Отчет по лабораторной работе № VIII по курсу
Алгоритмы и структуры данных

Студент группы М8О-103Б-22 Ахметшин Булат Рамилевич, № по списку 2

Контакты www, e-mail, icq, skype ahmbulat04@yandex.ru

Работа выполнена: 05.05.2023 г.

Преподаватель: доцент каф. 806 Никулин С.П.

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 списка на динамические структуры. _____

3. **Задание** (*вариант № 2 - {4, 2, 3}*): Вид списка - линейный однонаправленный, тип элемента - строковый, нестандартное действие - удаление подсписка слева и справа от данного элемента. _____

4. **Оборудование** (*лабораторное*):
ЭВМ _____, процессор _____, имя узла сети _____ с ОП _____ Мб,
НМД _____ Мб. Терминал _____ адрес _____. Принтер _____
Другие устройства _____

Оборудование ПЭВМ студента, если использовалось:
Процессор Intel(R) Core(TM) i7-10510U____ с ОП 8 ГБ____ НМД SSD 512 ГБ____ . Монитор Встроенный 1920x1080
Другие устройства _____

5. **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 UNIX_____, наименование Ubuntu_____ версия 22.04_____
интерпретатор команд GNU bash_____ версия 5.1.16_____
Система программирования Visual Studio Code_____ версия 1.77.3_____
Редактор текстов Sublime Text 3_____ версия 3211_____
Утилиты операционной системы Стандартные утилиты OS Linux_____
Прикладные системы и программы Редактор текста nano._____

Местонахождение и имена файлов программ и данных на домашнем компьютере
/home/bulat/Studying/prprm/cr/VIII/

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

Алгоритм нестандартной задачи я реализую следующим образом:

(a) Найти элемент в списке с указанным значчением.
(b) Если требуется удалить список слева - вызвать функцию удаления подсписка от его начала до найденного элемента.
(c) Если Требуется удалить список справа - вызвать функцию удаления подсписка от найденного элемента до конца.

Функция удаления подсписка принимает в аргументы правую и левую границы списка и удаляет все элементы, кроме крайнего правого.

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

Составить makefile, написать реализацию линейного одностороннего списка и требуемую нестандартную функцию.

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)

```
bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VIII$ ls
list.c  list.h  logs  main.c  makefile  tex
bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VIII$ cat makefile
CC = gcc
CFLAGS = -std=c99 -Wall -Wextra

main: main.o list.o
    $(CC) $(CFLAGS) -o main main.o list.o
main_o:
    $(CC) $(CFLAGS) -c main.c
deque_o:
    $(CC) $(CFLAGS) -c list.c
clean:
    rm -f *.o main
bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VIII$ cat list.h
#ifndef _LIST_H
#define _LIST_H

#include <stdbool.h>
#include <inttypes.h>
#include <stdlib.h>
#include <stdio.h>

typedef uint64_t size_t;

typedef struct {
    char* p;
    size_t l;
} string;

string make_string(char*);
bool strings_eq(string, string);

typedef struct node {
    struct node* r;
    string s;
} node;

node* make_node(string);

void link_node(node*, node*, node*);

void link_nodes(node*, node*);

void node_free(node*);

typedef struct {
    node* begin;
    node* back;
    size_t size;
} list;

void list_create_empty(list*);

void list_add(list*, string, size_t);

node* list_find_node(list*, string);

void list_remove(list*, string);

void list_remove_subtree(list*, node*, node*);

void list_free(list*);

size_t list_len(node*, node*);

void list_print(list*);

void list_cut(list*, char, string);


#endifbulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VIII$ cat list.c
#include "list.h"

string make_string(char* p) {
    string s;
    s.p = p;
    s.l = 0;
    while (s.p[s.l] != '\0') {
```

```
        ++s.l;
    }
    return s;
}

bool string_eq(string a, string b) {
    if (a.l != b.l) {
        return false;
    }
    size_t i = 0;
    while (i < a.l) {
        if (a.p[i] != b.p[i]) {
            return false;
        }
        ++i;
    }
    return true;
}

node* make_node(string s) {
    node* p = (node*)malloc(sizeof(node));
    p->r = NULL;
    p->s = s;
    return p;
}

void link_node(node* l, node* m, node* r) {
    if (! (l == NULL))
        l->r = m;
    m->r = r;
}

void link_nodes(node* l, node* r) {
    if (! (l == NULL)) {
        l->r = r;
    }
}

void node_free(node* p) {
    free(p->s.p);
    p->s.p = NULL;
    p->s.l = 0;
    p->r = NULL;
}

void list_create_empty(list* l) {
    l->begin = l->back = NULL;
    l->size = 0;
}

void list_add(list* l, string s, size_t index) {
    node* _p = make_node(s);
    if (index < l->size) {
        node* p = l->begin;
        for (size_t i = 0; i < index; ++i) {
            p = p->r;
        }
        link_node(p, _p, p->r);
    } else {
        if (l->back == NULL) {
            l->begin = l->back = _p;
            l->size = 0;
        } else {
            link_node(l->back, _p, NULL);
            l->back = _p;
        }
    }
    l->size++;
}

node* list_find_node(list* l, string s) {
    node* p = l->begin;
    while (! (p == NULL) && ! string_eq(p->s, s)) {
        p = p->r;
    }
    return p;
}

void list_remove(list* l, string s) {
    if (! (l->begin == NULL) && string_eq(l->begin->s, s)) {
        node* r = l->begin;
        l->begin = l->begin->r;
        if (l->begin == NULL) {
            l->back = NULL;
        } else {
```

```
            link_nodes(l->begin, l->begin->r->r);
        }
        node_free(r);
        l->size--;
    }
    else {
        node* p = l->begin;
        while (!(p->r == NULL) && ! string_eq(p->r->s, s)) {
            p = p->r;
        }
        if (! (p->r == NULL)) {
            node* r = p->r;
            link_nodes(p, p->r->r);
            node_free(r);
            l->size--;
        }
    }
}

void list_remove_subtree(list* l, node* s, node* e) {
    node* p = l->begin,* r;
    if (p != s) {
        while (! (p->r == NULL) && ! (p->r == s)) {
            p = p->r;
        }
        if (p->r == NULL) {
            return;
        }
        if (e == NULL) {
            l->back = p;
        }
        r = p->r;
        p->r = e;
        p = r;
    } else {
        l->begin = e;
    }
    while (! (p->r == NULL) && ! (p->r == e)) {
        r = p;
        p = p->r;
        node_free(r);
        --l->size;
    }
    node_free(p);
    --l->size;
}

void list_free(list* l) {
    if (l->begin == NULL) {
        l->back = NULL;
        l->size = 0;
        return;
    }
    node* p = l->begin,* r;
    while (! (p->r == NULL)) {
        r = p;
        p = p->r;
        node_free(r);
    }
    node_free(p);
    l->begin = l->back = NULL;
    l->size = 0;
}

size_t list_len(node* l, node* r) {
    if (l == NULL || r == NULL) {
        return 0;
    }
    node* p = l;
    size_t len = 1;
    while (! (p == r)) {
        p = p->r; ++len;
    }
    return len;
}

void list_print(list* l) {
    printf("{");
    if (! (l->begin == NULL)) {
        node* p = l->begin;
        while (! (p->r == NULL)) {
            printf(" %s,", p->s.p);
            p = p->r;
        }
        printf(" %s ", p->s.p);
```

```
    }
    printf("}");
}

void list_cut(list* l, char side, string s) {
    node* p = list_find_node(l, s);
    if (p == NULL) {
        return;
    }
    if (side == 'l' && ! (p == l->begin)) {
        list_remove_subtree(l, l->begin, p);
    } else if (side == 'r' && ! (p->r == NULL)) {
        list_remove_subtree(l, p->r, NULL);
    }
}bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VIII$ cat main.c
#include <stdio.h>
#include <inttypes.h>

#include "list.h"

#define STRING_LENGTH 32

char* int64_t_to_str(int64_t x) {
    char* p;
    if (x == 0) {
        p = (char*)malloc(sizeof(char) * 2);
        p[0] = '0';
        p[1] = '\0';
    }
    else {
        int64_t _x = x;
        size_t l = 0;
        while (_x != 0) {
            _x /= 10;
            ++l;
        }
        if (x < 0) {
            p = (char*)malloc(sizeof(char) * (l + 2));
            p[0] = '-';
            ++l;
            x = -x;
        } else {
            p = (char*)malloc(sizeof(char) * (l + 1));
        }
        p[l] = '\0';
        while (x != 0) {
            p[--l] = (char)((x % 10) + 48);
            x /= 10;
        }
    }

    return p;
}

int main() {

    list l;

    list_create_empty(&l);

    int32_t command = 0;
    bool running = true;

    char* s;
    size_t size;


    while (running) {
        printf("Menu:\n0. Exit\n1. Add element to the list\n2. Remove element from the list\n3. Cut sides.\n4. Print list
        printf("Please, input command: ");
        scanf(" %d", &command);

        switch (command)
        {
        case 1:
            s = (char*)calloc(STRING_LENGTH, sizeof(char));
            printf("Input string to be added: ");
            size = scanf(" %s[^\n]", s);
            list_add(&l, (string){.l = size, .p = s}, l.size);
            break;
        case 2:
            s = (char*)calloc(STRING_LENGTH, sizeof(char));
            printf("Input string that is to be removed from the list: ");
            size = scanf(" %s[^\n]", s);
            list_remove(&l, (string){.l = size, .p = s});
```

```
            break;
        case 3:
            s = (char*)calloc(STRING_LENGTH, sizeof(char));
            printf("Input central element: ");
            size = scanf(" %s[^\n]", s);
            printf("Which side you wish to be cut? [l, r]: ");
            char side;
            scanf(" %c[^\n]", &side);
            if (! (side == 'l' || side == 'r')) {
                printf("Wrong side picked: %c\n", side);
            } else {
                list_cut(&l, side, (string){.l = size, .p = s});
            }
            break;
        case 4:
            list_print(&l);
            printf("\n");
            break;
        case 5:
            printf("List length: %ld\n", l.size);
            break;
        case 0:
            running = false;
            break;
        default:
            printf("Wrong command picked\n");
            break;
        }
    }

    list_free(&l);

    return 0;
}bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VIII$ make
gcc -std=c99 -Wall -Wextra   -c -o main.o main.c
gcc -std=c99 -Wall -Wextra   -c -o list.o list.c
gcc -std=c99 -Wall -Wextra -o main main.o list.o
bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VIII$ ./main
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 1
Input string to be added: aboba
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 1
Input string to be added: bobaba
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 1
Input string to be added: bobiba
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 1
Input string to be added: iuiuio
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 1
Input string to be added: uauaua
Menu:
0. Exit
```

```
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 4
{ aboba, bobaba, bobiba, iuiuio, uauaua }
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 2
Input string that is to be removed from the list: bobiba
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 4
{ aboba, bobiba, iuiuio, uauaua }
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 5
List length: 4
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 3
Input central element: bobiba
Which side you wish to be cut? [l, r]: r
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 4
{ aboba, bobiba }
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 3
Input central element: bobiba
Which side you wish to be cut? [l, r]: l
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 4
{ bobiba }
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 2
Input string that is to be removed from the list: bobiba
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
```

```
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 4
{}
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 5
List length: 0
Menu:
0. Exit
1. Add element to the list
2. Remove element from the list
3. Cut sides.
4. Print list
5. Print list length
Please, input command: 0
```

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 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:** в ходе этой лабораторной работы я получил опыт реализации линейного одностороннего списка.

12. Недочёты при выполнении задания могут быть устранены следующим образом: _____

Подпись студента _____