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
«6» марта 2023 г. _____ /Стрельников.А.М./

Проверил:
Ст. преп. Кафедры ВС
«27» июня 2023 г. _____ /Фульман В.О./

Оценка «_____»

Новосибирск 2023

ОГЛАВЛЕНИЕ

ЗАДАНИЕ

Реализовать тип данных “Динамический массив целых чисел” - и основные функции для работы с ним. Разработать тестовое приложение для демонстрации реализованных функций.

ВЫПОЛНЕНИЕ РАБОТЫ

Для выполнения этой работы необходимо реализовать два кода Intvector.c, main.c и один заголовочный файл IntVector.h в названии.

Пару слов про динамический массив.
Динамический массив - это массив, размер которого определяется во время выполнения программы, в отличие от статического массива, размер которого определяется во время компиляции и остается неизменным во время выполнения.
В языке Си для работы с динамическими массивами используются функции malloc() и free(). Функция malloc() выделяет блок памяти определенного размера, который может использоваться для хранения элементов массива. Функция free() освобождает выделенную ранее память.

Разберем функции для работы с динамическими массивами подробнее.

```
IntVector *int_vector_new(size_t initial_capacity)
```

Создает массив нулевого размера.

Эта функция создает новый динамический массив Intvector с заданной емкостью “capacity”. Возвращает указатель на созданный массив. Если выделить память не удалось, функция возвращает NULL. Заметим, что при создании нового массива все элементы заполняются нулями.

```
IntVector *int_vector_copy(const IntVector *v)
```

Результат: Указатель на копию вектора `v`. `NULL`, если не удалось выделить память.

Данная функция создает новый динамический массив IntVector, который является копией переданного массива “v”. Возвращает указатель на созданный массив. Если не удалось, функция возвращает NULL.

```
void int_vector_free(IntVector *v)
```

Освобождает память, выделенную для вектора `v`.

Следующая функция освобождает память, занятую динамическим массивом “v”.

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

Результат: элемент под номером `index`. В случае выхода за границы массива поведение не определено.

Эта функция возвращает элемент массива “v” по заданному “index”. Если индекс находится вне диапазона массива, функция возвращает код ошибки “GETTING_INDEX_OUT_OF_ARRAY_ERROR”

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

Присваивает элементу под номером `index` значение `item`. В случае выхода за границы массива поведение не определено.

Функция устанавливает значение элемента “v” по заданному индексу “index” равным “item”. Если индекс находится вне диапазона массива, функция возвращает код ошибки “SETTING_VALUE_TO_ELEMENT_OUT_OF_ARRAY”

```
size_t int_vector_get_size(const IntVector *v)
```

Результат: размер вектора.

Функция “int_vector_get_size” возвращает текущее количество элементов в массиве “v”

```
size_t int_vector_get_capacity(const IntVector *v)
```

Результат: емкость вектора.

Эта функция возвращает текущую емкость массива “v”

```
int int_vector_push_back(IntVector *v, int item)
```

Добавляет элемент в конец массива. При необходимости увеличивает емкость массива.
Для простоты в качестве коэффициента роста можно использовать 2.

Функция добавляет новый элемент со значением “item” в конец массива “v”. Если при этом не хватает места в массиве, функция увеличивает его емкость. Если удалось добавить элемент, функция возвращает 0. Если произошла ошибка выделения памяти, то функция возвращает код ошибки “ARRAY_MEMORY_ALLOCATION_ERROR”

```
void int_vector_pop_back(IntVector *v)
```

Удаляет последний элемент из массива. Нет эффекта, если размер массива равен 0.

Функция удаляет последний элемент из массива “v”. Если массив пуст, функция ничего не делает

```
int int_vector_shrink_to_fit(IntVector *v)
```

Уменьшает емкость массива до его размера.

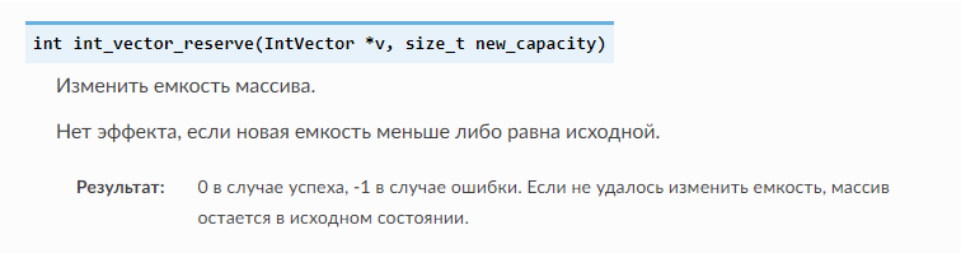
Результат: 0 в случае успешного изменения емкости, -1 в случае ошибки.

Данная функция уменьшает емкость массива “v” до количества его элементов. Если при этом не удалось перевыделить память для массива, то функция возвращает код ошибки “ ARRAY_MEMORE_ALLOCATION_ERROR”

```
int int_vector_resize(IntVector *v, size_t new_size)
```

Изменяет размер массива.

Эта функция изменяет количество элементов в массиве до заданного значения “new_size”. Если новое количество элементов больше текущего, новые элементы будут инициализированы нулями,если меньше - элементы, выходящие за пределы нового размера массива, будут удалены. Если удалось выделить память для нового с заданным количеством элементов, старые значения элементов копируются в новый массив, старый массив удаляется и его указатель заменяется на указатель на новый массив. Функция возвращает 0 в случае успешного выполнения и -2 в случае неудачи (невозможности выделить память для нового массива)



Последняя в нашем списке функция увеличивает емкость массива до заданного значения “new_capacity”. Если новая емкость меньше или равна текущей емкости, функция не делает ничего и возвращает 0. Если удалось выделить память для нового массива с заданной емкостью, старые значения элементов копируются в новый массив, старый массив удаляется и его указатель заменяется на указатель нового массива. Функция возвращает 0 в случае успешного выполнения и -2 в случае неудачи(невозможности выделить память для нового массива).

ПРИЛОЖЕНИЕ

IntVector.h

```c
#include <stdio.h> // подключаем библиотеку для ввода и вывода данных
#include <stdlib.h> // подключаем библиотеку для работы с памятью
#include <stdbool.h> // подключаем библиотеку с логическими значениями

// макросы для определения констант и кодов ошибок
#define INT_SIZE sizeof(int) // размер целого числа в байтах
#define STRUCT_MEMORY_ALLOCATION_ERROR -1 // ошибка при выделении памяти для структуры
```

```c
#define ARRAY_MEMORY_ALLOCATION_ERROR -2 // ошибка при выделении памяти для массива в структуре
#define GETTING_INDEX_OUT_OF_ARRAY_ERROR -3 // ошибка при получении элемента по индексу вне диапазона массива
#define SETTING_VALUE_TO_ELEMENT_OUT_OF_ARRAY -4 // ошибка при установке значения элемента вне диапазона массива

// Определение структуры данных IntVector, представляющей динамический массив целых чисел
typedef struct
{
    int* data; // указатель на массив целых чисел
    int capacity; // емкость массива (количество выделенных ячеек памяти)
    int size; // количество элементов в массиве
} IntVector;

// объявление функций
void print_error(int code_error);
IntVector* int_vector_new(size_t);
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);
void int_vector_pop_back(IntVector *v);
int int_vector_push_back(IntVector *v, int item);
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);
```

IntVector.c

```c
#include "intVector.h" // Подключение заголовочного файла intVector.h

void print_error(int code_error) // Объявление функции print_error с аргументом code_error
{
    switch (code_error) // Начало оператора switch, оцениваемого по аргументу code_error
    {
        case STRUCT_MEMORY_ALLOCATION_ERROR: // Если code_error равен константе STRUCT_ALLOCATION_ERROR
        printf("%s\n", "error: failed to alllocate memory for the vector"); // Выводим сообщение об ошибке
        break; // Завершаем выполнение оператора switch

        case ARRAY_MEMORY_ALLOCATION_ERROR: // Если code_error равен константе ARRAY_MEMORY_ALLOCATION_ERROR
        printf("%s\n", "error: failed to allocate memory for the array"); // Выводим сообщение об ошибке
        break; // Завершаем выполнение оператора switch

        case GETTING_INDEX_OUT_OF_ARRAY_ERROR: // Если code_error равен константе GETTING_INDEX_OUT_OF_ARRAY_ERROR
        printf("%s\n", "error: getting ellement out of array"); //выводим сообщение об ошибке
        break; // Завершаем выполнение оператора switch

        case SETTING_VALUE_TO_ELEMENT_OUT_OF_ARRAY: // Если code_error равен константе SETTING_VALUE_TO_ELEMENT_OUT_OF_ARRAY
        printf("%s\n", "error: setting value to element out of array"); // Выводим сообщеение об ошибке
        break; // Завершаем выполнение оператора switch

        default: // Если code_error не равен ни одной из перечисленных констант
        printf("unknown error"); // Выводим сообщение об ошибке
        break; // Завершаеем работу оператора switch
    }
}
IntVector* int_vector_new(size_t initial_capacity)
{
    IntVector *vector = malloc(sizeof(IntVector));
    // Выделяем память для структуры IntVector или начальная емкость не положительна, тогда выводим ошибку
    if (vector == NULL || initial_capacity <= 0)
        print_error(STRUCT_MEMORY_ALLOCATION_ERROR);

    // Иначе, инициализируем структуру, выделяем память для массива данных и проверяем успешность выделения
    else
    {
        vector->capacity = initial_capacity;
        vector->data = malloc(INT_SIZE * initial_capacity);

        // Если не удалось выделить память для массива дданных, освобожждаем
```

```
ранее выделенную память и выводим ошибку
        if (vector->data == NULL)
        {
            free(vector);
            print_error(ARRAY_MEMORY_ALLOCATION_ERROR);
            vector = NULL;
        }
    }
    // Возвращаем указатель на структуру IntVector
    return vector;
}
// Созданиее нового вектора и копирование данных из исходного вектора в него
IntVector *int_vector_copy(const IntVector *v)
{
    // Выделяем память под новый вектор
    IntVector *vector = int_vector_new(v->capacity);

    // Проверяем, была ли выделена память под новый вектор
    if (vector != NULL)
    {
        // Копируем данные из исходного вектора в новый
        vector->size = v->size;
        vector->capacity = v->capacity;
        for (int i = 0; i < vector->size; i++)
        {
            // Копируем элементы из исходного вектора в новый
            (vector->data)[i] = (v->data)[i];
        }
    }
    // Возвращаем указатель на новый вектор
    return vector;
}

// Функция, которая освобождает память, выделенную под структуру IntVector и
массив data
void int_vector_free(IntVector *v)
{
    if (v != NULL && v->data != NULL) // Проверяем, что указатель на структуру
IntVector и на массив data не являются NULL
    free(v->data); // Освобождаем память, выдделенную под массив data
    if (v != NULL) // Проверяем, что указатель на структуру IntVector не
является NULL
    free(v); // Освобождаем память, выделенную подд структуру IntVector
}

int int_vector_get_item(const IntVector *v, size_t index)
{
    int item; // Объявляем переменную item для хранения значения элемента
массива
    if (index >= 0 && index < v->size) // Проверяем, что индекс находится в
пределах размера массива
    item = *(v->data + index); // Присваиваем значение элемента массива по
указанному индексу переменной item
    else // Если индекс за пределами массива, то вызываем функцию вывода
сообщения об ошибке
    print_error(GETTING_INDEX_OUT_OF_ARRAY_ERROR);
    return item; // возвращаем значение элемента массива
```

```
}

//Функция для установки значения элемента вектора по индексу
void int_vector_set_item(IntVector *v, size_t index, int item)
{
    if(index >= 0 && index < v -> size) //если индекс находится в допустимом
диапазоне
        *(v->data + index) = item; //устанавливаем значение элемента вектора

    else //если индекс находится вне допустимого диапазона
        print_error(SETTING_VALUE_TO_ELEMENT_OUT_OF_ARRAY); //выводим
сообщение об ошибкее
}

size_t int_vector_get_size(const IntVector *v)
{
    return v->size; //возвращает размер вектора, который хранитсяв поле size
структуры Intvector
}

//функция возвращает теекущую емкость вектора
size_t int_vector_get_capacity(const IntVector *v)
{
    return v ->  capacity; // возвращает текущую емкость вектора
}

//функция уменьшает размер вектора на 1, удаляя последний элемент. Если
веектор пустой, то функция ничего не делает
void int_vector_pop_back(IntVector *v)
{
    if (v -> size != 0) //если вектор не пустой
    {
        --(v -> size); //уменьшаем размер вектора на 1
        v -> data[v -> size] = 0; //удаляем последний элемент
    }
}
//функция добавляет новый элемент в конец вектора. Если вектор заполнен, его
вместимость увеличивается в два раза
int int_vector_push_back(IntVector *v, int item)
{
    if(v -> size >= v -> capacity) //если размер вектора ббольше или равен его
емкости, значит нужно увеличить его емкость
    {
        int_vector_reserve(v, v -> capacity * 2); //вызываем функцию, которая
увеличит емкость вектора в два раза
    }
    v -> data[v -> size] = item; //записываем значение item в ячейку массива,
соответсвующую размеру вектора
    v -> size += 1; //увеличиваем рамзер вектора на 1
    return 0;
}
//функция изменяет размер выделенной памяти для вектора до его фактического
размера(то есть уменьшает выделенную память до минимально необходимого размера
для хранеения дданных)
 int int_vector_shrink_to_fit(IntVector *v)
 {
    bool shrink_to_fit_unsuccess = false; //флаг для отслежживания неуддачи
```

```
уменьшения размера вектора

    if(v -> size == 0 || v -> capacity == 0) //если размеер или емкость равны
нулю, то ничего уменьшать не нужно
    shrink_to_fit_unsuccess = true; //помечаем неудачу

    else
    {
        v -> capacity = v-> size; //изменяем емкость вектора на его текущий
размер
        v -> data = realloc(v -> data, INT_SIZE* v -> capacity);//изменяем
размер выделенной памяти для массива вектора на размер, равный размеру
элементов вектора умноженному на его емкость

        if (v -> data == NULL) //если выделить память не удалось, то помечаем
неудачу
        shrink_to_fit_unsuccess = true;
    }
    return -(int)shrink_to_fit_unsuccess; //возвращаем код ошибки, 0-успех, -1
- неудача
 }
//функция изменяет размер веткора "v" на размер "new_size"
 int int_vector_resize(IntVector *v, size_t new_size)
{
    if(new_size > v -> size) //если новый размер текущего размера вектора
    {
        if(new_size > v-> capacity) //если новый размер больше вместимости
вектора
        {
            int_vector_reserve(v, new_size * 2); //увеличиваем вместимость
вектора ддо двух раз больше нового размера
            for(int i = v -> size; i < v -> capacity; i++) //заполняем
недостающие элементы вектора нулями
            {
                v -> data[i] = 0;
            }
        }
        v -> size = new_size; //обновляем размер вектора
    }
    if (new_size < v -> size) //если новый размер меньше текущего размера
вектора
    {
        v -> size = new_size; //обновляем размер вектора
    }
    return 0;
}

//функция для изменения размера вектора "Intvector" на заданное значение
"new_size"
int int_vector_reserve(IntVector *v, size_t new_capacity)
{
    bool reserve_unsuccess = false; //флаг неудачного выделения памяти
    if (new_capacity < v -> capacity || new_capacity < 0) //если новая емкость
меньше текущей или отрицательная
        reserve_unsuccess = true; //установить флаг неудачи
    else //в противном случае
    {
```

```
        v ->capacity = new_capacity; //установить новую емкость
        v -> data = realloc(v -> data, INT_SIZE * new_capacity);
//перевыделить память
        if(v -> data == NULL) //если не удалось выделить память
        {
            print_error(ARRAY_MEMORY_ALLOCATION_ERROR); //вывести ошибку
            reserve_unsuccess = true; //установить флаг неудачи
        }
    }
    return -(int)reserve_unsuccess;
}
```

Main.c

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

int main()
{
// Создание и инициализация вектора a типа IntVector с начальной емкостью 5
printf("%s\n", "Объявление и инициализация вектора целых чисел с емкостью 5");
IntVector* a = int_vector_new(5);

// Вывод размера и емкости вектора a
printf("%s %ld %s%ld\n\n", "Размер вектора:", int_vector_get_size(a),".
Емкость:", int_vector_get_capacity(a));

// Изменение размера вектора a на 10 и вывод его нового размера
printf("%s\n", "Изменение размера вектора a, заполнение его элементами,
создание вектора b и копирование вектора a в b");
int_vector_resize(a, 10);
printf("%s %ld\n\n\n", "Теперь вектор имеет размер:", int_vector_get_size(a));

// Создание вектора b как копии вектора a и вывод размера и емкости обоих
векторов
IntVector* b = int_vector_copy(a);
printf("%s\n %s %p %ld %ld\n", "Теперь у нас есть два вектора:", "вектор a:",
a, int_vector_get_size(a), int_vector_get_capacity(a));
printf("%s %p %ld %ld\n\n\n", "вектор b:", b, int_vector_get_size(b),
int_vector_get_capacity(b));

// Освобождение памяти, занятой вектором b, сжатие емкости вектора a и вывод
его размера и емкости
int_vector_free(b);
int_vector_shrink_to_fit(a);
printf("%s %ld %s%ld\n\n", "Вывод размера и емкости векторов:",
int_vector_get_size(a),". Емкость:", int_vector_get_capacity(a));

// Вызов функции reserve, изменяющей емкость вектора a на 20 и вывод результата
printf("%d\n\n", int_vector_reserve(a, 20));
printf("%s %ld %s%ld\n\n", "Вывод размера и емкости векторов:",
int_vector_get_size(a),". Емкость:", int_vector_get_capacity(a));

// Вывод элементов вектора a и добавление в него нового элемента
for (size_t i = 0; i < int_vector_get_size(a); i++)
{
printf("%ld. %d\n", i, int_vector_get_item(a, i));
}
printf("\n\n\n\n");
int_vector_push_back(a, 1);

for (size_t i = 0; i < int_vector_get_size(a); i++)
{
printf("%ld. %d\n", i, int_vector_get_item(a, i));
}
printf("\nsize: %ld\n", int_vector_get_size(a));
printf("-----------------------------------\n");
int_vector_set_item(a, 9, -5); // устанавливаем значение -5 для 10-го элемента
вектора a
```

```
int_vector_pop_back(a); // удаляем последний элемент из вектора a
for (size_t i = 0; i < int_vector_get_size(a); i++)
{
printf("%ld. %d\n", i, int_vector_get_item(a, i));
}
printf("\nsize: %ld\n", int_vector_get_size(a));
return 0;
}
```