**Отчёт по лабораторной работе** №\_\_\_\_\_\_\_ по курсу\_\_\_\_\_\_\_

студент группы М8О-107Б-18 Гамов Павел Антонович № по списку4

Адрес (e-mail) pagamov@gmail.com

Работа выполнена: “ “ сентября 2018г.

Преподаватель: асп. Каф. 806 Ридли А.Н.

Входной контроль знаний с оценкой

Отчёт сдан “ “ 2018 г итоговая оценка

Подпись преподавателя

1. Тема:

Язык Си. Стеки, деки, очереди. Реализация их на языке Си.

Знакомство с утилитой make. Линковка файлов.

1. Цель работы:

Познакомиться с утилитой make, создать свой файл makefile для своего проекта.

Реализовать одну из заданных в варианте структур. Реализовать нужные для каждого из видов структур функции. Также реализовать одну из предложенных функций для данной структуры.

1. Задание:

На языке Си создать очередь на массивах. Реализовать функции добавления, удаления, pop, push. Написать быструю сортировку Хоара для этой очереди.

1. **Оборудование** (лабораторное):

ЭВМ Pentium , процессор2.7 ГГц , имя узла сети с ОП 4096 МБ

НМД 2 ГБ. Терминал bash адрес . Принтер

Другие устройства

Оборудование ПЭВМ студента, если использовалось:

Процессор 2.7 GHz Intel Core i5, ОП 8 GB (1867 MHz LPDDR3), НМД 256 ГБ. Монитор встроенный

Другие устройства

1. Программное обеспечение (лабораторное):

Операционная система семейства Unix , наименование Ubuntu версия 18.04.1 LTS

Интерпретатор команд bash версия 4.3.48

Система программирования версия

Редактор текстов Emacs версия 7.4

Утилиты операционной системы

Прикладные системы и программы

Местонахождения и имена файлов программ и данных

1. Программное обеспечение ЭВМ студента, если использовалось:

Операционная система семейства Unix , наименование Ubuntu версия 18.04.1 LTS

Интерпретатор команд bash версия 4.3.48

Система программирования версия

Редактор текстов GNU Emacs версия 25.2.2

Утилиты операционной системы

Прикладные системы и программы

Местонахождения и имена файлов программ и данных /home/pavel

1. **Идея, метод, алгоритм** решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальное описание с пред- и постусловиями)
2. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты, либо соображения по тестированию].

Создаем файл makefile нужный для утилиты make, в котором будем линковать все наши файлы.

В struct.h пропишем саму структуру. Она состоит из двух целочисленных массивов, указателя на голову или начало очереди, а также текущий размер выделенной памяти на массивы очереди. (нужно для динамического выделения памяти при переполнении).

queue.c и queue.h содержат основные функции нашей очереди.

sort.c и sort.h содержат в себе функции для сортировки подаваемой очереди.

main.c содержит меню выбора действий с вечным циклов while.

Пункты 1-7 отчёта составляются **строго** **до** начала лабораторной работы.

Допущен к выполнению работы. Подпись преподавателя

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с текстовыми примерами, подписанный преподавателем)
2. **Дневник отладки** должен содержать дату и время сеансов отладки, и основные ошибки (ошибки в сценарии и программе, не стандартные операции) 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 | дом | .2018 |  |  |  |  |

1. Замечание автора по существу работы
2. Выводы

Реализация стеков, деков, очередей позволяет иметь конкретный набор функций, которые олицетворяют конкретный тип структуры. В зависимости от различных ситуаций, когда требуется использования подобных структур, быстрая линковка уже написанных функций в проект позволяет без проблем и затрат времени использовать их для своих задач.

Касательно утилиты make и написания файлов makefile. При довольно внушительных размерах проекта, такой подход к перекомпиляции файлов действительно помогает сэкономить некоторое количество времени.

1. **Недочеты**, допущенные при выполнении задания, могут быть устранены следующим образом

Подпись студента

**makefile**

CC = gcc

LD = gcc

CCFLAGS = -Wall -pedantic -std=c99

###\_\_\_\_###

a.out: main.o queue.o sort.o ; $(LD) -o a.out main.o queue.o sort.o

main.o: main.c queue.h sort.h struct.h ; $(CC) $(CCFLAGS) -c main.c

queue.o: queue.c queue.h struct.h ; $(CC) $(CCFLAGS) -c queue.c

sort.o: sort.c sort.h struct.h ; $(CC) $(CCFLAGS) -c sort.c

clean: ; rm \*.o

###\_\_\_###

**main.c**

#include <stdio.h>

#include <stdlib.h>

#include "queue.h"

#include "sort.h"

#include "struct.h"

int pass = 1;

char q;

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

Queue \* Q = createQueue(1);

printf("\na to add\nd to delete\n0 to delete all\n1 to get pop\n2 to push\n3 for sort\ns to see\nq for exit\n\n");

while (pass) {

scanf("%c", &q);

switch (q) {

case ('a'):

addQueue(Q);

break;

case ('d'):

deleteElemQueue(Q);

break;

case ('0'):

deleteQueue(Q);

break;

case ('1'):

printf("%d\n", popQueue(Q));

break;

case ('2'):

pushQueue(Q);

break;

case ('3'):

qSortQueue(Q);

break;

case ('s'):

printQueue(Q);

printf(" count: %d\n", getCountQueue(Q));

break;

case ('q'):

pass = 0;

break;

}

}

return 0;

}

**queue.c**

#include <stdio.h>

#include <stdlib.h>

#include "queue.h"

int findEmptyQueue(Queue \* Q) {

for (int i = 0; i < Q->size; i++) {

if (Q->index[i] == -2) {

return i;

}

}

return -1;

}

void resizeQueue(Queue \* Q, int delta) {

Q->data = (int \*)realloc(Q->data, delta \* sizeof(int));

Q->index = (int \*)realloc(Q->index, delta \* sizeof(int));

for (size\_t i = Q->size; i < delta; i++) {

Q->index[i] = -2;

}

Q->size = delta;

}

int getCountQueue(Queue \* Q) {

if (Q->head == -1) {

return 0;

}

int pivet = Q->head;

int count = 0;

while (1) {

if (Q->index[pivet] == -2) {

break;

} else if (Q->index[pivet] == -1) {

count++;

break;

} else {

pivet = Q->index[pivet];

count++;

}

}

return count;

}

void printQueue(Queue \* Q) {

int pivet = Q->head;

if (getCountQueue(Q) != 0) {

do {

printf("%d ", Q->data[pivet]);

pivet = Q->index[pivet];

} while(pivet != -1);

} else {

printf("empty\n");

}

}

Queue \* createQueue(int size) {

Queue \* Q = (Queue \*)malloc(sizeof(Queue));

Q->data = (int \*)malloc(size \* sizeof(int));

Q->index = (int \*)malloc(size \* sizeof(int));

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

Q->index[i] = -2;

}

Q->head = -1;

Q->size = size;

return Q;

}

void addQueue(Queue \* Q) {

int a;

int pivet = Q->head;

int new;

printf("how many new? ");

scanf("%d", &a);

printf("write: ");

if (Q->size < a + getCountQueue(Q)) {

resizeQueue(Q, a + getCountQueue(Q));

}

if (Q->head == -1) {

Q->head = findEmptyQueue(Q);

}

while (a > 0) {

if (Q->index[pivet] == -1) {

new = findEmptyQueue(Q);

scanf("%d", &Q->data[new]);

Q->index[new] = -1;

Q->index[pivet] = new;

a--;

} else if (Q->index[pivet] == -2) {

Q->index[pivet] = -1;

scanf("%d", &Q->data[pivet]);

a--;

} else {

pivet = Q->index[pivet];

}

}

}

void deleteElemQueue(Queue \* Q) {

if (Q->head != -1) {

int target;

int pivet = Q->head;

int prev;

printf("write: ");

scanf("%d", &target);

while (pivet != -1) {

if (Q->data[pivet] == target) {

if (pivet == Q->head) {

Q->head = Q->index[pivet];

Q->index[pivet] = -2;

pivet = Q->head;

} else {

Q->index[prev] = Q->index[pivet];

Q->index[pivet] = -2;

pivet = Q->index[prev];

}

} else {

prev = pivet;

pivet = Q->index[pivet];

}

}

printf("done\n");

}

}

int popQueue(Queue \* Q) {

if (getCountQueue(Q) != 0) {

int res;

int pivet = Q->head;

Q->head = Q->index[pivet];

res = Q->data[pivet];

Q->index[pivet] = -2;

return res;

} else {

printf("empty\n");

return -1;

}

}

void pushQueue(Queue \* Q) {

if (Q->size > 0) {

int number;

int pivet = Q->head;

int empty;

printf("write: ");

scanf("%d", &number);

if (Q->size == getCountQueue(Q))

resizeQueue(Q, Q->size + 1);

empty = findEmptyQueue(Q);

if (Q->head == -1) {

Q->head = empty;

} else {

while (Q->index[pivet] != -1) {

pivet = Q->index[pivet];

}

Q->index[pivet] = empty;

}

Q->index[empty] = -1;

Q->data[empty] = number;

} else {

printf("no queue\n");

}

}

void deleteQueue(Queue \* Q) {

for (int i = 0; i < Q->size; i++) {

Q->index[i] = -2;

}

Q->head = -1;

}

**queue.h**

#ifndef QUEUE\_H

#define QUEUE\_H

#include <stdio.h>

#include <stdlib.h>

#include "struct.h"

Queue \* createQueue(int size);

void printQueue(Queue \* Q);

void addQueue(Queue \* Q);

void deleteElemQueue(Queue \* Q);

int popQueue(Queue \* Q);

void pushQueue(Queue \* Q);

void deleteQueue(Queue \* Q);

#endif

**sort.c**

#include <stdio.h>

#include <stdlib.h>

#include "sort.h"

#include "queue.h"

int getCountQueueQ(Queue \* Q) {

if (Q->head == -1) {

return 0;

}

int pivet = Q->head;

int count = 0;

while (1) {

if (Q->index[pivet] == -2) {

break;

} else if (Q->index[pivet] == -1) {

count++;

break;

} else {

pivet = Q->index[pivet];

count++;

}

}

return count;

}

void swapQ(int \* arr, int i, int j) {

arr[i] = arr[i] + arr[j];

arr[j] = arr[i] - arr[j];

arr[i] = arr[i] - arr[j];

}

void qsortQ(int \* arr, int b, int e);

void qsortQ(int \* arr, int b, int e) {

int l = b;

int r = e;

int piv;

int pivet = arr[(l + r) / 2];

while (l <= r) {

while (arr[l] < pivet)

l++;

while (arr[r] > pivet)

r--;

if (l <= r) {

piv = arr[l];

arr[l++] = arr[r];

arr[r--] = piv;

}

}

if (b < r)

qsortQ(arr, b, r);

if (e > l)

qsortQ(arr, l, e);

}

void qSortQueue(Queue \* Q) {

if (getCountQueueQ(Q) != 0) {

int \* pivetArr = (int \*)malloc(getCountQueueQ(Q) \* sizeof(int));

int pivet = Q->head;

int i = 0;

while (pivet != -1) {

pivetArr[i++] = Q->data[pivet];

pivet = Q->index[pivet];

}

qsortQ(pivetArr, 0, getCountQueueQ(Q) - 1);

i = 0;

pivet = Q->head;

while (pivet != -1) {

Q->data[pivet] = pivetArr[i++];

pivet = Q->index[pivet];

}

free(pivetArr);

}

}

**sort.h**

#ifndef SORT\_H

#define SORT\_H

#include <stdio.h>

#include <stdlib.h>

#include "struct.h"

void qSortQueue(Queue \* Q);

void qsortQ(int \* arr, int b, int e);

int getCountQueue(Queue \* Q);

#endif

**struct.h**

#ifndef STRUCT\_H

#define STRUCT\_H

typedef struct {

int \* data;

int \* index;

int head;

int size;

} Queue;

#endif