**Отчет по лабораторной работе №VIII** по курсу практикум на ЭВМ

Студент группы М8О-110Б-21, № по списку 2 Агеева Алиса

Контакты www, e-mail, icq, skype alisa.ageewa2003@gmail.com

Работа выполнена: « » \_\_\_\_\_\_\_\_\_\_\_\_202\_\_\_г.

Преподаватель: доцент каф. 806 Никулин Сергей Петрович

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

Отчет сдан « » \_\_\_\_\_\_\_\_\_202 \_\_ г., итоговая оценка

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

1. **Тема:** Линейные списки
2. **Цель работы:** Составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением на динамические структуры
3. **Задание** (*вариант №* 28 )**:**  Тип элемента: перечислимый. Вид списка: линейный однонаправленный. Нестандартное действие: Удаление предшествующего и следующего элемента заданному
4. **Оборудование** (лабораторное):

ЭВМ Intel Pentium G2140, процессор 3.30 GHz , имя узла сети Cameron с ОП 8096 Мб, НМД 7906 Мб. Терминал ASUS адрес dev/pets/3 Принтер HP Laserjet 6P

Другие устройства

*Оборудование ПЭВМ студента, если использовалось:*

Процессор \_\_Intel core i5-7300HQ 2.50 GHz с ОП 8096 Мб, НМД 131072 Мб. Монитор ASUS

Другие устройства

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Операционная система семейства Windows , наименование Ubuntu версия 18.15.0

интерпретатор команд bash версия 4.4.20

Система программирования GNU версия 5.8.13

Редактор текстов emacs версия 25.2.2

Утилиты операционной системы cat

Прикладные системы и программы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства Windows , наименование Fedora версия 33

интерпретатор команд bash версия 5.0.17

Система программирования Clion версия 2020.3

Редактор текстов emacs версия 25.2.2 Утилиты операционной системы cat, gcc

Прикладные системы и программы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Местонахождение и имена файлов программ и данных на домашнем компьютере

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Заведем следующие структуры:

struct List – структура листа

{

struct Node\* head;

int size;

struct Node\* end;

}

enum colors {

WHITE = 1,

BLUE,

RED,

YELLOW,

GREEN,

}; - перечеслимый тип

Структура списка. Содержит указатель, а также размер списка.

Опишем следующие функции:

* bool Equal(Iterator \*lhs, Iterator \*rhs)

Функция сравнения двух итераторов. Итераторы равны тогда и только тогда, когда указывают на один и тот же элемент

* Iterator \*Next(Iterator \*i)

Функция перехода к следующему элементу. Возвращает итератор на следующий элемент.

* void Store(const Iterator \*i, const float t)

Функция присваивает значение *t* элементу, на который указывает итератор.

* void Create(List \*l)

Функция создания списка. Выделяет память для барьерного элемента, указателю на следующий элемент присваивает значение барьерного элемента. Присваивает размеру списка значение 0

* Iterator First(const List \*l)

Функция возвращает итератор на первый элемент списка.

* Iterator Last(const List \*l)

Функция проверки на пустоту списка

* Iterator Insert(List \*l, Iterator \*i, const float t)

Функция вставки элемента в список. Возвращает итератор на только что вставленный элемент. Итератор i указывает на элемент, предшествующий вставляемому, то есть тот, после которого необходимо вставить.

* Iterator Delete(List \*l, Iterator \*i)

Функция удаления. Возвращает итератор на следующий элемент после удаляемого. Итератор i указывает на элемент, предшествующий удаляемому.

* int Size(const List \*l)

Функция возвращает размер списка.

* void print\_list(const List \*l)

Функция печати списка. С помощью итераторов пробегает по всем элементам списка и выводит их значение.

* void delPrevandNext(List \*l) – aeyrwbz pflfybz
* Iterator search\_prev(const List \*l, const float t)

Функция поиска предыдущего элемента. С помощью итераторов пробегает по списку и выполняет поиск элемента, удовлетворяющего условию:

Значение следующего элемента равно заданному.

* void Destroy(List \*l)

Функция уничтожения списка.

В основной части программы будем использовать меню (предварительно создав пустой список), в котором есть 6 опций:

1. Печать списка (Print list)

Вызывает функцию print\_list

1. Вставка в список (Insert in list)

Запрашивает значение вставляемого элемента, а также значение элемента, перед которым нужно вставить. Затем выполняет поиск элемента, предшествующего тому, перед которым необходимо вставить. После этого вызывает функцию Insert

1. Удаление из списка (Delete from list)

Запрашивает значение удаляемого элемента, выполняет поиск предшествующего, а затем вызывает функцию Delete

1. Размер списка (Size)

Вызывает функцию Size

1. Задание (Task)

Вызывает функцию task

1. Выход

Выходит из меню, завершая программу.

1. **Сценарий выполнения работы** [план работы,первоначальный текст программы в черновике(можно на отдельном листе)итесты либо соображения по тестированию].

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

*Допущен к выполнению работы.* **Подпись преподавателя****\_\_\_\_\_**

**8 Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами,подписанныйпреподавателем).

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct Node

{

int data;

struct Node\* next;

};

enum colors {

WHITE = 1,

BLUE,

RED,

YELLOW,

GREEN,

};

struct List

{

struct Node\* head;

int size;

struct Node\* end;

};

struct List createList(){

struct List list;

list.head = NULL;

list.size = 0;

return list;

}

void enum\_out(int x) {

switch (x) {

case 1:

printf("WHITE");

break;

case 2:

printf("BLUE");

break;

case 3:

printf("RED");

break;

case 4:

printf("YELLOW");

break;

case 5:

printf("GREEN");

break;

}

printf(" ");

}

typedef struct Node\* ListIterator;

void set\_element(struct Node\* \_node, int element){

if (\_node != NULL){

\_node->data = element;

}

}

void set\_next\_element(struct Node\* \_node, struct Node\* \_next){

if (\_node != NULL){

\_node->next = \_next;

}

}

bool is\_list\_empty(struct List\* \_list){

return \_list->size == 0;

}

ListIterator end(struct List\* \_list){

return \_list->end;

}

ListIterator begin(struct List\* \_list){

if (\_list != NULL && \_list->head != NULL){

return \_list->head;

}

else

return end(\_list);

}

ListIterator next(ListIterator It){

return It->next;

}

ListIterator plus(struct List\* \_list, ListIterator It, int \_n){

if (is\_list\_empty(\_list)){

return begin(\_list);

}

ListIterator ansIterator = It;

int i = 0;

while (i < \_n && next(ansIterator) != end(\_list)){

ansIterator = next(ansIterator);

++i;

}

return ansIterator;

}

int distance(ListIterator \_begin, ListIterator \_end){

int ans = 0;

ListIterator curIt = \_begin;

while (curIt != \_end){

++ans;

curIt = next(curIt);

}

return ans;

}

void increase(struct List\* \_list, ListIterator \_lIt, int \_n){

int i = 0;

while (i < \_n && \_lIt != end(\_list)){

\_lIt = next(\_lIt);

}

}

int getListLength(struct List\* \_list){

return distance(begin(\_list), end(\_list));

}

void pushBack(struct List\* \_list, int \_newData){

struct Node\* newNode = (struct Node\*) malloc(sizeof(struct Node));

set\_element(newNode, \_newData);

if (is\_list\_empty(\_list)){

set\_next\_element(newNode, end(\_list));

\_list->head = newNode;

++\_list->size;

return;

}

struct Node\* lastNode = begin(\_list);

while (next(lastNode) != end(\_list)){

lastNode = next(lastNode);

}

set\_next\_element(lastNode, newNode);

set\_next\_element(newNode, end(\_list));

++\_list->size;

}

void popBack(struct List\* \_list){

ListIterator prevNode = plus(\_list, begin(\_list), \_list->size - 2);

if (getListLength(\_list) > 1){

ListIterator deletedNode = next(prevNode);

free(deletedNode);

set\_next\_element(prevNode, end(\_list));

--\_list->size;

}

else if (getListLength(\_list) == 1){

free(\_list->head);

\_list->head = end(\_list);

--\_list->size;

}

}

void insert(struct List\* \_list, int \_newData, int \_pos){

if (\_pos > 0){

if (is\_list\_empty(\_list) || \_pos > getListLength(\_list)){

pushBack(\_list, \_newData);

return;

}

struct Node\* newNode = (struct Node\*) malloc(sizeof(struct Node));

set\_element(newNode, \_newData);

if (\_pos == 1){

set\_next\_element(newNode, \_list->head);

\_list->head = newNode;

++\_list->size;

return;

}

struct Node\* prevNode = plus(\_list, begin(\_list), \_pos - 2);

set\_next\_element(newNode, next(prevNode));

set\_next\_element(prevNode, newNode);

++\_list->size;

}

}

void printList(struct List\* \_list){

printf("list: ");

struct Node\* curNode = begin(\_list);

while (curNode != end(\_list)){

//printf("%d ", curNode->data);

enum\_out(curNode->data);

curNode = next(curNode);

}

printf("\n");

}

ListIterator findPrevNode(struct List\* \_list, int \_data){

ListIterator ans = begin(\_list);

if (ans->data == \_data){

return ans;

}

while (ans != end(\_list)){

if (next(ans)->data == \_data){

return ans;

}

else {

ans = next(ans);

}

}

return ans;

}

bool deleteByValue(struct List\* \_list, int \_data){

ListIterator prevNode = findPrevNode(\_list, \_data);

if (prevNode == begin(\_list)){

if (prevNode == end(\_list)){

return false;

}

if (prevNode->data == \_data){

struct Node\* deletedNode = prevNode;

\_list->head = next(deletedNode);

free(deletedNode);

--\_list->size;

return true;

}

else {

struct Node\* deletedNode = next(prevNode);

set\_next\_element(prevNode, next(deletedNode));

free(deletedNode);

--\_list->size;

return true;

}

}

else if (prevNode != end(\_list)){

struct Node\* deletedNode = next(prevNode);

set\_next\_element(prevNode, next(deletedNode));

free(deletedNode);

--\_list->size;

return true;

}

return false;

}

void deletePrevAndNext(struct List\* \_list, int \_data){

while (deleteByValue(\_list, \_data - 1));

while (deleteByValue(\_list, \_data + 1));

}

void destroyList(struct List\* \_list){

struct Node\* deletedNode = begin(\_list);

if (deletedNode == end(\_list)){

return;

}

struct Node\* nextNode = next(deletedNode);

while (nextNode != end(\_list)){

printf("%d %d\n", deletedNode->data, nextNode->data);

free(deletedNode);

deletedNode = nextNode;

nextNode = next(nextNode);

}

free(deletedNode);

}

int deleteByPosition(struct List\* \_list, int position){

ListIterator Data = begin(\_list);

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

Data = next(Data);

}

return Data->data;

}

void menu(){

printf("+ add el., '-' pop el. from the end of list, t - task, d - delete from list, i - put el. in entered position, p - print, l - lenght\n");

}

int main(){

menu();

printf("\n");

printf("Add and delete: 1 - for WHITE, 2 - for BLUE, 3 - for RED, 4 - for YELLOW, 5 - for GREEN\n");

char \*strings\_colors[] = {"WHITE", "BLUE", "RED", "YELLOW", "GREEN"};

struct List list = createList();

int symb;

while ((symb = getchar()) != EOF) {

switch (symb) {

case '+': {

getchar();

int newElem;

scanf("%d", &newElem);

getchar();

pushBack(&list, newElem);

break;

}

case 'p': {

printList(&list);

break;

}

case '-': {

popBack(&list);

break;

}

case 't': {

getchar();

int elem;

scanf("%d", &elem);

deletePrevAndNext(&list, elem);

break;

}

case 'i': {

getchar();

int newElem;

scanf("%d", &newElem);

getchar();

int position;

scanf("%d", &position);

insert(&list, newElem, position);

break;

}

case 'd':{

getchar();

int position;

scanf("%d", &position);

deleteByValue(&list, deleteByPosition(&list, position));

break;

}

case 'l':{

int len = getListLength(&list);

printf("Length: %d\n", len);

}

}

}

destroyList(&list);

return 0;

}

+ add el., '-' pop el. from the end of list, t - task, d - delete from list, i - put el. in entered position, p - print, l – length

Add and delete: 1 - for WHITE, 2 - for BLUE, 3 - for RED, 4 - for YELLOW, 5 - for GREEN

+ 1

+ 2

p

list: WHITE BLUE

+ 3

p

list: WHITE BLUE RED

l

Length: 3

+ 4

p

list: WHITE BLUE RED YELLOW

l

Length: 4

f 2

p

list: WHITE BLUE RED YELLOW

t 2

p

list: BLUE YELLOW

+ 1

+ 2

+ 3

+ 4

p

list: BLUE YELLOW WHITE BLUE RED YELLOW

l

Length: 6

+ add el., '-' pop el. from the end of list, t - task, d - delete from list, i - put el. in entered position, p - print, l – length

Add and delete: 1 - for WHITE, 2 - for BLUE, 3 - for RED, 4 - for YELLOW, 5 - for GREEN

+ 2

+ 3

+ 4

p

list: BLUE RED YELLOW

t 3

p

list: RED

l

Length: 1

+ 1

+ 2

+ 4

+ 5

+ 3

p

list: RED WHITE BLUE YELLOW GREEN RED

l

Length: 6

d 1

p

list: RED BLUE YELLOW GREEN RED

d 1

p

list: RED BLUE YELLOW GREEN RED

d 2

p

list: RED YELLOW GREEN RED

+ 2

+ 3

p

list: RED YELLOW GREEN RED BLUE RED

l

Length: 6

d 1

p

list: RED YELLOW GREEN RED RED

t 2

p

list: RED YELLOW GREEN RED RED

+ add el., '-' pop el. from the end of list, t - task, d - delete from list, i - put el. in entered position, p - print, l – length

Add and delete: 1 - for WHITE, 2 - for BLUE, 3 - for RED, 4 - for YELLOW, 5 - for GREEN

+ 1

+ 2

+ 3

+ 4

+ 1

+ 3

p

list: WHITE BLUE RED YELLOW WHITE RED

t 2

p

list: BLUE YELLOW WHITE RED

+ 1

+ 2

+ 3

+ 5

p

list: BLUE YELLOW WHITE RED WHITE BLUE RED GREEN

l

Length: 8

t 1

p

list: BLUE WHITE WHITE BLUE RED GREEN

l

Length: 6

d 1

d 3

p

list: BLUE BLUE GREEN

1. **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. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  | или |  |  |  |  |  |
|  | дом. |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы

1. **Выводы**

Я научился обрабатывать линейные списки на Си.

Недочёты при выполнении задания могут быть устранены следующим образом: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_