**Отчет по лабораторной работе № 26**

по курсу “Фундаментальная информатика”

Студент группы М8О-103Б-21

Батулин Евгений Андреевич, № по списку 2

Контакты: e-mail: uggin@inbox.ru, telegram: @uggin0

Работа выполнена: «14» мая 2021 г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

Отчет сдан « » \_\_\_\_\_\_\_\_\_20\_\_ г., итоговая оценка \_\_\_\_\_\_\_\_

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

1. **Тема:** Абстрактные типы данных. Рекурсия. Модульное программирование на языке Си.
2. **Цель работы:** составить и отладить модуль определений и модуль реализации по заданной схеме модуля определений для абстрактного (пользовательского) типа данных (стека, очереди, списка или дека, в зависимости от варианта задания), составить программный модуль, сортирующий экземпляр указанного абстрактного типа данных заданным методом, используя только операции, импортированные из модуля UUDT
3. **Задание:** составить и отладить модуль определений и реализации линейного списка, составить модуль процедуры по поиску в списке двух элементов, идущих подряд, переставляющий данные элементы между собой при выполнении условия (первый больше второго), выполняющий сортировку методом пузырька
4. **Оборудование** (студента):

Процессор *Intel Core i9-9980HK ES(QQLS), 8c/16t @ 4.4GHz* с ОП *32768* Мб, НМД *6656* Гб. Монитор *1920x1080*

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

Операционная система семейства: *Windows*, наименование: *10*, версия *1809 LTSC*

интерпретатор команд: *MSYS* версия *1.3.0.0*.

Система программирования -- версия --**,** редактор текстов *Visual Studio Code*, версия *1.66.2*

Утилиты операционной системы:

Прикладные системы и программы: gcc

1. **Идея, метод, алгоритм** решения задачи

Первоначально необходимо сделать модуль определений. Я создал подобный модуль согласно примеру из условия работы. Кроме структуры узлов линейного списка, я создам структуру самого линейного списка, в которой буду определяться указатели на начальный и конечный узел списка, что несколько упростит реализацию операций редактирования и навигации по нему. При создании пустого списка указатели на его начало и конец приравниваются, а значения этих узлов приравниваются к NULL. При вставке/удалении узла поиск узла с нужным значением производится при помощи дополнительного узла, выступающего в роли указателя на другие узлы внутри списка.

1. **Сценарий выполнения работы**
2. Запуск среды программирования
3. Создание программы
4. Проверка работоспособности программы на различных данных, вводимых человеком
5. Отладка
6. Протоколирование работы отлаженной программы
7. Завершение работы

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Описание тестируемого случая** |
| 1  6 10  7 90  2 10 15  10  11 | [ (0 - 10)(1 - 15)(2 - 90) ] | Проверка работы программы в типичной ситуации |
| 1 10 1 10 20 1 20 40 1 20 34  1 34 45 1 20 28  3 34 2 | 10  20  40  28 | Попытка удаления ветви целиком, проверка на сохранения порядка у младших узлов |
| 1 5915  1 5915 932  14  1 5915 540  1 932 14  14 | 1  0 | Проверка функции по варианту задания |

1. **Распечатка протокола**

list.h

#ifndef LIST\_H

#define LIST\_H

#include <stdbool.h>

#include <stdlib.h>

typedef int Item;

typedef struct listnode {

    Item data;

    struct listnode \*next;

} ListNode;

typedef struct {

    ListNode \*head;

    ListNode \*current;

    ListNode \*tail;

} List;

List \*list\_create();

void list\_insert(List \*list, Item key, Item value);

void list\_erase(List \*list, Item value);

bool list\_is\_empty(List \*list);

void list\_destroy(List \*list);

void list\_push\_front(List \*list, Item value);

void list\_push\_back(List \*list, Item value);

void list\_pop\_front(List \*list);

void list\_pop\_back(List \*list);

void list\_print(List \*list);

int list\_size(List \*list);

int get\_length(List \*list);

void sort\_bubble(List \*list, int size);

#endif

list.c

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

#include "list.h"

typedef int Item;

List \*list\_create()

{

    List \*list = (List\*) malloc(sizeof(List));

    list->head = (ListNode\*) malloc(sizeof(ListNode));

    list->head = NULL;

    list->tail = NULL;

    list->tail = list->head;

    return list;

}

void list\_insert(List \*list, Item key, Item value)

{

    ListNode \*to\_insert = (ListNode\*) malloc(sizeof(ListNode));

    to\_insert->data = value;

    if (!list\_is\_empty(list)) {

        ListNode \*ptr = list->head;

        printf("sup1\n");

        if (list->head == NULL){

        }

        printf("%d", ptr->data);

        while(ptr->data != key){

        ptr = ptr->next;

        }

        ListNode \*relink = ptr->next;

        ptr->next = to\_insert;

        ptr->next->next = relink;

    }

}

void list\_erase(List \*list, Item value)

{

    ListNode \*ptr = list->head;

    ListNode \*ptr\_prev = NULL;

    while(ptr->data != value){

        if (ptr->next == NULL){

            exit; //??

        } else {

            ptr\_prev = ptr;

            ptr = ptr->next;

        }

    }

    if (ptr == list->head) {

        ListNode \*to\_free = list->head;

        list->head = list->head->next;

        free(to\_free);

    } else {

        ListNode \*to\_free = ptr;

        ptr\_prev->next = ptr->next;

        free(to\_free);

    }

}

bool list\_is\_empty(List \*list)

{

    return (list->head == NULL);

}

void list\_destroy(List \*list)

{

    ListNode \*n = list->head->next;

    while (list->head != NULL)

    {

        n = list->head->next;

        free(list->head);

        list->head = n;

    }

    list->head = NULL;

}

void list\_push\_front(List \*list, Item value)

{

    ListNode \*pushed = (ListNode\*) malloc(sizeof(ListNode));

    pushed->data = value;

    if(list\_is\_empty(list)){

        list->head = pushed;

        list->head->next = NULL;

        list->tail = list->head;

    } else {

        pushed->next = list->head;

        list->head = pushed;

    }

}

void list\_pop\_front(List \*list)

{

    ListNode \*ptr = list->head;

    ListNode \*to\_pop = list->head;

    if (list->head == NULL) {

        free(list->head);

        list->head = NULL;

    } else {

        list->head = list->head->next;

        free(to\_pop);

    }

}

void list\_push\_back(List \*list, Item value)

{

    ListNode \*pushed = (ListNode\*) malloc(sizeof(ListNode));

    pushed->data = value;

    pushed->next = NULL;

    if(list\_is\_empty(list)){

        list->head = pushed;

        list->tail = list->head;

    } else {

        list->tail->next = pushed;

        list->tail = list->tail->next;

    }

}

void list\_pop\_back(List \*list)

{

    ListNode \*ptr = list->head;

    ListNode \*ptr\_prev = NULL;

    while(ptr->next != NULL){

        ptr\_prev = ptr;

        ptr = ptr->next;

    }

    if (ptr == list->head) {

        free(list->head);

        list->head = NULL;

    } else {

        ListNode \*to\_free = ptr;

        ptr\_prev->next = NULL;

        free(to\_free);

    }

}

void list\_print(List \*list) {

   ListNode \*ptr = list->head;

   printf("\n[ ");

   int i = 0;

   while(ptr != NULL) {

      printf("(%d - %d)", i, ptr->data);

      ptr = ptr->next;

      i++;

   }

   printf(" ]");

}

int list\_size(List \*list)

{

    int size = 0;

    ListNode \*ptr = list->head;

    for(ptr = list->head; ptr != NULL; ptr = ptr->next) {

      size++;

    }

    return size;

}

void sort\_bubble(List \*list, int size){

    if (list->head == NULL){

            exit; //??

    } else {

        int i, j, k, tempData;

        k = size;

        ListNode \*ptr = list->head;

        ListNode \*next = list->head->next;

        for(i = 0; i < size - 1; i++) {

            ptr = list->head;

            next = list->head->next;

            for (j = 0; j < size - i - 1; j++){

                if ( ptr->data > next->data ) {

                    tempData = ptr->data;

                    ptr->data = next->data;

                    next->data = tempData;

                }

                ptr = ptr->next;

                next = next->next;

            }

        }

    }

}

bool isInt(const char\*str) {

    while(\*str)  {

        if((\*str< '0' || \*str > '9') && \*str != '-' && \*str != '.')

            return false;

        \*str++;

    }

    return true;

}

main.c

int main() {

    printf("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

    printf("|                       List Editor                      |\n");

    printf("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

    printf("|Create List - |1| Insert - |2| Delete - |3| Empty? - |4||\n");

    printf("|     Destroy - |5| Push front - |6| Push back - |7|     |\n");

    printf("|       Pop front - |8| Pop back - |9| Exit - |0|        |\n");

    printf("|            Bubble sort - |10| Print - |11|             |\n");

    printf("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

    char input[] = "";

    bool execute = true;

    bool isListCreated = false;

    int task = -1;

    List \*list = list\_create();

    int value = 0;

    while (execute) {

        char input[] = "";

        printf("\n");

        Item data = 0;

        Item key = 0;

        int size = 0;

        Item value = 0;

        scanf("%s", input);

        if (isInt(input)) {

            if (!strcmp("0", input)) task = 0;

            if (!strcmp("1", input)) task = 1;

            if (!strcmp("2", input)) task = 2;

            if (!strcmp("3", input)) task = 3;

            if (!strcmp("4", input)) task = 4;

            if (!strcmp("5", input)) task = 5;

            if (!strcmp("6", input)) task = 6;

            if (!strcmp("7", input)) task = 7;

            if (!strcmp("8", input)) task = 8;

            if (!strcmp("9", input)) task = 9;

            if (!strcmp("10", input)) task = 10;

            if (!strcmp("11", input)) task = 11;

        } else {

            printf("Try something else :) ");

            task = -1;

        }

        switch (task) {

            case 0:

                execute = false;

                if (isListCreated) {

                    list\_destroy(list);

                }

                break;

            case 1:

                //already created

                break;

            case 2:

                scanf("%d", &key);

                scanf("%s", input);

                if (isInt(input)){

                    data = atoi(input);

                    list\_insert(list, key, data);

                } else {

                    printf("Incorrect input");

                }

                break;

            case 3:

                scanf("%s", input);

                if (isInt(input)){

                    key = atoi(input);

                    list\_erase(list, key);

                } else {

                    printf("Incorrect input");

                }

                break;

            case 4:

                printf("Emptiness - %d" , list\_is\_empty(list));

                break;

            case 5:

                list\_destroy(list);

                list = list\_create();

                break;

            case 6:

                scanf("%s", input);

                if (isInt(input)){

                    value = atoi(input);

                    list\_push\_front(list, value);

                } else {

                    printf("Incorrect input");

                }

                break;

            case 7:

                scanf("%s", input);

                if (isInt(input)){

                    value = atoi(input);

                    list\_push\_back(list, value);

                } else {

                    printf("Incorrect input");

                }

                break;

            case 8:

                list\_pop\_front(list);

                break;

            case 9:

                list\_pop\_back(list);

                break;

            case 10:

                size = list\_size(list);

                sort\_bubble(list, size);

                break;

            case 11:

                list\_print(list);

                break;

            default:

                printf("Incorrect input");

                break;

        }

    }

    return 0;

}

**9. Дневник отладки**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
| 1 | дом | 14\05\2022 | 13:54 | Insert вызывало критическую ошибку при пустом листе | Добавлена проверка на пустоту листа |  |

1. **Замечания автора** по существу работы

Данная работа понравилась мне тем, что в её процессе можно было отработать навыки работы с самыми популярными структурами данных, что будет полезным подспорьем при создании каких-либо практических проектов, ведь с большой долей вероятности в них придётся использовать одну из структур из этой работы. Знание того, как структура работает изнутри – гарантия её корректного воспроизведения в реальном проекте.

1. **Выводы**

Подводя итог, в процессе данной работы я реализовал часто встречающийся в рабочих задачах абстрактный тип данных, попрактиковал модульное программирование и улучшил свои навыки программирования на Си.

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