списка.
2. Вставка нового элемента в список.
3. Удаление элемента из списка.
4. Подсчет длины списка.

Составлена нестандартная функция:

обмена местами (к - 1)-й и (к + 1)-й элементов списка

Отработаны навыки передачи динамических структур данных в функции, использование глобальны переменных в структуре. Применялась процедурная парадигма программирования. В процессе написания программы использовалась арифметика указателей.

## Список использованных ресурсов

* Лорин Г. Сортировка и системы сортировки. -М.: Наука, 1983.
* Кнут Д. Искусство программирования для ЭВМ. Т 3. Сортировка и поиск. -М.Мир, 1976.
* \* Карасев С.Б., Кошелева Т.Я., Чернышов Л.Н. Машинные алгоритмы обработки информации. -М.: Изд-во МАИ, 1987.
* \* Разумов О.С. Организация данных в вычислительных системах. -М.: Статистика, 1978. - 184 с.
* Кормен Т., Лейзерсон Ч, Ривест Р. Алгоритмы: построение и анализ. -М.: МЦНМО, 2000. - 960 с., ил.
* \* Шенъ А. Программирование. Теоремы и задачи. -М.: МЦНМО, 1995.