**Отчет по курсовой работе № 8** по курсу

Практикум программирование

Студент группы: М8О - 110Б - 23, Гула Дмитрий Александрович, № по списку: 6,

Контакты: guladmitriy1331@gmail.com

Работа выполнена: ««18» мая 2024г.

Преподаватель: каф. 806 Почечура А. А.

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

Отчет сдан «25» мая 2024 г., итоговая оценка \_\_\_\_\_

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

1. **Тема:** Обработка линейного списка заданной организации
2. **Цель работы: Составить и отладить программу на языке Си для обработки линейного списка с использованием итераторов**
3. **Задание (**вариант 15 ): Преобразовать линейный двунаправленный список с барьерным элементом, тип элементов — вещественные числа, с применением итераторов. Реализовать выполнение одного нестандартного и четырех стандартных действий:
4. Печать списка.
5. Вставка нового элемента в список.
6. Удаление элемента из списка.
7. Подсчёт длины списка.
8. Удалить каждый k-ый элемент списка (нестандартное действие).
9. **Оборудование** (лабораторное):

ЭВМ , процессор . имя узла сети с ОП Мб, НМД Мб. Терминал

адрес . Принтер Другие устройства



*Оборудование ПЭВМ студента, если использовалось:*

Процессор \_Intel Core i3-12500H \_ с ОП 8 Гб НМД 256 Гб. Монитор 1920x1080~144Hz Другие устройства



1. **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 \_Linux\_, наименование \_Ubuntu\_ версия 22.04.2 интерпретатор команд \_GNU bash\_ версия 5.1.16.

Система программирования С. Редактор текстов emacs версия 29.1

Утилиты операционной системы

Прикладные системы и программы Emacs

Местонахождение и имена файлов программ и данных на домашнем компьютере /home/

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

**Идея**

Целью данной курсовой работы является разработка программы на языке Си для обработки линейного двунаправленного списка с барьерным элементом, тип элементов — вещественные числа. Для навигации по списку используются итераторы. Программа должна поддерживать выполнение одного нестандартного действия и четырех стандартных действий: печать списка, вставка нового элемента, удаление элемента, подсчёт длины списка, а также удаление каждого k-го элемента списка.

**Метод**

Программа состоит из трех основных файлов:

1. Заголовочный файл list\_and\_struct.h, содержащий определения структур и прототипы функций.
2. Файл listFunctions.c, в котором реализованы функции для работы со списком.
3. Файл main.c, который предоставляет интерфейс пользователя и демонстрирует работу программы.
4. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

1.Инициализация списка:

* Функция bdBarList\_init создает новый список с начальной емкостью, указанной в параметре count. Список состоит из элементов типа bdBarList\_elem, которые хранят значение val, индексы следующего и предыдущего элементов.
* Буфер элементов списка создается с помощью malloc.

2.Определение итератора:

* Итератор представлен структурой bdBarList\_iter, которая хранит индекс предыдущего элемента и указатель на список.

3.Вставка элемента перед указанным итератором:

* Функция insert\_before вставляет новый элемент перед указанным итератором. Если список достигает своей максимальной емкости, происходит увеличение емкости с помощью функции grow\_buff.

4.Удаление элемента по итератору:

* Функция bdBarList\_remove удаляет элемент, на который указывает итератор. Если после удаления список становится слишком маленьким, его емкость уменьшается с помощью shrink\_buff.

5.Освобождение памяти:

* Функция bdBarList\_destroy освобождает выделенную память для буфера и самого списка.

6.Операции с элементами списка:

* del\_every\_kth удаляет каждый k-й элемент списка.
* insert вставляет новый элемент перед указанной позицией.
* removeEl удаляет элемент на указанной позиции.

7.Ввод и вывод:

* В функции main программа открывает меню команд и ожидает ввода пользователя. В зависимости от выбранной команды (1, 2, 3, 4 или h), программа выполняет соответствующую операцию.

8.Печать списка:

* Функция print\_list выводит содержимое списка на экран.

**Сценарий выполнения работы:**

1. Запуск программы:

* Программа выводит приветственное сообщение "Программа начинает работу" и инициализирует список с начальной емкостью 100.

2.Ввод команд:

* Пользователь вводит команды, используя меню:
* h - открыть меню.
* 1 [value, position] - вставить value перед позицией position.
* 2 [position] - удалить элемент на позиции position.
* 3 - распечатать список.
* 4 [k] - удалить каждый k-й элемент.

3.Пример работы:

* Пользователь вводит, например, 1 10 3, чтобы вставить значение 10 перед 3-м элементом списка.
* Далее, 3, чтобы распечатать список.

4.Завершение работы:

* Программа завершает работу при получении конца ввода (EOF).

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).

#ifndef C\_BIDIRECTIONAL\_WITH\_BARRIER\_ELEMENT\_H  
#define C\_BIDIRECTIONAL\_WITH\_BARRIER\_ELEMENT\_H  
  
#include <stdlib.h>  
#include <stdbool.h>  
#include <assert.h>  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <stdbool.h>  
#include <math.h>  
  
  
typedef struct bdBarList bdBarList;  
typedef struct bdBarList\_elem bdBarList\_elem;  
typedef bdBarList\* bdBarList\_handle;  
typedef struct bdBarList\_iter bdBarList\_iter;  
  
  
  
bdBarList\_handle bdBarList\_init(int count);  
bdBarList\_iter bdBarList\_iter\_next(bdBarList\_iter it);  
bdBarList\_iter bdBarList\_iter\_prev(bdBarList\_iter it);  
bool bdBarList\_iter\_equal(bdBarList\_iter it1, bdBarList\_iter it2);  
bdBarList\_iter bdBarList\_iter\_begin(bdBarList\_handle l);  
bdBarList\_iter bdBarList\_iter\_end(bdBarList\_handle l);  
double bdBarList\_iter\_val(bdBarList\_iter it);  
bool bdBarList\_is\_empty(bdBarList\_handle l);  
void insert\_before(bdBarList\_iter\* it, double val);  
void print\_list(bdBarList\_handle l);  
bdBarList\_iter get\_n\_iter(int n, bdBarList\_handle l);  
bool grow\_buff(bdBarList\_handle l);  
void shrink\_buff(bdBarList\_handle l);  
void bdBarList\_pop(bdBarList\_iter\* it);  
void bdBarList\_destroy(bdBarList\_handle l);  
void bdBarList\_remove(bdBarList\_iter\* it);  
void del\_every\_kth(bdBarList\_handle l, int k);  
void insert(bdBarList\_handle l, int pos, double val);  
void removeEl(bdBarList\_handle l, int pos);  
void open\_menu();  
  
  
  
#endif //C\_BIDIRECTIONAL\_WITH\_BARRIER\_ELEMENT\_H

#include "bidirectional-with-barrier-element.h"  
  
  
struct bdBarList\_elem {  
 double val;  
 int next;  
 int prev;  
};  
  
struct bdBarList {  
 int head;  
 int tail;  
 bdBarList\_elem\* buffer;  
 int capacity;  
 int current\_size;  
 int last\_empty;  
};  
  
struct bdBarList\_iter {  
 int prev;  
 bdBarList\_handle list;  
};  
  
  
  
bdBarList\_handle bdBarList\_init(int count) {  
 bdBarList\_handle list = malloc(sizeof(bdBarList));  
 if (list == NULL) {  
 return NULL;  
 }  
 list->head = -1;  
 list->tail = -1;  
 list->current\_size = 0;  
 list->capacity = count;  
 list->buffer = malloc(sizeof(bdBarList\_elem) \* count);  
 if (list->buffer == NULL) {  
 free(list);  
 return NULL;  
 }  
  
 for (int i = 0; i < count - 1; i++) {  
 ((list->buffer)[i]).next = i + 1;  
 ((list->buffer)[i]).prev = i - 1;  
 }  
 list->buffer[0].prev = -1;  
 list->buffer[count - 1].next = 0;  
 list->last\_empty = 0;  
 return list;  
}  
  
  
bdBarList\_iter bdBarList\_iter\_next(bdBarList\_iter list\_iter) {  
 if (list\_iter.prev == -1) {  
 return (bdBarList\_iter) { list\_iter.list->head, list\_iter.list };  
 }  
 return (bdBarList\_iter) { list\_iter.list->buffer[list\_iter.prev].next, list\_iter.list };  
}  
  
bdBarList\_iter bdBarList\_iter\_prev(bdBarList\_iter list\_iter) {  
 if (list\_iter.prev == -1) {  
 return (bdBarList\_iter) { list\_iter.list->head, list\_iter.list };  
 }  
 return (bdBarList\_iter) { list\_iter.list->buffer[list\_iter.prev].prev, list\_iter.list };  
}  
  
bool bdBarList\_iter\_equal(bdBarList\_iter it1, bdBarList\_iter it2) {  
 return (it1.prev == it2.prev && it1.list == it2.list);  
}  
  
  
bdBarList\_iter bdBarList\_iter\_begin(bdBarList\_handle list) {  
 return (bdBarList\_iter) { -1, list };  
}  
  
bdBarList\_iter bdBarList\_iter\_end(bdBarList\_handle list) {  
 return (bdBarList\_iter) { list->tail, list };  
}  
  
double bdBarList\_iter\_val(bdBarList\_iter list\_iter) {  
 if (list\_iter.prev == -1) {  
 return (list\_iter.list->buffer)[list\_iter.list->head].val;  
 }  
 else {  
 return (list\_iter.list->buffer)[list\_iter.list->buffer[list\_iter.prev].next].val;  
 }  
}  
  
  
bool bdBarList\_is\_empty(bdBarList\_handle list) {  
 return list->current\_size == 0;  
}  
  
void insert\_before(bdBarList\_iter\* list\_iter, double val) {  
 if (list\_iter->list->current\_size == list\_iter->list->capacity) {  
 if (!grow\_buff(list\_iter->list)) {  
 return;  
 }  
 }  
 bdBarList\_elem new\_elem;  
 new\_elem.val = val;  
 list\_iter->list->current\_size++;  
 int new\_index = list\_iter->list->last\_empty;  
 list\_iter->list->last\_empty = list\_iter->list->buffer[new\_index].next;  
 if (list\_iter->prev == -1) {  
 new\_elem.next = list\_iter->list->head;  
 new\_elem.prev = -1;  
 if (list\_iter->list->head != -1) {  
 list\_iter->list->buffer[list\_iter->list->head].prev = new\_index;  
 }  
 list\_iter->list->head = new\_index;  
 list\_iter->prev = new\_index;  
  
 if (list\_iter->list->tail == -1) {  
 list\_iter->list->tail = new\_index;  
 }  
 } else {  
 new\_elem.next = list\_iter->list->buffer[list\_iter->prev].next;  
 new\_elem.prev = list\_iter->prev;  
 if (list\_iter->list->buffer[list\_iter->prev].next != -1) {  
 list\_iter->list->buffer[list\_iter->list->buffer[list\_iter->prev].next].prev = new\_index;  
 }  
 list\_iter->list->buffer[list\_iter->prev].next = new\_index;  
 list\_iter->prev = new\_index;  
  
 if (new\_elem.next == -1) {  
 list\_iter->list->tail = new\_index;  
 }  
 }  
 list\_iter->list->buffer[new\_index] = new\_elem;  
}  
  
  
void print\_list(bdBarList\_handle list) {  
 if (list->current\_size == 0) {  
 printf("Список пустой\n");  
 return;  
 }  
 for (bdBarList\_iter list\_iter = bdBarList\_iter\_begin(list); !bdBarList\_iter\_equal(list\_iter, bdBarList\_iter\_end(list)); list\_iter = bdBarList\_iter\_next(list\_iter)) {  
 printf("%lf ", bdBarList\_iter\_val(list\_iter));  
 }  
 printf("\n");  
}  
  
bdBarList\_iter get\_n\_iter(int n, bdBarList\_handle list) {  
 bdBarList\_iter iter = bdBarList\_iter\_begin(list);  
 for (int i = 1; i < n; i++) {  
 iter = bdBarList\_iter\_next(iter);  
 }  
 return iter;  
}  
  
bool grow\_buff(bdBarList\_handle list) {  
 int old\_capacity = list->capacity;  
 int new\_capacity = old\_capacity \* 2;  
 bdBarList\_elem\* new\_buff = realloc(list->buffer, new\_capacity \* sizeof(bdBarList\_elem));  
 if (new\_buff != NULL) {  
 list->buffer = new\_buff;  
 list->last\_empty = old\_capacity;  
 for (int i = old\_capacity; i < new\_capacity - 1; i++) {  
 list->buffer[i].next = i + 1;  
 list->buffer[i].prev = i - 1;  
 }  
 list->capacity = new\_capacity;  
 list->buffer[new\_capacity - 1].next = -1;  
 return true;  
 }  
 return false;  
}  
  
void shrink\_buff(bdBarList\_handle list) {  
 bdBarList\_handle tmp = bdBarList\_init(list->current\_size);  
 while (!bdBarList\_is\_empty(list)) {  
 bdBarList\_iter t1 = bdBarList\_iter\_begin(list);  
 double val = bdBarList\_iter\_val(t1);  
 bdBarList\_remove(&t1);  
 bdBarList\_iter t2 = bdBarList\_iter\_end(&tmp);  
 insert\_before(&t2, val);  
 }  
  
 list->buffer = realloc(list->buffer, list->capacity \* 2 / 3 \* sizeof(bdBarList\_elem));  
 list->last\_empty = 0;  
 list->capacity = list->capacity \* 2 / 3;  
 while (!bdBarList\_is\_empty(&tmp)) {  
 bdBarList\_iter t1 = bdBarList\_iter\_begin(&tmp);  
 double val = bdBarList\_iter\_val(t1);  
 bdBarList\_pop(&t1);  
 bdBarList\_iter t2 = bdBarList\_iter\_end(list);  
 insert\_before(&t2, val);  
 }  
}  
  
void bdBarList\_pop(bdBarList\_iter\* list\_iter) {  
 bdBarList\_remove(list\_iter);  
 if (list\_iter->list->current\_size == list\_iter->list->capacity \* 4 / 9) {  
 shrink\_buff(list\_iter->list);  
 }  
}  
  
void bdBarList\_remove(bdBarList\_iter\* list\_iter) {  
 bdBarList\_elem\* to\_del;  
 int tmp = list\_iter->list->last\_empty;  
  
 list\_iter->list->current\_size--;  
 if (list\_iter->prev == -1) {  
 to\_del = &(list\_iter->list->buffer[list\_iter->list->head]);  
 list\_iter->list->last\_empty = list\_iter->list->head;  
 int tmp = list\_iter->list->buffer[list\_iter->list->tail].next;  
 list\_iter->list->buffer[list\_iter->list->tail].next = list\_iter->list->head;  
 if (list\_iter->list->current\_size == 0) {  
 list\_iter->list->head = -1;  
 list\_iter->list->tail = -1;  
 }  
 else {  
 list\_iter->list->head = to\_del->next;  
 }  
 to\_del->next = tmp;  
 }  
 else {  
 to\_del = &(list\_iter->list->buffer[list\_iter->list->buffer[list\_iter->prev].next]);  
  
 if (list\_iter->list->current\_size == 0) {  
 list\_iter->list->head = -1;  
 list\_iter->list->tail = -1;  
 }  
 int tmp = list\_iter->list->buffer[list\_iter->prev].next;  
 list\_iter->list->last\_empty = list\_iter->list->buffer[list\_iter->prev].next;  
 list\_iter->list->buffer[list\_iter->prev].next = to\_del->next;  
 to\_del->next = list\_iter->list->buffer[list\_iter->list->tail].next;  
 list\_iter->list->buffer[list\_iter->list->tail].next = tmp;  
 if (to\_del->next == list\_iter->list->tail) {  
 list\_iter->list->tail = list\_iter->prev;  
 }  
 }  
}  
  
void bdBarList\_destroy(bdBarList\_handle list) {  
 free(list->buffer);  
 free(list);  
}  
  
void del\_every\_kth(bdBarList\_handle list, int k) {  
 if (list->current\_size < k) {  
 printf("Слишком мало элементов списка\n");  
 return;  
 }  
 int current\_index\_to\_del = k;  
 while (current\_index\_to\_del <= list->current\_size) {  
 removeEl(list, current\_index\_to\_del);  
 current\_index\_to\_del += k - 1;  
 }  
}  
  
void insert(bdBarList\_handle list, int pos, double val) {  
 if(pos > list->current\_size + 1) {  
 printf("Слишком большой индекс\n");  
 return;  
 }  
 bdBarList\_iter temp = get\_n\_iter(pos, list);  
 insert\_before(&temp, val);  
}  
  
void removeEl(bdBarList\_handle list, int pos) {  
 if (pos > list->current\_size || list->current\_size == 0) {  
 printf("Слишком большой индекс\n");  
 return;  
 }  
 bdBarList\_iter temp = get\_n\_iter(pos, list);  
 bdBarList\_pop(&temp);  
}  
  
void open\_menu() {  
 printf("h - открыть меню.\n");  
 printf("1 [value, position] - вставить value перед позицией position.\n");  
 printf("2 [position] - удалить элемент на позиции position.\n");  
 printf("3 - распечатать список.\n");  
 printf("4 [k] - Удалить каждый k-й элемент\n");  
}

#include "bidirectional-with-barrier-element.h"  
  
int main(void) {  
 printf("Программа начинает работу\n");  
 bdBarList\_handle l = bdBarList\_init(100);  
 char command;  
 open\_menu();  
 while (scanf(" %c", &command) != EOF) {  
 switch (command) {  
 case 'h': {  
 open\_menu();  
 break;  
 }  
 case '1': {  
 double val;  
 int pos;  
 scanf("%lf %d", &val, &pos);  
 insert(l, pos, val);  
 break;  
 }  
 case '2': {  
 int pos;  
 scanf("%d", &pos);  
 removeEl(l, pos);  
 break;  
 }  
 case '3': {  
 print\_list(l);  
 break;  
 }  
 case '4': {  
 int k;  
 scanf("%d", &k);  
 del\_every\_kth(l, k);  
 break;  
 }  
 default:  
 printf("Ошибка: Неверный выбор меню. Попробуйте еще раз.\n");  
 }  
 }  
}

C:\Users\Дима\CLionProjects\C\kp8\cmake-build-debug\kp8.exe

Программа начинает работу

h - открыть меню.

1 [value, position] - вставить value перед позицией position.

2 [position] - удалить элемент на позиции position.

3 - распечатать список.

4 [k] - Удалить каждый k-й элемент

1 1 1

1 2 2

1 3 3

3

1.000000 2.000000 3.000000

1 4 4

1 5 5

1 6 6

1 7 7

1 8 8

1 9 9

3

1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000

h

h - открыть меню.

1 [value, position] - вставить value перед позицией position.

2 [position] - удалить элемент на позиции position.

3 - распечатать список.

4 [k] - Удалить каждый k-й элемент

2 5

3

1.000000 2.000000 3.000000 4.000000 6.000000 7.000000 8.000000 9.000000

4 2

3

1.000000 3.000000 6.000000 8.000000

4 1

3

Список пустой

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

1. **Замечания автора** по существу работы



1. **Выводы**

В данной работе была разработана и отлажена программа на языке Си для обработки линейного двунаправленного списка с барьерным элементом, используя итераторы. Программа успешно выполняет заданные действия: печать списка, вставка нового элемента, удаление элемента, подсчет длины списка, а также удаление каждого k-го элемента списка.

Подпись студента