Отчет по лабораторной работе №VIII по курсу практикум на ЭВМ

Студент группы М8О-101Б-20 Ядров Артем Леонидович, № по списку 28

Контакты www, e-mail, icq, skype temayadrow@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: вещественный. Вид списка: линейный однонаправленный с барьерным элементом. Нестандартное действие: выполнить попарный обмен значениями элементов списка

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 _____
_____

5 Программное обеспечение (лабораторное):
Операционная система семейства Unix, 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
Местонахождение и имена файлов программ и данных stud/208104
_____
Программное обеспечение ЭВМ студента, если использовалось:
Операционная система семейства Unix, 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 home/Temi4
_____

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 Item {
  struct Item *next;
  float data;
  };
  Структура элемента списка. Содержит указатель на следующий элемент, а также значение текущего элемента.
- typedef struct {
  struct Item *node;
  } Iterator;
  Структура итератора. Содержит указатель на элемент списка.
- typedef struct {
  struct Item *head;
  int size;
- } List;
  Структура списка. Содержит указатель на барьерный элемент (терминатор), 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.
- float Fetch(const Iterator *i)
  Функция возвращает значение элемента, на который указывает итератор.
- 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 возвращает указатель на последний элемент списка. В данном случае терминатор.
- bool Empty(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 task(List *l)
  Функция выполнения обмена значениями. С помощью итераторов пробегает по списку через одного, выполняя обмен значениями.
- 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

2 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

3 Удаление из списка (Delete from list)
Запрашивает значение удаляемого элемента, выполняет поиск предшествующего, а затем вызывает функцию Delete

4 Размер списка (Size)
Вызывает функцию Size

5 Задание (Task)
Вызывает функцию task

6 Выход
Выходит из меню, завершая программу.

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

Тесты:

- 1
- 1 2
- 1 2 3
- 1 2 3 4 5 6 7 8 9
- 2 1 4 3 6 5 8 7 10 9
- 2 3 1

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, подписанный преподавателем).

```
[Temi4@localhost KP8]$ cat list.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct Item {
   struct Item *next;
   float data;
};

typedef struct {
   struct Item *node;
} Iterator;

bool Equal(Iterator *lhs, Iterator *rhs) {
   return lhs->node == rhs->node;
}

Iterator *Next(Iterator *i) {
   i->node = i->node->next;
   return i;
}

float Fetch(const Iterator *i) {
   return i->node->data;
}

void Store(const Iterator *i, const float t) {
   i->node->data = t;
}

typedef struct {
   struct Item *head;
   int size;
} List;

void Create(List *l) {
   l->head = malloc(sizeof(struct Item));
   l->head->next = l->head;
   l->size = 0;
}

Iterator First(const List *l) {
   Iterator res = {l->head->next};
   return res;
}

Iterator Last(const List *l) {
   Iterator res = {l->head};
   return res;
}

bool Empty(const List *l) {
   Iterator fst = First(l);
   Iterator lst = Last(l);
   return Equal(&fst, &lst);
}

Iterator Insert(List *l, Iterator *i, const float t) {
   Iterator res = {malloc(sizeof(struct Item))};
```

```
    if (!res.node) {
        return Last(l);
    }
    res.node->data = t;
    res.node->next = i->node->next;
    i->node->next = res.node;
    l->size++;
    return res;
}

Iterator Delete(List *l, Iterator *i) {
    Iterator res = Last(l);
    Iterator next = {i->node->next};
    if (Equal(&res, &next)) {
        return res;
    }
    res.node = i->node->next->next;
    struct Item *a = i->node->next;
    i->node->next = res.node;
    free(a);
    l->size--;
    return res;
}

int Size(const List *l) {
    return l->size;
}

void print_list(const List *l) {
    if (l->head == NULL) {
        printf("List doesn't exists\n");
        return;
    }
    Iterator lst = Last(l);
    for (Iterator i = First(l); !Equal(&i, &lst); Next(&i)) {
        printf("%.4f ", Fetch(&i));
    }
    printf("\n");
}

void task(List *l) {
    Iterator lst = Last(l);
    for (Iterator i = First(l); !Equal(&i, &lst); Next(&i), Next(&i)) {
        if (i.node->next == lst.node) {
            return;
        }
        float c = i.node->data;
        Iterator j = {i.node->next};
        Store(&i, Fetch(&j));
        Store(&j, c);
    }
}

Iterator search_prev(const List *l, const int n) {
    Iterator res = Last(l);
    for (int i = 0; i <= n - 1; i++) {
        Next(&res);
    }
    return res;
}

void Destroy(List *l) {
```

```
    struct Item *i = l->head->next;
    while (i != l->head) {
        struct Item *pi = i;
        i = i->next;
        free(pi);
    }
    free(l->head);
    l->head = 0;
    l->size = 0;
}

int main() {
    List *l = malloc(sizeof(List));
    Create(l);
    int num = 0;
    int choose;
    bool g = true;
    while (g) {
        printf("1. Print list\t 2. Insert in list\t 3. Delete from list\t 4. Size\t 5. Task\t 6. Exit\n");
        scanf("%d", &choose);
        switch (choose) {
            case 1: {
                print_list(l);
                break;
            }
            case 2: {
                if (l->head == NULL) {
                    Create(l);
                }
                Iterator i = Last(l);
                float val;
                printf("Write value\n");
                scanf("%f", &val);
                if (l->size != 0) {
                    printf("Write the number where you want insert element\n");
                    scanf("%d", &num);
                    if (0 <= num && num <= Size(l)) {
                        i = search_prev(l, num);
                    } else {
                        printf("Element with %d number doesn't exists\n", num);
                        break;
                    }
                }
                Insert(l, &i, val);
                break;
            }
            case 3: {
                if (l->head == NULL) {
                    printf("List doesn't exists\n");
                    break;
                }
                float prev;
                printf("Write number of deleted element\n");
                scanf("%d", &num);
                if (num == Size(l)) {
                    Destroy(l);
                    break;
                }
                if (0 <= num && num < Size(l)) {
                    Iterator i = search_prev(l, num);
                    Delete(l, &i);
                } else {
```

```
                printf("Element with %d number doesn't exists\n", num);
            }
            break;
        }
        case 4: {
            if (l->head == NULL) {
                printf("List doesn't exists\n");
                break;
            }
            printf("%d\n", Size(l));
            break;
        }
        case 5: {
            if (l->head == NULL) {
                printf("List doesn't exists\n");
                break;
            }
            if (Size(l) != 1) {
                task(l);
            }
            break;
        }
        case 6: {
            g = false;
            break;
        }
        default: {
            printf("Wrong answer\n");
        }
      }
   }
   return 0;
}
[Temi4@localhost KP8]$ gcc list.c
[Temi4@localhost KP8]$ ./a.out
1. Print list    2. Insert in list    3. Delete from list    4. Size    5. Task   6. Exit
1

1. Print list    2. Insert in list    3. Delete from list    4. Size    5. Task   6. Exit
2
Write value
1
1. Print list    2. Insert in list    3. Delete from list    4. Size    5. Task   6. Exit
1
1.0000
1. Print list    2. Insert in list    3. Delete from list    4. Size    5. Task   6. Exit
4
1
1. Print list    2. Insert in list    3. Delete from list    4. Size    5. Task   6. Exit
5
1. Print list    2. Insert in list    3. Delete from list    4. Size    5. Task   6. Exit
1
1.0000
1. Print list    2. Insert in list    3. Delete from list    4. Size    5. Task   6. Exit
3
Write number of deleted element
0
1. Print list    2. Insert in list    3. Delete from list    4. Size    5. Task   6. Exit
1

1. Print list    2. Insert in list    3. Delete from list    4. Size    5. Task   6. Exit
3
```

Write number of deleted element
0
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
List doesn't exists
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
1
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
2
Write the number where you want insert element
1
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
1.0000 2.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
4
2
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
5
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
2.0000 1.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
3
Write number of deleted element
0
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
1.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
2
Write the number where you want insert element
1
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
3
Write the number where you want insert element
2
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
1.0000 2.0000 3.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
5
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
2.0000 1.0000 3.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
5
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
1.0000 2.0000 3.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
4

Write the number where you want insert element
3
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
5
Write the number where you want insert element
4
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
6
Write the number where you want insert element
5
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
7
Write the number where you want insert element
6
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
8
Write the number where you want insert element
7
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
9
Write the number where you want insert element
8
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
1.0000 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000 8.0000 9.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
4
9
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
5
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
2.0000 1.0000 4.0000 3.0000 6.0000 5.0000 8.0000 7.0000 9.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
10
Write the number where you want insert element
8
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
2.0000 1.0000 4.0000 3.0000 6.0000 5.0000 8.0000 7.0000 10.0000 9.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
11
Write the number where you want insert element
100
Element with 100 number doesn't exists
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
2.0000 1.0000 4.0000 3.0000 6.0000 5.0000 8.0000 7.0000 10.0000 9.0000

1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
5
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
1.0000 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000 8.0000 9.0000 10.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
3
Write number of deleted element
4
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
1.0000 2.0000 3.0000 4.0000 6.0000 7.0000 8.0000 9.0000 10.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
3
Write number of deleted element
9
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
List doesn't exists
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
1
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
2
Write the number where you want insert element
0
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
2.0000 1.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
2
Write value
3
Write the number where you want insert element
1
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
2.0000 3.0000 1.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
5
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
1
3.0000 2.0000 1.0000
1. Print list 2. Insert in list 3. Delete from list 4. Size 5. Task 6. Exit
6

9 **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

10 **Замечания автора** по существу работы_____

_____

_____

_____

_____

_____

11 Выводы

Я научился обрабатывать линейные списки на Си._____

_____

_____

_____

Недочёты при выполнении задания могут быть устранены следующим образом:

_____

_____

_____

Подпись студента _____