Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра вычислительных систем

ОТЧЕТ
по практической работе 2

по дисциплине «**Программирование**»

Выполнил:
студент гр. ИС-241
«30» марта 2023 г. _____ /Бондаренко А.А./

Проверил:
ст. преп. кафедры ВС
«__» марта 2023 г. _____ /Фульман В.О./

Оценка «_____»

Новосибирск 2023

ОГЛАВЛЕНИЕ

ЗАДАНИЕ

Реализовать тип данных «Динамический массив целых чисел» — IntVector и основные функции для работы с ним. Разработать тестовое приложение для демонстрации реализованных функций.

Рекомендуемая структура проекта:

```
.
|-- Makefile
`-- src
    |-- IntVector.c
    |-- IntVector.h
    `-- main.c
```

Требования к работе:

1. Должны обрабатываться ошибки выделения памяти.
2. Не должно быть утечек памяти.
3. При тестировании приложения необходимо проверить граничные случаи. Например, работоспособность операции добавления элемента после уменьшения размера массива до нуля.

ВЫПОЛНЕНИЕ РАБОТЫ

IntVector.h

В заголовочном файле IntVector.h содержится объявление основных функций для работы с типом данных «Динамический массив целых чисел» IntVector и реализован он сам в виде структуры с тремя полями:

1. Размер - количество заполненных элементов
2. Ёмкость - количество элементов, для которых зарезервирована память
3. Сам массив данных - указатель на участок памяти, в котором хранятся элементы

```
1 typedef struct {
2     size_t size; // размер
3     size_t capacity; // ёмкость
4     int* data; // массив
5 } IntVector;
```

Исходный код заголовочного файла приведён в приложении 1.

IntVector.c

В IntVector.c реализованы функции, объявленные в IntVector.h:

IntVector* int_vector_new(size_t initial_capacity)

Функция создаёт новый вектор нулевого размера.

Принимает на вход ёмкость вектора, после чего выделяет память под структуру с помощью malloc. Если память выделить не удалось, функция возвращает NULL. Дальше функция также выделяет память для массива с ёмкостью, полученной в качестве параметра, и происходит такая же проверка успешности выделения памяти. После, если оба выделения памяти прошли как надо, ёмкости массива присваивается полученное значение ёмкости, а размеру массива – ноль.

Функция возвращает указатель на инициализированный вектор.

```
 1 IntVector* int_vector_new(size_t initial_capacity)
 2 {
 3     IntVector* vector = malloc(sizeof(IntVector));
 4 
 5     if (!vector) {
 6         free(vector);
 7         return NULL;
 8     }
 9 
10     vector->data = malloc(initial_capacity * sizeof(int));
11 
12     if (!(vector->data)) {
13         free(vector);
14         return NULL;
15     }
16 
17     vector->capacity = initial_capacity;
18     vector->size = 0;
19 
20     return vector;
21 }
```

IntVector* int_vector_copy(const IntVector* v)

Функция создаёт указатель на копию вектора.

Принимает на вход уже существующий вектор. Работает по большому счёту аналогично предыдущей функции: выделяет память под копию всей структуры и возвращает NULL, если сделать это не удалось. После с такой же последующей проверкой выделяется память под массив. В случае успеха в соответствующие поля вектора-копии переносятся значения размера и ёмкости, затем с помощью цикла for копируются все значения элементов массива.

Функция возвращает указатель на структуру-копию.

```
IntVector* int_vector_copy(const IntVector* v)
{
    IntVector* vector_copy = malloc(sizeof(IntVector));

    if (!vector_copy) {
        free(vector_copy);
        return NULL;
    }

    vector_copy->data = malloc(v->capacity * sizeof(int));

    if (!(vector_copy->data)) {
        free(vector_copy);
        return NULL;
    }

    vector_copy->capacity = v->capacity;
    vector_copy->size = v->size;

    for (int i = 0; i < vector_copy->size; i++) {
        vector_copy->data[i] = v->data[i];
    }

    return vector_copy;
}
```

void int_vector_free(IntVector* v)

Функция освобождает память, выделенную под вектор.

Принимает на вход уже существующий вектор. Принцип работы очень простой: с помощью стандартной функции free() освобождается сначала память, выделенная под массив, потом память, выделенная под всю структуру.

Функция ничего не возвращает (тип возвращаемого значения void).

```
void int_vector_free(IntVector* v)
{
    free(v->data);
    free(v);
}
```

int int_vector_get_item(const IntVector* v, size_t index)

Функция возвращает элемент под номером index.
Принимает на вход вектор и индекс искомого элемента.
Функция возвращает нужный элемент, обращаясь к нему по индексу.

```
int int_vector_get_item(const IntVector* v, size_t index)
{
    return v->data[index];
}
```

void int_vector_set_item(IntVector* v, size_t index, int item)

Функция присваивает указанное значение элементу под номером index.
Принимает на вход вектор, индекс элемента и новое значение для него. Обращаясь к элементу по индексу, функция изменяет его значение. В случае, если индекс равен размеру массива, функция увеличивает значение размера массива на 1.
Функция ничего не возвращает (тип возвращаемого значения void).

```
void int_vector_set_item(IntVector* v, size_t index, int item)
{
    v->data[index] = item;

    if(index == v->size) {
        v->size = index + 1;
    }
}
```

size_t int_vector_get_size(const IntVector* v)

Функция возвращает размер массива. Принимает на вход вектор.

```
size_t int_vector_get_size(const IntVector* v)
{
    return v->size;
}
```

size_t int_vector_get_capacity(const IntVector* v)

Функция возвращает ёмкость массива. Принимает на вход вектор.

```
size_t int_vector_get_size(const IntVector* v)
{
    return v->capacity;
}
```

int int_vector_push_back(IntVector* v, int item)

Функция добавляет элемент в конец массива, при необходимости увеличивая ёмкость массива.
Принимает на вход вектор и добавляемое значение. Если ёмкости не хватает, чтобы вместить новый элемент (т.е. размер равен ёмкости), то с использованием временного массива expance память перевыделяется функцией realloc, увеличивая ёмкость в два раза.

Если память не удалось выделить, функция возвращает -1 (т.к. тип возвращаемого значения функции - int).

После временный массив с перевыделенной памятью присваивается изначальному, удваивается ёмкость. Затем переданное значение присваивается последнему элементу, а размер массива увеличивается на 1.

```
int int_vector_push_back(IntVector* v, int item)
{
    if (v->size == v->capacity) {
        int* expance = realloc(v->data, v->capacity * sizeof(int) * 2);
        if (!expance) {
            return -1;
        }
        v->data = expance;
        v->capacity = v->capacity * 2;
    }

    v->data[v->size] = item;
    v->size++;

    return 0;
}
```

void int_vector_pop_back(IntVector* v)

Функция удаляет последний элемент из массива, если размер массива больше 0.

Удаление последнего элемента массива происходит за счёт уменьшения размера массива на 1.

Функция ничего не возвращает (тип возвращаемого значения void).

```
void int_vector_pop_back(IntVector* v)
{
    if (v->size > 0) {
        v->size--;
    }
}
```

int int_vector_shrink_to_fit(IntVector* v)

Функция уменьшает ёмкость массива до его размера.

Принимает на вход вектор. При условии, что размер массива не равен нулю, а ёмкость массива больше его размера, происходит перевыделение памяти посредством временного массива. Происходит проверка выделения памяти – в случае провала функция вернёт -1.

После изначальному массиву присваивается временный, значению ёмкости присваивается значение размера, делая ёмкость равной размеру.

Функция возвращает -1, если размер массива равен 0 или в случае, когда ёмкость и так не больше значения размера массива.

```
 1 int int_vector_shrink_to_fit(IntVector* v)
 2 {
 3     if (v->size == 0) {
 4         return -1;
 5     }
 6
 7     if (v->capacity > v->size) {
 8         int* expance = realloc(v->data, v->size * sizeof(int));
 9         if (!expance) {
10             return -1;
11         }
12         v->data = expance;
13         v->capacity = v->size;
14
15         return 0;
16     }
17
18     return -1;
19 }
```

int int_vector_resize(IntVector* v, size_t new_size)

Функция изменяет размер массива.

Принимает на вход вектор и его новый размер. В случае, когда новый размер больше старого, то добавленные элементы заполняются нулями с помощью цикла for и обращения к элементам по индексу в диапазоне от старого значения размера до нового. После значение размера массива заменяется новым.

Если же новый размер меньше старого, то значение размера массива заменяется новым, после чего вызывается функция int_vector_shrink_to_fit для уменьшения ёмкости массива.

Функция возвращает -1, если новое значение размера массива равняется старому или если оно больше ёмкости массива.

```
 1 int int_vector_resize(IntVector* v, size_t new_size)
 2 {
 3     if (new_size == v->size || new_size > v->capacity) {
 4         return -1;
 5     }
 6
 7     if (new_size > v->size) {
 8         for (int i = v->size; i < new_size; i++) {
 9             v->data[i] = 0;
10         }
11         v->size = new_size;
12     }
13
14     if (new_size < v->size) {
15         v->size = new_size;
16         int_vector_shrink_to_fit(v);
17     }
18     return 0;
19 }
```

int int_vector_reserve(IntVector* v, size_t new_capacity)

Функция изменяет ёмкость массива.

Принимает на вход вектор и его новую ёмкость.

С использованием временного массива и функции realloc память под массив перевыделяется с новой ёмкостью, затем при успешном выделении памяти (при ошибке функция возвращает -1) он присваивается изначальному массиву, старая ёмкость заменяется на новую.

Функция возвращает -1, если новое значение ёмкости меньше или равно старому.

```c
int int_vector_reserve(IntVector* v, size_t new_capacity)
{
    if (new_capacity <= v->capacity) {
        return -1;
    }

    int* expance = realloc(v->data, new_capacity * sizeof(int));

    if (!expance) {
        return -1;
    }
    v->data = expance;
    v->capacity = new_capacity;

    return 0;
}
```

test.c

Полный код тестового приложения находиться в приложении 3. В общей сложности главная задача программы – проверка и демонстрация корректной работы всех приведённых в лабораторной работе функций посредством их последовательного вызова.

Вывод тестового приложения test.c выглядит следующим образом:

```
jumkot@Jumkot:~/prog/Lab/laba2$ make run
./app
1
Вызов int_vector_new: создает массив нулевого размера start

2
Вызов int_vector_get_capacity: возвращает ёмкость вектора start
start->capacity = 10

3
Вызов int_vector_set_item: присваивает элементу под номером index = i значение item = i * 5
```

```
4
Вызов int_vector_get_item: возвращает элемент под номером index = i
start->data[0] = 0
start->data[1] = 5
start->data[2] = 10
start->data[3] = 15
start->data[4] = 20
start->data[5] = 25
start->data[6] = 30
start->data[7] = 35
start->data[8] = 40
start->data[9] = 45

5
Вызов int_vector_copy: создает указатель test на копию вектора start
Проверка:
test->data[1] = 5
start->data[1] = 5


6
Вызов int_vector_free: освобождает память, выделенную для вектора test
Проверка:
test->data[0] = 0
start->data[0] = 5

7
Вызов int_vector_pop_back: удаляет последний элемент из массива start->data
Последний элемент до: start->data[9] = 45

Последний элемент после: start->data[8] = 40

8
Вызов int_vector_get_size: возвращает размер вектора start
Размер массива: 9
Ёмкость массива: 10

9
Вызов int_vector_push_back: добавляет элемент в конец массива start->data
Проверка: start->data[9] = 2038

10
Вызов int_vector_reserve: изменяет ёмкость массива start->capacity
Старая ёмкость: 10

Новая ёмкость: 15
```

```
11
Вызов int_vector_resize: изменяет размер массива start->size
start->data[0] = 0
start->data[1] = 5
start->data[2] = 10
start->data[3] = 15
start->data[4] = 20
start->data[5] = 25
start->data[6] = 30
start->data[7] = 35
start->data[8] = 40
start->data[9] = 2038
start->data[10] = 0
start->data[11] = 0
start->data[12] = 0
start->data[13] = 0
start->data[14] = 0

12
Вызов int_vector_shrink_to_fit: уменьшает ёмкость массива start->capacity до его размера start->size
Старая ёмкость: 15, размер: 12

Новая ёмкость: 12, размер: 12

ГРАНИЧНЫЙ СЛУЧАЙ:
1) уменьшение размера массива до нуля
    size = 0
    start->data[0] = 0

2) добавление элемента в конец массива
    size = 1
    start->data[0] = 451

Вызов int_vector_free для освобождения памяти, выделенной для вектора start
```

makefile

Для сборки приложения был написан makefile в соответствии с предложенной структурой:

```
.
|-- Makefile
`-- src
    |-- IntVector.c
    |-- IntVector.h
    `-- main.c
```

То есть программы и заголовочный файл находятся в каталоге src, на который указывается в мейк-файле с использованием маски *.c. Благодаря этому все файлы с расширением .c сначала собираются в объектные файлы без компоновки (опция -c).

После уже объектные файлы компилируются также с использованием маски *.o. Затем, когда исполняемый файл собран и скомпилирован, объектные файлы удаляются.

Также отдельно вынесены опции clean для удаления исполняемого файла и run для его запуска.

Код мейк-файла приведён в приложении 4.

ПРИЛОЖЕНИЕ

Приложение 1

IntVector.h

```h
#include <stdio.h>

typedef struct {
    size_t size; // размер
    size_t capacity; // ёмкость
    int* data; // массив
} IntVector; // Структура IntVector

IntVector* int_vector_new(size_t initial_capacity);

IntVector* int_vector_copy(const IntVector* v);

void int_vector_free(IntVector* v);

int int_vector_get_item(const IntVector* v, size_t index);

void int_vector_set_item(IntVector* v, size_t index, int item);

size_t int_vector_get_size(const IntVector* v);

size_t int_vector_get_capacity(const IntVector* v);

int int_vector_push_back(IntVector* v, int item);

void int_vector_pop_back(IntVector* v);

int int_vector_shrink_to_fit(IntVector* v

int int_vector_resize(IntVector* v, size_t new_size);

int int_vector_reserve(IntVector* v, size_t new_capacity);
```

Приложение 2

IntVector.c

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    size_t size; // размер
    size_t capacity; // ёмкость
    int* data; // массив
} IntVector;

IntVector* int_vector_new(size_t initial_capacity)
{
    IntVector* vector = malloc(sizeof(IntVector));

    if (!vector) {
        free(vector);
        return NULL;
    }

    vector->data = malloc(initial_capacity * sizeof(int));

    if (!(vector->data)) {
        free(vector);
        return NULL;
    }

    vector->capacity = initial_capacity;
    vector->size = 0;

    return vector;
}

IntVector* int_vector_copy(const IntVector* v)
{
    IntVector* vector_copy = malloc(sizeof(IntVector));

    if (!vector_copy) {
        free(vector_copy);
        return NULL;
    }

    vector_copy->data = malloc(v->capacity * sizeof(int));

    if (!(vector_copy->data)) {
        free(vector_copy);
        return NULL;
    }

    vector_copy->capacity = v->capacity;
    vector_copy->size = v->size;

    for (int i = 0; i < vector_copy->size; i++) {
        vector_copy->data[i] = v->data[i];
```

```
    }

    return vector_copy;
}

void int_vector_free(IntVector* v)
{
    free(v->data);
    free(v);
}

int int_vector_get_item(const IntVector* v, size_t index)
{
    return v->data[index];
}

void int_vector_set_item(IntVector* v, size_t index, int item)
{
    v->data[index] = item;

    if(index == v->size) {
        v->size = index + 1;
    }
}

size_t int_vector_get_size(const IntVector* v)
{
    return v->size;
}

size_t int_vector_get_capacity(const IntVector* v)
{
    return v->capacity;
}

int int_vector_push_back(IntVector* v, int item)
{
    if (v->size == v->capacity) {
        int* expance = realloc(v->data, v->capacity * sizeof(int) * 2);
        if (!expance) {
            return -1;
        }
        v->data = expance;
        v->capacity = v->capacity * 2;
    }

    v->data[v->size] = item;
    v->size++;

    return 0;
}

void int_vector_pop_back(IntVector* v)
{
    if (v->size > 0) {
        v->size--;
    }
}
```

```c

int int_vector_shrink_to_fit(IntVector* v)
{
    if (v->size == 0) {
        return -1;
    }

    if (v->capacity > v->size) {
        int* expance = realloc(v->data, v->size * sizeof(int));
        if (!expance) {
            return -1;
        }
        v->data = expance;
        v->capacity = v->size;

        return 0;
    }

    return -1;
}

int int_vector_resize(IntVector* v, size_t new_size)
{
    if (new_size == v->size || new_size > v->capacity) {
        return -1;
    }

    if (new_size > v->size) {
        for (int i = v->size; i < new_size; i++) {
            v->data[i] = 0;
        }
        v->size = new_size;
    }

    if (new_size < v->size) {
        v->size = new_size;
        int_vector_shrink_to_fit(v);
    }
    return 0;
}

int int_vector_reserve(IntVector* v, size_t new_capacity)
{
    if (new_capacity <= v->capacity) {
        return -1;
    }

    int* expance = realloc(v->data, new_capacity * sizeof(int));

    if (!expance) {
        return -1;
    }
    v->data = expance;
    v->capacity = new_capacity;

    return 0;
}
```

Приложение 3

test.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "IntVector.h"
#define BLUE "\e[1;34m"
#define CLOSE "\e[0m"
#define VALUE 10

void is_error(IntVector* v) // Проверяет на ошибку
{
    if (!v) {
        printf("Error of memory\n");
        exit(1);
    }
}

int main()
{
    IntVector *start, *test;

    printf("1\n");
    printf("Вызов %sint_vector_new%s: создает массив нулевого размера
%sstart%s\n", BLUE, CLOSE, BLUE, CLOSE);
    start = int_vector_new(VALUE);
    is_error(start);
    printf("\n");

    printf("2\n");
    printf("Вызов %sint_vector_get_capacity%s: возвращает ёмкость вектора
%sstart%s\n", BLUE, CLOSE, BLUE, CLOSE);
    printf("start->capacity = %ld\n", int_vector_get_capacity(start));
    printf("\n");

    printf("3\n");
    printf("Вызов %sint_vector_set_item%s: присваивает элементу под номером
index = %si%s значение item = %si * 5%s\n", BLUE, CLOSE, BLUE, CLOSE, BLUE,
CLOSE);
    for (int i = 0; i < int_vector_get_capacity(start); i++) {
        int_vector_set_item(start, i, i * 5);
    }
    printf("\n");

    printf("4\n");
    printf("Вызов %sint_vector_get_item%s: возвращает элемент под номером
index = %si%s\n", BLUE, CLOSE, BLUE, CLOSE);
    for (int i = 0; i < int_vector_get_capacity(start); i++) {
        printf("start->data[%d] = %d\n", i, int_vector_get_item(start, i));
    }
    printf("\n");

    printf("5\n");
    printf("Вызов %sint_vector_copy%s: создает указатель test на копию
вектора %sstart%s\n", BLUE, CLOSE, BLUE, CLOSE);
    test = int_vector_copy(start);
```

```
    is_error(test);
    printf("Проверка:\ntest->data[1] = %d\nstart->data[1] = %d\n\n",
int_vector_get_item(test, 1), int_vector_get_item(start, 1));
    printf("\n");

    printf("6\n");
    printf("Вызов %sint_vector_free%s: освобождает память, выделенную для
вектора %stest%s\n", BLUE, CLOSE, BLUE, CLOSE);
    int_vector_free(test);
    printf("Проверка:\ntest->data[0] = %d\nstart->data[0] = %d\n\n",
int_vector_get_item(test, 1), int_vector_get_item(start, 1));

    printf("7\n");
    printf("Вызов %sint_vector_pop_back%s: удаляет последний элемент из
массива %sstart->data%s\n", BLUE, CLOSE, BLUE, CLOSE);
    printf("Последний элемент до: start->data[%ld] = %d\n\n",
int_vector_get_size(start) - 1, start->data[int_vector_get_size(start) -
1]);
    int_vector_pop_back(start);
    printf("Последний элемент после: start->data[%ld] = %d\n\n",
int_vector_get_size(start) - 1, start->data[int_vector_get_size(start) -
1]);

    printf("8\n");
    printf("Вызов %sint_vector_get_size%s: возвращает размер вектора
%sstart%s\n", BLUE, CLOSE, BLUE, CLOSE);
    printf("Размер массива: %ld\nЁмкость массива: %ld\n\n",
int_vector_get_size(start), int_vector_get_capacity(start));

    printf("9\n");
    printf("Вызов %sint_vector_push_back%s: добавляет элемент в конец
массива %sstart->data%s\n", BLUE, CLOSE, BLUE, CLOSE);
    int_vector_push_back(start, 2038);
    printf("Проверка: start->data[%ld] = %d\n\n",
int_vector_get_capacity(start) - 1, int_vector_get_item(start, VALUE - 1));

    printf("10\n");
    printf("Вызов %sint_vector_reserve%s: изменяет ёмкость массива %sstart-
>capacity%s\n", BLUE, CLOSE, BLUE, CLOSE);
    printf("Старая ёмкость: %ld\n\n", int_vector_get_capacity(start));
    int_vector_reserve(start, VALUE + 5);
    printf("Новая ёмкость: %ld\n\n", int_vector_get_capacity(start));
    printf("\n");

    printf("11\n");
    printf("Вызов %sint_vector_resize%s: изменяет размер массива %sstart-
>size%s\n", BLUE, CLOSE, BLUE, CLOSE);
    int_vector_resize(start, VALUE + 2);
    for (int i = 0; i < int_vector_get_capacity(start); i++) {
        printf("start->data[%d] = %d\n", i, int_vector_get_item(start, i));
    }
    printf("\n");

    printf("12\n");
    printf("Вызов %sint_vector_shrink_to_fit%s: уменьшает ёмкость массива
%sstart->capacity%s до его размера %sstart->size%s\n", BLUE, CLOSE, BLUE,
CLOSE, BLUE, CLOSE);

```

```
    printf("Старая ёмкость: %ld, размер: %ld\n\n",
int_vector_get_capacity(start), int_vector_get_size(start));
    int_vector_shrink_to_fit(start);
    printf("Новая ёмкость: %ld, размер: %ld\n\n",
int_vector_get_capacity(start), int_vector_get_size(start));

    printf("ГРАНИЧНЫЙ СЛУЧАЙ:\n1) уменьшение размера массива до нуля\n");
    int_vector_resize(start, 0);
    printf("    size = %ld\n    start->data[0] = %d\n",
int_vector_get_size(start), int_vector_get_item(start, 0));
    printf("\n");
    printf("2) добавление элемента в конец массива\n");
    int_vector_push_back(start, 451);
    printf("    size = %ld\n    start->data[0] = %d\n",
int_vector_get_size(start), int_vector_get_item(start, 0));
    printf("\n");

    printf("Вызов %sint_vector_free%s для освобождения памяти, выделенной
для вектора %sstart%s\n", BLUE, CLOSE, BLUE, CLOSE);
    int_vector_free(start);
}
```

Приложение 4

makefile

```
all: app
app:
        gcc -Wall -Werror -c src/*.c
        gcc -Wall -Werror *.o -o app
        rm *.o
clean:
        rm app
run:
        ./app
```