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. ИВ-222  «4» мая 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | Очнев А.Д. |
| --- | --- | --- |
|  |  |  |
| Проверил:  Старший преподаватель кафедры вычислительных систем  «5» мая 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | Фульман В.О. |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[ЗАДАНИЕ 3](#_30j0zll)

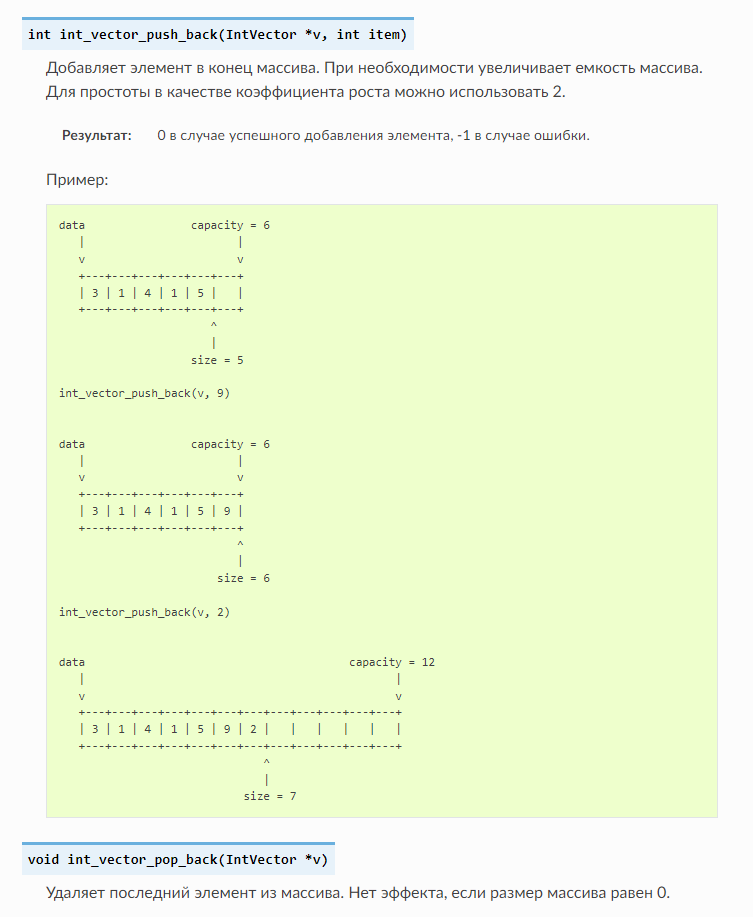
[ВЫПОЛНЕНИЕ РАБОТЫ 6](#_1fob9te)

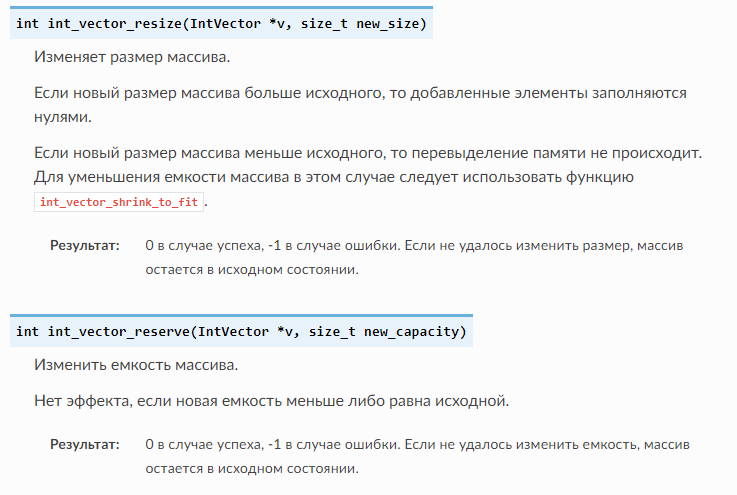
[ПРИЛОЖЕНИЕ 15](#_tyjcwt)

## **ЗАДАНИЕ**

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







## **ВЫПОЛНЕНИЕ РАБОТЫ**

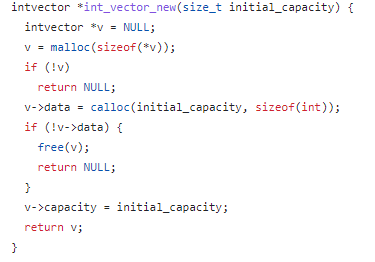
Создаём заголовочный файл IntVector.h, где объявляем шаблон структуры и прототипы функций:



Реализовываем функции в файле IntVector.c:

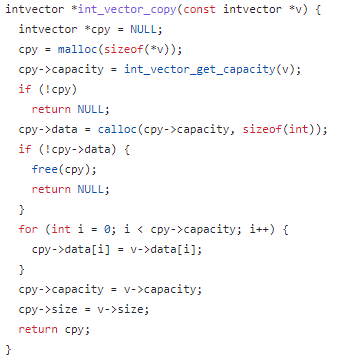
**int\_vector\_new**

Выделяем память для структуры с помощью функции malloc, если выделение не произошло, то возвращаем NULL.Выделяем память для массива data и заполняем нулями с помощью функции calloc, если выделение не произошло, то освобождаем память выделенную под структуру и возвращаем NULL Приравниваем полю структуры capacity значение переданной ёмкости. Возвращаем указатель на структуру.



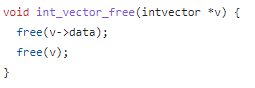
**int\_vector\_copy**

Выделяем память для структуры с помощью функции malloc, если выделение не произошло, то возвращаем NULL.Выделяем память для массива data и заполняем нулями с помощью функции calloc, если выделение не произошло, то освобождаем память выделенную под структуру и возвращаем NULL. Приравниваем полям структуры capacity и size значения переданной структуры. Возвращаем указатель на структуру.



**int\_vector\_free**

Чтобы избежать утечек памяти, сначала очищаем память для массива, затем для структуры.



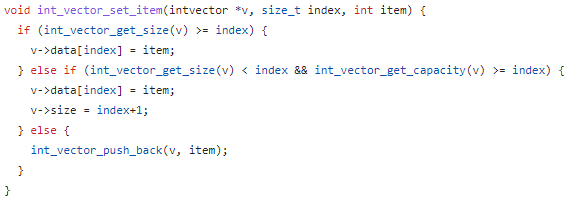
**int\_vector\_get\_item**

Возвращаем значение массива по индексу index.



**int\_vector\_set\_item**

Если размер массива больше или равен index, тогда мы приравниваем элемент массива по индексу index к значению item. Если переданное значение index лежит в диапазоне от размера до ёмкости, то тогда мы приравниваем элемент массива по индексу index к значению item, после чего приравниваем размер к index. Если index больше ёмкости, то мы добавляем элемент item в конец массива с помощью функции push\_back.



**int\_vector\_get\_size**

Возвращаем значение поля size из структуры.



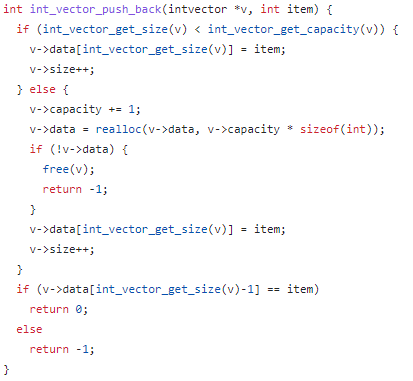
**int\_vector\_get\_capacity**

Возвращаем значение поля capacity из структуры.



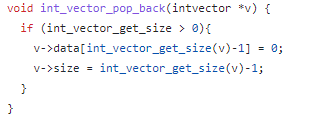
**int\_vector\_push\_back**

Если размер массива меньше его емкости, то мы приравниваем элемент массива по индексу size к значению item, после чего увеличиваем размер на один. Если размер равен емкости, то мы увеличиваем емкость на один, после чего довыделяем память для массива. Если выделение не произошло, то освобождаем память выделенную под структуру и возвращаем -1. Приравниваем элемент массива по индексу size к значению item, после чего увеличиваем размер на один. При успешной инициализации возвращаем 0, -1 если произошла ошибка.

****

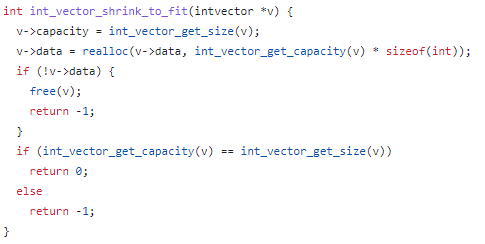
**int\_vector\_pop\_back**

Если размер массива больше 0, то мы приравниваем элемент массива по индексу size-1 к нулю, после чего уменьшаем размер на один.



**int\_vector\_shrink\_to\_fit**

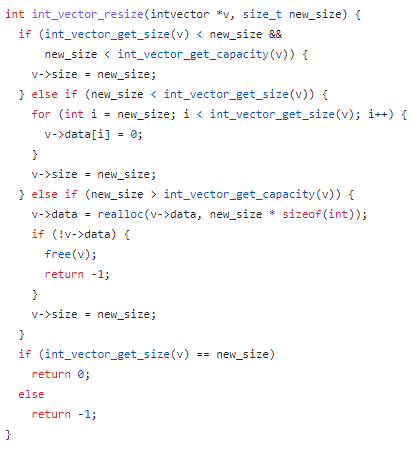
Приравниваем емкость массива к его размеру, после чего довыделяем память для массива. Если выделение не произошло, то освобождаем память выделенную под структуру и возвращаем -1. При успешном уменьшении ёмкости возвращаем 0, -1 если произошла ошибка.



**int\_vector\_resize**

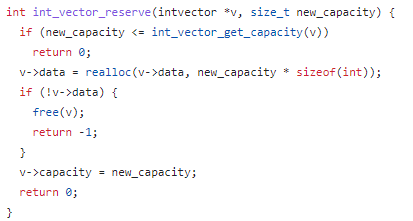
Если новый размер больше текущего, но меньше емкости, то размер массива приравниваем к новому размеру. Если новый размер меньше текущего, то приравниваем каждый элемент массива по индексам от нового размера до старого к нулю, после чего размер массива приравниваем к новому размеру. Если новый размер больше ёмкости, то довыделяем память для массива функцией realloc, проверяем, выделилась ли память, после чего размер массива приравниваем к новому размеру. При успешном

изменении размера возвращаем 0, -1 если произошла ошибка.

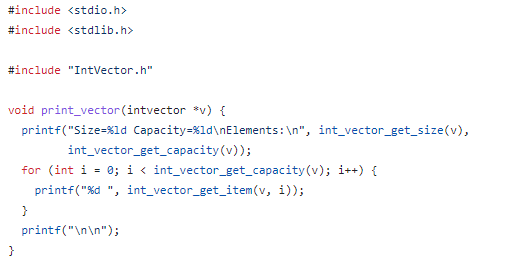


**int\_vector\_reserve**

Если новая емкость меньше или равна старой емкости массива, то возвращаем 0. Иначе довыделяем память для массива функцией realloc, проверяем, выделилась ли память. Изменяем значение емкости на новое.

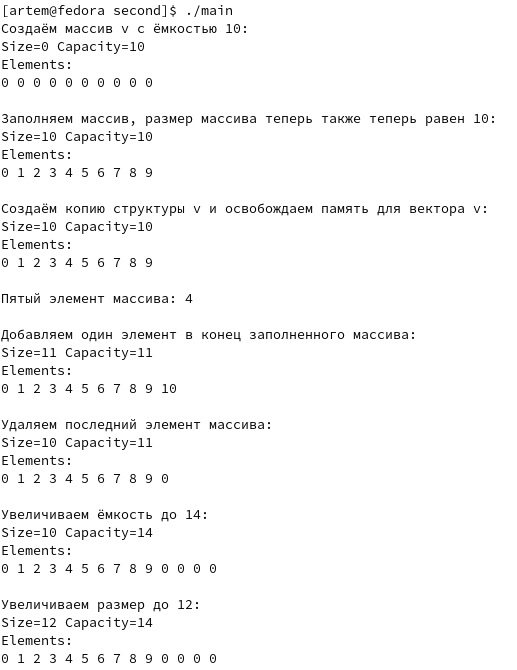


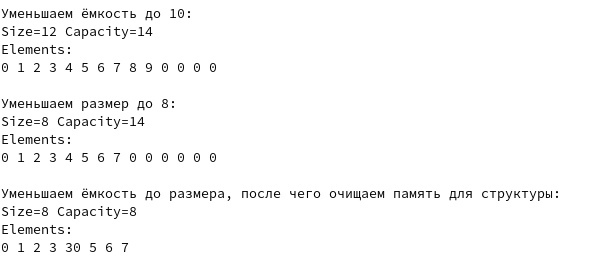
В файле main.c проверяем работу функций:



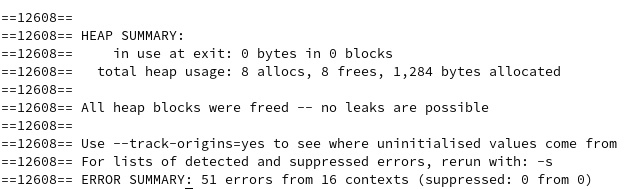
## 

Вывод:





Проверим на наличие утечек с помощью команды valgrind:



Утечек не обнаружено.

## 

## **ПРИЛОЖЕНИЕ**

**main.c**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | #include <stdio.h>  #include <stdlib.h>  #include "IntVector.h"  **void** **print\_vector**(intvector \*v) {  printf("Size=%ld Capacity=%ld**\n**Elements:**\n**", int\_vector\_get\_size(v),  int\_vector\_get\_capacity(v));  **for** (**int** i = **0**; i < int\_vector\_get\_capacity(v); i++) {  printf("%d ", int\_vector\_get\_item(v, i));  }  printf("**\n\n**");  }  **int** **main**() {  intvector \*v = int\_vector\_new(**10**);  printf("Создаём массив v с ёмкостью 10:**\n**");  print\_vector(v);    **for** (**int** i = **0**; i < **10**; i++) {  int\_vector\_set\_item(v, i, i);  }  printf("Заполняем массив, размер массива теперь также теперь равен 10:**\n**");  print\_vector(v);    intvector \*t = int\_vector\_copy(v);  int\_vector\_free(v);  printf("Создаём копию структуры v и освобождаем память для вектора v:**\n**");  print\_vector(t);    printf("Пятый элемент массива: %d**\n\n**", int\_vector\_get\_item(t, **4**));    printf("Добавляем один элемент в конец заполненного массива:**\n**");  int\_vector\_push\_back(t, **10**);  print\_vector(t);    printf("Удаляем последний элемент массива:**\n**");  int\_vector\_pop\_back(t);  print\_vector(t);    printf("Увеличиваем ёмкость до 14:**\n**");  int\_vector\_reserve(t, **14**);  print\_vector(t);    printf("Увеличиваем размер до 12:**\n**");  int\_vector\_resize(t, **12**);  print\_vector(t);    printf("Уменьшаем ёмкость до 10:**\n**");  int\_vector\_reserve(t, **10**);  print\_vector(t);    printf("Уменьшаем размер до 8:**\n**");  int\_vector\_resize(t, **8**);  print\_vector(t);    printf("Уменьшаем ёмкость до размера, после чего очищаем память для структуры:**\n**");  int\_vector\_shrink\_to\_fit(t);  int\_vector\_set\_item(t, **4**, **30**);  print\_vector(t);    int\_vector\_free(t);  **return** **0**;  } |
| --- | --- |

**IntVector.h**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | #pragma once  **typedef** **struct**  {  **int**\* data;  **size\_t** size;  **size\_t** capacity;  } 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); |
| --- | --- |

**IntVector.c**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152 | #include <stdlib.h>  #include "IntVector.h"  intvector \***int\_vector\_new**(**size\_t** initial\_capacity) {  intvector \*v = NULL;  v = malloc(**sizeof**(\*v));  **if** (!v)  **return** NULL;  v->data = calloc(initial\_capacity, **sizeof**(**int**));  **if** (!v->data) {  free(v);  **return** NULL;  }  v->capacity = initial\_capacity;  **return** v;  }  intvector \***int\_vector\_copy**(**const** intvector \*v) {  intvector \*cpy = NULL;  cpy = malloc(**sizeof**(\*v));  cpy->capacity = int\_vector\_get\_capacity(v);  **if** (!cpy)  **return** NULL;  cpy->data = calloc(cpy->capacity, **sizeof**(**int**));  **if** (!cpy->data) {  free(cpy);  **return** NULL;  }  **for** (**int** i = **0**; i < cpy->capacity; i++) {  cpy->data[i] = v->data[i];  }  cpy->capacity = v->capacity;  cpy->size = v->size;  **return** cpy;  }  **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) {  **if** (int\_vector\_get\_size(v) >= index) {  v->data[index] = item;  } **else** **if** (int\_vector\_get\_size(v) < index && int\_vector\_get\_capacity(v) >= index) {  v->data[index] = item;  v->size = index+**1**;  } **else** {  int\_vector\_push\_back(v, item);  }  }  **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** (int\_vector\_get\_size(v) < int\_vector\_get\_capacity(v)) {  v->data[int\_vector\_get\_size(v)] = item;  v->size++;  } **else** {  v->capacity += **1**;  v->data = realloc(v->data, v->capacity \* **sizeof**(**int**));  **if** (!v->data) {  free(v);  **return** -**1**;  }  v->data[int\_vector\_get\_size(v)] = item;  v->size++;  }  **if** (v->data[int\_vector\_get\_size(v)-**1**] == item)  **return** **0**;  **else**  **return** -**1**;  }  **void** **int\_vector\_pop\_back**(intvector \*v) {  **if** (int\_vector\_get\_size > **0**){  v->data[int\_vector\_get\_size(v)-**1**] = **0**;  v->size = int\_vector\_get\_size(v)-**1**;  }  }  **int** **int\_vector\_shrink\_to\_fit**(intvector \*v) {  v->capacity = int\_vector\_get\_size(v);  v->data = realloc(v->data, int\_vector\_get\_capacity(v) \* **sizeof**(**int**));  **if** (!v->data) {  free(v);  **return** -**1**;  }  **if** (int\_vector\_get\_capacity(v) == int\_vector\_get\_size(v))  **return** **0**;  **else**  **return** -**1**;  }  **int** **int\_vector\_resize**(intvector \*v, **size\_t** new\_size) {  **if** (int\_vector\_get\_size(v) < new\_size &&  new\_size < int\_vector\_get\_capacity(v)) {  v->size = new\_size;  } **else** **if** (new\_size < int\_vector\_get\_size(v)) {  **for** (**int** i = new\_size; i < int\_vector\_get\_size(v); i++) {  v->data[i] = **0**;  }  v->size = new\_size;  } **else** **if** (new\_size > int\_vector\_get\_capacity(v)) {  v->data = realloc(v->data, new\_size \* **sizeof**(**int**));  **if** (!v->data) {  free(v);  **return** -**1**;  }  v->size = new\_size;  }  **if** (int\_vector\_get\_size(v) == new\_size)  **return** **0**;  **else**  **return** -**1**;  }  **int** **int\_vector\_reserve**(intvector \*v, **size\_t** new\_capacity) {  **if** (new\_capacity <= int\_vector\_get\_capacity(v))  **return** **0**;  v->data = realloc(v->data, new\_capacity \* **sizeof**(**int**));  **if** (!v->data) {  free(v);  **return** -**1**;  }  v->capacity = new\_capacity;  **return** **0**;  } |
| --- | --- |