МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа № 14

по дисциплине: Основы программирования

тема: «Реализация структуры
данных «Вектор»

Выполнил: ст.группы
Игнатьев Артур Олегович

Проверил:
Преподаватель Притчин Иван Сергеевич
Преподаватель Черников Сергей Викторович

Белгород 2023г.

Лабораторная работа «Реализация структуры данных «Вектор»

Цель работы: усовершенствование навыков в создании библиотек, получение навыков работы с системой контроля версий git.

Содержание отчета:

Тема лабораторной работы.

- Цель лабораторной работы.
- Ссылка на открытый репозиторий с решением.
- Исходный код файлов:

  – vector.h / vector.c

  – vectorVoid.h / vectorVoid.c

  – main.
- Результат выполнения команд
- Выводы по работе.

Решение заданий:

Репозиторий с решениями:

https://github.com/NTK-Hub/Labs/tree/master/libs/data_structures/vector

vector.h:

```c
#include <malloc.h>
#include <stdint.h>
#include <assert.h>
#include <memory.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

typedef struct vector {
    int *data; // указатель на элементы вектора
    size_t size; // размер вектора
    size_t capacity; // вместимость вектора
} vector;

//возвращает структуру-дескриптор вектор из n значений
vector createVector(size_t n);

//изменяет количество памяти, выделенное под хранение элементов вектора
void reserve(vector *v, size_t newCapacity);

//удаляет элементы из контейнера, но не освобождает выделенную память
void clear(vector *v);

//освобождает память, выделенную под неиспользуемые элементы
void shrinkToFit(vector *v);

//освобождает память, выделенную вектору
void deleteVector(vector *v);

//проверяет является ли вектор пустым
bool isEmpty(vector *v);

//проверяет является ли вектор полным
bool isFull(vector *v);

//возвращает i-ый элемент вектора v
int getVectorValue(vector *v, size_t i);

//добавляет элемент x в конец вектора v
void pushBack(vector *v, int x);
```

```
//удаляет последний элемент из вектора
void popBack(vector *v);

//возвращает указатель на index-ый элемент вектора
int *atVector(vector *v, size_t index);

// возвращает указатель на последний элемент вектора
int *back(vector *v);

//возвращает указатель на первый элемент вектора
int *front(vector *v);
```

vector.c:

```
#include <stdio.h>
#include "vector.h"

vector createVector(size_t n) {
    int *memory = malloc(sizeof(int) * n);
    if (memory == NULL) {
        fprintf(stderr, "bad alloc ");
        exit(1);
    } else
        return (vector) {memory, 0, n};
}

void reserve(vector *v, size_t newCapacity) {
    v->data = (int *) realloc(v->data, sizeof(int) * newCapac-
ity);

    if (newCapacity < v->size)
        v->size = newCapacity;

    v->capacity = newCapacity;

    if (newCapacity == 0)
        return;

    if (v->data == NULL) {
        fprintf(stderr, "bad alloc ");
        exit(1);
    }
}

void clear(vector *v) {
    v->size = 0;
}

void shrinkToFit(vector *v) {
    reserve(v, v->size);
}
```

```
void deleteVector(vector *v) {
    free(v->data);
}

bool isEmpty(vector *v) {
    return v->size == 0;
}

bool isFull(vector *v) {
    return v->size == v->capacity;
}

int getVectorValue(vector *v, size_t i) {
    return v->data[i];
}

void pushBack(vector *v, int x) {
    if (isFull(v) && isEmpty(v))
        reserve(v, 1);
    else if (isFull(v))
        reserve(v, 2 * v->capacity);

    v->data[v->size] = x;
    v->size++;
}

void popBack(vector *v) {
    if (isEmpty(v)) {
        fprintf(stderr, "is empty");
        exit(1);
    } else
        v->size--;
}

int *atVector(vector *v, size_t index) {
    if (index > v->capacity)
        fprintf(stderr, "IndexError: a[%zu] is not exists", in-
dex);
    else if (v->capacity == 0 && index < 0) {
        fprintf(stderr, "bad alloc ");
        exit(1);
    }
    return (int *) &v->data[index];
}

int *back(vector *v) {
    return atVector(v, v->size - 1);
}

int *front(vector *v) {
    return v->data;
}
```

vectorVoid.h:

```
#include <limits.h>
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <stdbool.h>
#include <memory.h>

typedef struct vectorVoid {
    void *data; // указатель на нулевой элемент вектора
    size_t size; // размер вектора
    size_t capacity; // вместимость вектора
    size_t baseTypeSize; // размер базового типа:
    // например, если вектор хранит int -
    // то поле baseTypeSize = sizeof(int)
    // если вектор хранит float -
    // то поле baseTypeSize = sizeof(float)
} vectorVoid;

//возвращает структуру-дескриптор вектор базового типа из n зна-
чений
vectorVoid createVectorV(size_t n, size_t baseTypeSize);

//изменяет количество памяти, выделенное под хранение элементов
вектора
void reserveV(vectorVoid *v, size_t newCapacity);

//освобождает память, выделенную под неиспользуемые элементы
void shrinkToFitV(vectorVoid *v);

//удаляет элементы из контейнера, но не освобождает выделенную
память
void clearV(vectorVoid *v);

//освобождает память, выделенную вектору
void deleteVectorV(vectorVoid *v);

//проверяет является ли вектор пустым
bool isEmptyV(vectorVoid *v);

//проверяет является ли вектор полным
bool isFullV(vectorVoid *v);

//записывает по адресу destination index-ый элемент вектора v
void getVectorValueV(vectorVoid *v, size_t index, void *destina-
tion);

//записывает на index-ый элемент вектора v значение, расположен-
ное по
//адресу source
void setVectorValueV(vectorVoid *v, size_t index, void *source);
```

```
//удаляет последний элемент из вектора
void popBackV(vectorVoid *v);

//добавляет элемент x в конец вектора v
void pushBackV(vectorVoid *v, void *source);
```

vectorVoid.c:

```
#include "vectorVoid.h"

vectorVoid createVectorV(size_t n, size_t baseTypeSize) {
    int *memory = malloc(baseTypeSize * n);
    if (memory == NULL) {
        fprintf(stderr, "bad alloc ");
        exit(1);
    } else
        return (vectorVoid) {memory, 0, n, baseTypeSize};
}

void reserveV(vectorVoid *v, size_t newCapacity) {
    v->data = (int *) realloc(v->data, v->baseTypeSize * new-
Capacity);

    if (newCapacity < v->size)
        v->size = newCapacity;

    v->capacity = newCapacity;

    if (newCapacity == 0)
        return;

    if (v->data == NULL) {
        fprintf(stderr, "bad alloc ");
        exit(1);
    }
}

void shrinkToFitV(vectorVoid *v) {
    reserveV(v, v->size);
}

void clearV(vectorVoid *v) {
    v->size = 0;
}

void deleteVectorV(vectorVoid *v) {
    free(v->data);
}

bool isEmptyV(vectorVoid *v) {
    return v->size == 0;
}
```

```
bool isFullV(vectorVoid *v) {
    return v->size == v->capacity;
}

void getVectorValueV(vectorVoid *v, size_t index, void *destina-
tion) {
    char *source = (char *) v->data + index * v->baseTypeSize;
    memcpy(destination, source, v->baseTypeSize);
}

void setVectorValueV(vectorVoid *v, size_t index, void *source)
{
    char *destination = (char *) v->data + index * v->ba-
seTypeSize;
    memcpy(destination, source, v->baseTypeSize);
}

void popBackV(vectorVoid *v) {
    if (isEmptyV(v)) {
        fprintf(stderr, "is empty");
        exit(1);
    } else
        v->size--;
}

void pushBackV(vectorVoid *v, void *source) {
    if (isFullV(v) && isEmptyV(v))
        reserveV(v, 1);
    else if (isFullV(v))
        reserveV(v, 2 * v->capacity);

    setVectorValueV(v, v->size, source);
    v->size++;
}
```

test_vector.h:

```
void test_createVector();

void test_createZeroVector();

void test_reserve1();

void test_reserve2();

void test_reserve();

void test_shrinkToFit();

void test_isEmpty();

void test_isFull();
```

```
void test_getVectorValue1();

void test_getVectorValue2();

void test_getVectorValue3();

void test_getVectorValue();

void test_pushBack_emptyVector();

void test_pushBack_fullVector();

void test_popBack_notEmptyVector();

void test_atVector_notEmptyVector();

void test_atVector_requestToLastElement1();

void test_atVector_requestToLastElement2();

void test_atVector_requestToLastElement();

void test_back_oneElementInVector();

void test_front_oneElementInVector();

void test_back();

void test_front();

void testVector();
```

test_vector.c:

```
#include <assert.h>
#include "vector.h"
#include "test_vector.h"

void test_createVector() {
    vector v = createVector(7);

    assert(v.size == 0);
    assert(v.capacity == 7);

    deleteVector(&v);
}
```

```
void test_createZeroVector() {
    vector v = createVector(0);

    assert(v.size == 0);
    assert(v.capacity == 0);

    deleteVector(&v);
}

void test_reserve1() {
    vector v = createVector(7);
    v.size = 5;
    reserve(&v, 6);

    assert(v.size == 5);
    assert(v.capacity == 6);

    deleteVector(&v);
}

void test_reserve2() {
    vector v = createVector(7);
    v.size = 5;
    reserve(&v, 4);

    assert(v.size == 4);
    assert(v.capacity == 4);

    deleteVector(&v);
}

void test_reserve() {
    test_reserve1();
    test_reserve2();
}

void test_shrinkToFit() {
    vector v = createVector(4);
    v.size = 2;
    shrinkToFit(&v);

    assert(v.size == v.capacity);

    deleteVector(&v);
}
```

```
void test_isEmpty() {
    vector v = createVector(4);

    assert(isEmpty(&v));

    deleteVector(&v);
}

void test_isFull() {
    vector v = createVector(4);
    v.size = 4;

    assert(isFull(&v));

    deleteVector(&v);
}

void test_getVectorValue1() {
    vector v = createVector(4);
    v.data[2] = 45;

    assert(getVectorValue(&v, 2) == v.data[2]);

    deleteVector(&v);
}

void test_getVectorValue2() {
    vector v = createVector(10);
    v.data[10] = 12;

    assert(getVectorValue(&v, 10) == v.data[10]);

    deleteVector(&v);
}

void test_getVectorValue3() {
    vector v = createVector(5);
    *v.data = 1;

    assert(getVectorValue(&v, 0) == *v.data);

    deleteVector(&v);
}

void test_getVectorValue() {
    test_getVectorValue1();
    test_getVectorValue2();
    test_getVectorValue3();
}
```

```
void test_pushBack_emptyVector() {
    vector v = createVector(10);
    pushBack(&v, 15);

    assert(*v.data == 15);
    assert(v.size == 1);
    assert(v.capacity == 10);

    deleteVector(&v);
}

void test_pushBack_fullVector() {
    vector v = createVector(4);
    v.size = 4;
    pushBack(&v, 1);

    assert(v.data[4] == 1);
    assert(v.size == 5);
    assert(v.capacity == 8);

    deleteVector(&v);
}

void test_popBack_notEmptyVector() {
    vector v = createVector(0);
    pushBack(&v, 10);

    assert (v.size == 1);
    popBack(&v);

    assert (v.size == 0);
//    printf("%zu", v.capacity);
    assert (v.capacity == 1);

    deleteVector(&v);
}

void test_atVector_notEmptyVector() {
    vector v = createVector(5);
    v.size = 3;

    assert(atVector(&v, 2) == &v.data[2]);

    deleteVector(&v);
}
```

```
void test_atVector_requestToLastElement1() {
    vector v = createVector(5);

    assert(atVector(&v, 5) == &v.data[5]);

    deleteVector(&v);
}

void test_atVector_requestToLastElement2() {
    vector v = createVector(5);

    assert(atVector(&v, 0) == v.data);

    deleteVector(&v);
}

void test_atVector_requestToLastElement() {
    test_atVector_requestToLastElement1();
    test_atVector_requestToLastElement2();
}

void test_back_oneElementInVector() {
    vector v = createVector(1);
    v.size = 1;

    assert(back(&v) == v.data);

    deleteVector(&v);
}

void test_front_oneElementInVector() {
    vector v = createVector(1);

    assert(front(&v) == v.data);

    deleteVector(&v);
}

void test_back() {
    vector v = createVector(7);
    v.size = 6;

    assert(back(&v) == &v.data[5]);

    deleteVector(&v);
}
```

```
void test_front() {
    vector v = createVector(6);

    v.size = 5;

    assert(front(&v) == v.data);

    deleteVector(&v);
}

void testVector() {
    test_createVector();
    test_createZeroVector();
    test_reserve();
    test_shrinkToFit();
    test_isEmpty();
    test_isFull();
    test_getVectorValue();
    test_pushBack_emptyVector();
    test_pushBack_fullVector();
    test_popBack_notEmptyVector();
    test_atVector_notEmptyVector();
    test_atVector_requestToLastElement();
    test_back_oneElementInVector();
    test_front_oneElementInVector();
    test_back();
    test_front();
}
```

Вывод: на этой лабораторной работе я усовершенствовал навыки в создании библиотек, получил навыки работы с системой контроля версий git.