**Отчет по лабораторной работе № 26** по курсу

Практикум программирование

Студент группы: М8О - 110Б - 23, Гула Дмитрий Александрович, № по списку: 6,

Контакты: guladmitriy1331@gmail.com

Работа выполнена: ««25» мая 2024г.

Преподаватель: каф. 806 Почечура А. А.

Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан «31» мая 2024 г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Тема:** Сортировка таблицы методом пирамидальной сортировки и двоичный поиск по ключу.
2. **Цель работы:** Разработать программу на языке Си с использованием процедур и функций для сортировки таблицы методом пирамидальной сортировки с просеиванием и выполнения двоичного поиска по ключу.
3. **Задание**

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

1. Элементы таблицы с самого начала упорядочены.
2. Элементы таблицы расставлены в обратном порядке.
3. Элементы таблицы не упорядочены.

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.

1. **Оборудование** (лабораторное):

ЭВМ , процессор . имя узла сети с ОП Мб, НМД Мб. Терминал

адрес . Принтер Другие устройства



*Оборудование ПЭВМ студента, если использовалось:*

Процессор \_Intel Core i3-12500H \_ с ОП 8 Гб НМД 256 Гб. Монитор 1920x1080~144Hz Другие устройства



1. **Программное обеспечение** (лабораторное):

Операционная система семейства , наименование версия



интерпретатор команд версия

Система программирования версия

Редактор текстов

версия

Утилиты операционной системы



Прикладные системы и программы



Местонахождение и имена файлов программ и

данных

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства \_Linux\_, наименование \_Ubuntu\_ версия 22.04.2 интерпретатор команд \_GNU bash\_ версия 5.1.16.

Система программирования С. Редактор текстов emacs версия 29.1

Утилиты операционной системы

Прикладные системы и программы Emacs

Местонахождение и имена файлов программ и данных на домашнем компьютере /home/

1. **Идея, метод, алгоритм** решения задачи (в формах: словесной, псевдокода, графической [блоксхема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)

**Идея**

Целью задания является разработка модуля определений и реализации для абстрактного типа данных (АТД) "очередь" с возможностью выполнения операции поиска первого элемента, который меньше своего непосредственного предшественника, и смещения его к началу очереди до тех пор, пока он не станет первым или больше своего предшественника. Также необходимо реализовать метод сортировки для очереди.

**Метод**

Программа состоит из трех основных файлов:

1. **Модуль определений (queue.h)**:
   * Определяет структуру данных "очередь" и прототипы функций, работающих с этой структурой.
2. **Модуль реализации (queue.c)**:
   * Реализует функции, описанные в модуле определений.
   * Реализует операции для поиска и смещения элемента в очереди и метод сортировки с использованием метода вставки.
3. **main.c**:

* Файл, предоставляющий интерфейс пользователя и демонстрирующий работу программы.

**Алгоритм**

1. **Алгоритм:**
   * Создать временную пустую очередь, которая будет содержать отсортированные элементы.
   * Извлекать элементы из исходной очереди по одному.
   * Вставлять каждый элемент в правильное место в отсортированной очереди так, чтобы сохранить упорядоченность по возрастанию.
   * После того как все элементы будут извлечены и вставлены, скопировать отсортированные элементы обратно в исходную очередь.
2. **Шаги алгоритма подробно:**
   * Пока исходная очередь не пуста:
     + Извлекаем элемент из начала исходной очереди.
     + Пока отсортированная очередь не пуста и элементы отсортированной очереди больше извлеченного элемента:
       - Извлекаем элемент из отсортированной очереди и вставляем его обратно в исходную очередь.
     + Вставляем извлеченный элемент в отсортированную очередь.
   * Копируем элементы из отсортированной очереди обратно в исходную очередь.
3. **Завершение алгоритма:**
   * Исходная очередь теперь содержит отсортированные элементы.
4. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

**1. Инициализация и создание очереди**

При запуске программы будет создана очередь, которая будет использоваться для хранения элементов.

Queue\* q = queue\_create();

**2. Добавление элементов в очередь**

Пользователь добавляет элементы в очередь. Для целей демонстрации, предположим, что пользователь добавляет элементы [4, 2, 6, 3, 1].

queue\_push(q, 4)

queue\_push(q, 2)

queue\_push(q, 6)

queue\_push(q, 3)

queue\_push(q, 1)

**3. Вывод исходного состояния очереди**

Программа выводит исходное состояние очереди перед сортировкой.

Исходное состояние очереди:

4 2 6 3 1

**4. Сортировка очереди методом вставки**

Происходит сортировка очереди методом вставки. Функция queue\_sort(q) изменяет порядок элементов в очереди.

insertion\_sort\_queue(q);

**5. Вывод отсортированной очереди**

После сортировки программа выводит отсортированное состояние очереди.

print\_queue(q);

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).

#ifndef C\_QUEUE\_H  
#define C\_QUEUE\_H  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <stdbool.h>  
  
typedef struct Queue Queue;  
typedef Queue\* Queue\_handle;  
  
  
struct Queue {  
 int first;  
 int size;  
 int capacity;  
 double\* buff;  
};  
  
void init\_queue(Queue\_handle q);  
void destroy\_queue(Queue\_handle q);  
int queue\_size(Queue\_handle q);  
bool queue\_is\_empty(Queue\_handle q);  
bool grow\_buffer(Queue\_handle q);  
bool queue\_push(Queue\_handle q, double val);  
bool shrink\_buffer(Queue\_handle q);  
bool queue\_pop(Queue\_handle q);  
void print\_queue(Queue\_handle q);  
void find\_and\_shift(Queue\_handle q);  
void insertion\_sort\_queue(Queue\_handle q);  
void print\_menu();  
  
#endif //C\_QUEUE\_H

#include "queue.h"  
  
void init\_queue(Queue\_handle q) {  
 q->first = 0;  
 q->capacity = 10;  
 q->size = 0;  
 q->buff = (double \*) malloc(q->capacity \* sizeof(double));  
 if (q->buff == NULL) {  
 fprintf(stderr, "Не удалось выделить память...\n");  
 exit(EXIT\_FAILURE);  
 }  
}  
  
  
void destroy\_queue(Queue\_handle q) {  
 free(q->buff);  
 q->buff = NULL;  
 q->capacity = 0;  
 q->first = 0;  
 q->size = 0;  
 free(q);  
}  
  
int queue\_size(Queue\_handle q) {  
 return q->size;  
}  
  
bool queue\_is\_empty(Queue\_handle q) {  
 return queue\_size(q) == 0;  
}  
  
  
bool grow\_buffer(Queue\_handle q) {  
 int new\_cap = q->capacity \* 3 / 2;  
 double \*new\_buff = realloc(q->buff, sizeof(double) \* new\_cap);  
 if (new\_buff == NULL) {  
 return false;  
 }  
 q->buff = new\_buff;  
 for (int i = 0; i < (q->capacity - q->first); i++) {  
 q->buff[new\_cap - 1 - i] = q->buff[q->capacity - 1 - i];  
 }  
 q->first = new\_cap - q->capacity;  
 q->capacity = new\_cap;  
 return true;  
}  
  
bool queue\_push(Queue\_handle q, double val) {  
 if (q->size == q->capacity) {  
 if (!grow\_buffer(q)) {  
 return false;  
 }  
 }  
 q->buff[(q->first + q->size) % q->capacity] = val;  
 q->size++;  
 return true;  
}  
  
bool shrink\_buffer(Queue\_handle q) {  
 if (q->size < q->capacity / 4) {  
 int new\_cap = q->capacity / 2;  
 double \*new\_buff = (double \*) malloc(sizeof(double) \* new\_cap);  
 if (new\_buff == NULL) {  
 fprintf(stderr, "Не удалось выделить память");  
 return false;  
 }  
  
 for (int i = 0; i < q->size; i++) {  
 new\_buff[i] = q->buff[(q->first + i) % q->capacity];  
 }  
  
 free(q->buff);  
 q->buff = new\_buff;  
 q->capacity = new\_cap;  
 q->first = 0;  
 }  
 return true;  
}  
  
bool queue\_pop(Queue\_handle q) {  
 if (queue\_is\_empty(q)) {  
 printf("Очередь и так пустая\n");  
 return false;  
 }  
 q->first = (q->first + 1) % q->capacity;  
 q->size--;  
 shrink\_buffer(q);  
 return true;  
}  
  
void print\_queue(Queue\_handle q) {  
 if (queue\_is\_empty(q)) {  
 printf("Очередь пуста\n");  
 return;  
 }  
  
 printf("Очередь: ");  
 int index = q->first;  
 for (int i = 0; i < q->size; ++i) {  
 printf("%.2f ", q->buff[index]);  
 index = (index + 1) % q->capacity;  
 }  
 printf("\n");  
}  
  
void find\_and\_shift(Queue\_handle q) {  
 if (q->size < 2) {  
 return;  
 }  
 Queue\_handle temp\_queue = (Queue\_handle) malloc(sizeof(Queue));  
 init\_queue(temp\_queue);  
 double prev = q->buff[q->first];  
 queue\_pop(q);  
 queue\_push(temp\_queue, prev);  
 bool found = false;  
 double element\_to\_shift = 0;  
 while (!queue\_is\_empty(q)) {  
 double current = q->buff[q->first];  
 queue\_pop(q);  
 if (!found && current < prev) {  
 element\_to\_shift = current;  
 found = true;  
 } else {  
 queue\_push(temp\_queue, current);  
 }  
 prev = current;  
 }  
 if (!found) {  
 while (!queue\_is\_empty(temp\_queue)) {  
 double val = temp\_queue->buff[temp\_queue->first];  
 queue\_pop(temp\_queue);  
 queue\_push(q, val);  
 }  
 destroy\_queue(temp\_queue);  
 return;  
 }  
 init\_queue(q);  
 bool inserted = false;  
 while (!queue\_is\_empty(temp\_queue)) {  
 double val = temp\_queue->buff[temp\_queue->first];  
 queue\_pop(temp\_queue);  
 if (!inserted && element\_to\_shift < val) {  
 queue\_push(q, element\_to\_shift);  
 inserted = true;  
 }  
 queue\_push(q, val);  
 }  
 if (!inserted) {  
 queue\_push(q, element\_to\_shift);  
 }  
 destroy\_queue(temp\_queue);  
}  
  
void insertion\_sort\_queue(Queue\_handle q) {  
 if (q->size < 2) {  
 return;  
 }  
 for (int i = 1; i < q->size; i++) {  
 find\_and\_shift(q);  
 }  
}  
  
void print\_menu() {  
 printf("h - помощь\n");  
 printf("0 - завершить программу\n");  
 printf("1 <value> - добавить элемент в очередь\n");  
 printf("2 - удалить элемент из очереди\n");  
 printf("3 - вывести очередь\n");  
 printf("4 - отсортировать очередь при помощи метода вставки\n");  
}

#include "stdio.h"  
#include "queue.h"  
  
int main() {  
 printf("Hello world\n");  
 Queue\_handle q = (Queue\_handle) malloc(sizeof(Queue));  
 if (q == NULL) {  
 fprintf(stderr, "Ошибка: не удалось выделить память для очереди\n");  
 return EXIT\_FAILURE;  
 }  
 init\_queue(q);  
 char command;  
 print\_menu();  
 while (scanf(" %c", &command) != EOF) {  
 switch (command) {  
 case 'h': {  
 print\_menu();  
 break;  
 }  
 case '0': {  
 printf("Завершаем программу...");  
 destroy\_queue(q);  
 exit(0);  
 }  
 case '1': {  
 double value;  
 scanf("%lf", &value);  
 queue\_push(q, value);  
 break;  
 }  
 case '2': {  
 queue\_pop(q);  
 break;  
 }  
 case '3': {  
 print\_queue(q);  
 break;  
 }  
 case '4': {  
 insertion\_sort\_queue(q);  
 break;  
 }  
 default:  
 printf("Ошибка: Неверный выбор меню. Попробуйте еще раз.\n");  
 }  
 }  
}

C:\Users\Дима\CLionProjects\C\LB26\cmake-build-debug\LB26.exe

Hello world

h - помощь

0 - завершить программу

1 <value> - добавить элемент в очередь

2 - удалить элемент из очереди

3 - вывести очередь

4 - отсортировать очередь при помощи метода вставки

1 5

1 4

1 3

1 2

1 1

3

Очередь: 5.00 4.00 3.00 2.00 1.00

2

2

3

Очередь: 3.00 2.00 1.00

4

3

Очередь: 1.00 2.00 3.00

2

2

3

Очередь: 3.00

1 2

1 1

3

Очередь: 3.00 2.00 1.00

4

3

Очередь: 1.00 2.00 3.00

0

Завершаем программу...

Process finished with exit code 0

**Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб.  или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы



1. **Выводы**

В данной работе была разработана и отлажена программа на языке Си для обработки очереди. Программа успешно выполняет заданные действия: печать очереди, вставка нового элемента, сортировка очереди

Подпись студента