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-242  «25» мая 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Игнатенко Г. Д./ |
|  |  |  |
| Проверил:  Старший Преподаватель Кафедры ВС  «26» мая 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Фульман В.О../ |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[ЗАДАНИЕ 3](#_Toc35593781)

[ВЫПОЛНЕНИЕ РАБОТЫ 4](#_Toc35593782)

[ПРИЛОЖЕНИЕ 15](#_Toc35593783)

# **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**

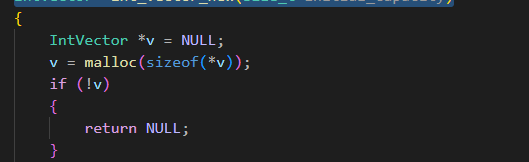
Описывается ход работы над заданием с приложением снимков экрана;

1. Функция

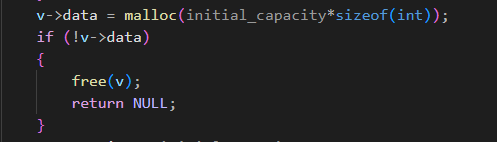
Функaция IntVector \*int\_vector\_new(size\_t initial\_capacity)

Создает массив нулевого размера.

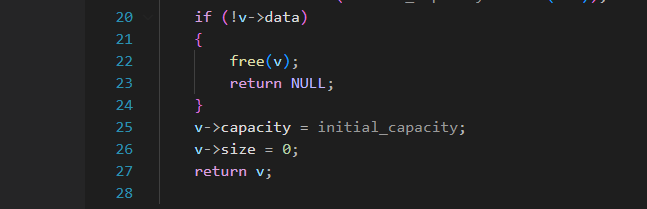
Выделяем под структуру память, в случае, если программа не сможет выделить память, вернуть нулевой указатель.



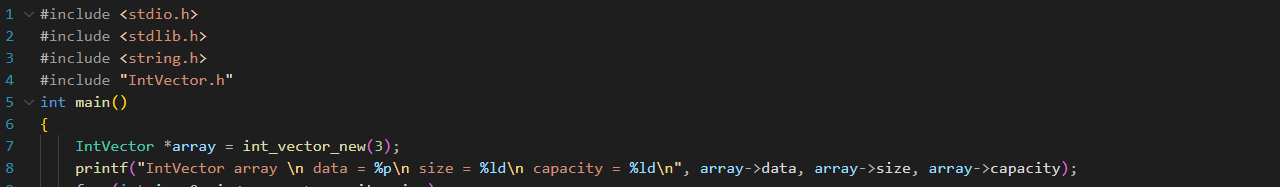
Далее выделяем память под массив структуры и в случае ошибки, освободить память, вернуть нулевой указатель.

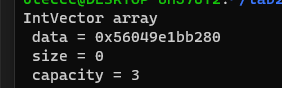


Если все прошло успешно, и программе удалось выделить память, присвоим в структуре переменной емкости, значение, которое было подано с вызовом функции.  
Зададим переменной size 0, так как пока, в массиве ничего не лежит. Вернем указатель на вектор.



Проверим, как работает функция





Capacity = 3, т.к мы подали в функцию данное значение.

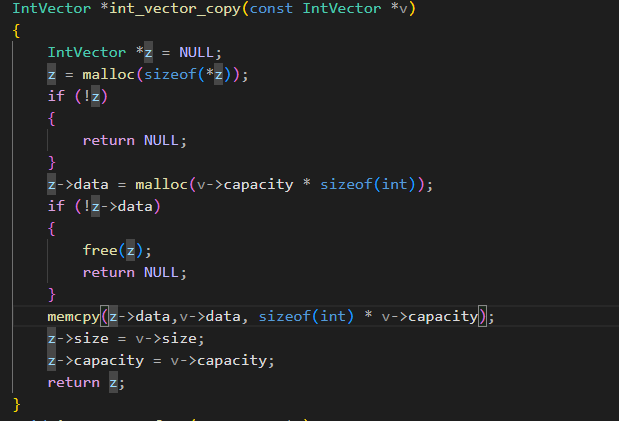
1. Функция

IntVector \*int\_vector\_copy(const IntVector \*v)

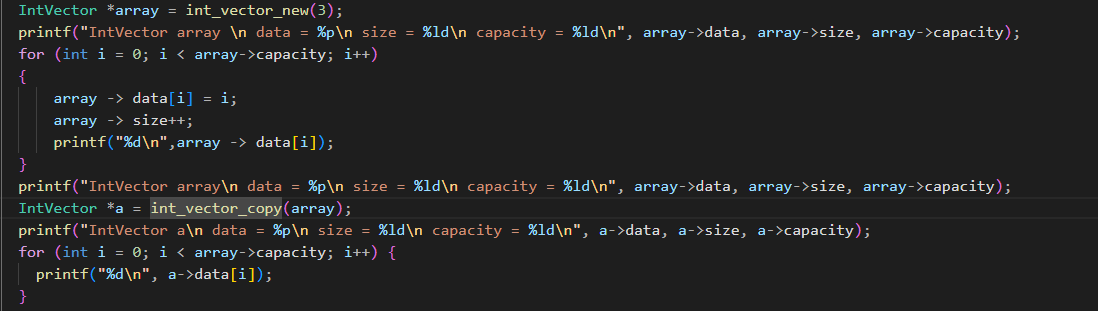
Указатель на копию вектора.

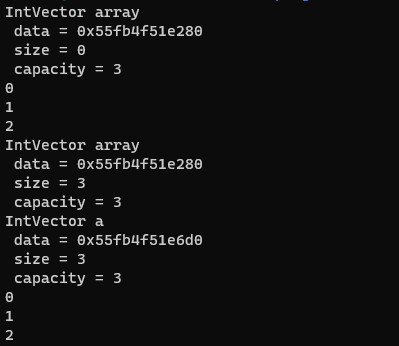
Проводим те же самые действия, только с новым указателем, созданным в функции и выделяем емкость, которая изначально была в векторе V.

После, копируем элементы из одного вектора, в другой, присваиваем значение емкости и размера. Возвращаем указатель на скопированный вектор v.



Запишем значения в вектор array некоторые значения и проверим, как будет работать функция int\_vector\_copy, затем выведем, что лежит в IntVector a.





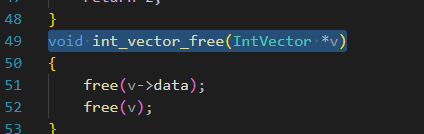
Функция работает правильно, и полностью копирует подаваемый вектор.

1. Функция

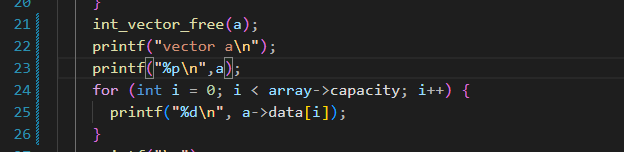
void int\_vector\_free(IntVector \*v)

Освобождает выделенную память.

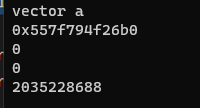
Освобождаем память массива data, а после и полностью указателя на структуру.



После отработки функции, выведем указатель и посмотрим, какие элементы до сих пор хранятся в a->data.



Как видим, указатель на память остался, но в нем уже лежит всякий мусор.

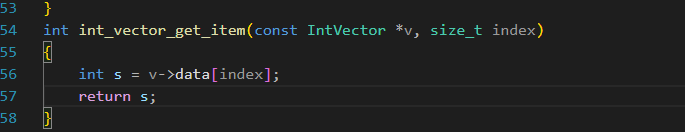


1. Функция

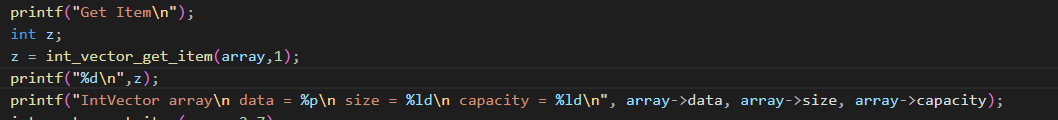
int int\_vector\_get\_item(const IntVector \*v, size\_t index)

Возвращает элемент под номером index.

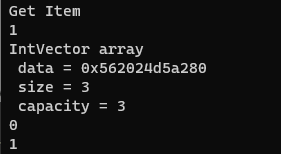
Создаем новую переменную для значения и после возвращаем данный элемент.



Попробуем взять 1 индекс элемента, должно вывести один.



Выводит один и в первом индексе array->data лежит единица.

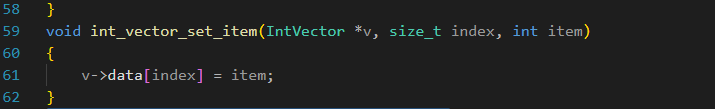


1. Функция

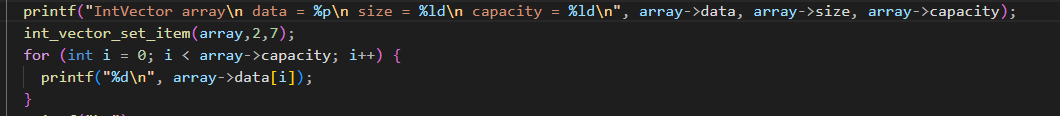
void int\_vector\_set\_item(IntVector \*v, size\_t index, int item)

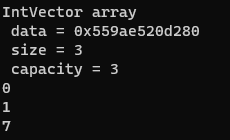
Присваивает элементу под номером index значение item.

Меняем значение элемента по нахождению индекса на новый элемент, заданный в вызове функции.



Вызовем функция и заменим второй индекс на элемент 7.





Изначально, во втором индексе лежало двойка, после применения функции, стала цифра 7,

1. Функция

size\_t int\_vector\_get\_size(const IntVector \*v)

Размер вектора.

Возвращаем значение size в поданном указатели на структуру IntVector.

1. Функция

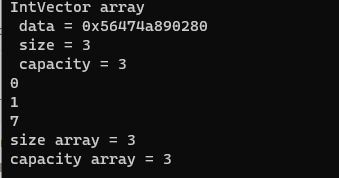
size\_t int\_vector\_get\_capacity(const IntVector \*v)

Емкость вектора.

Возвращаем значение емкости в поданном указатели на структуру IntVector.

Объединим функции под номер 7 и 8. Присвоим значения, возращаемые данные функция переменным типа size\_t и выведем значения на экран.





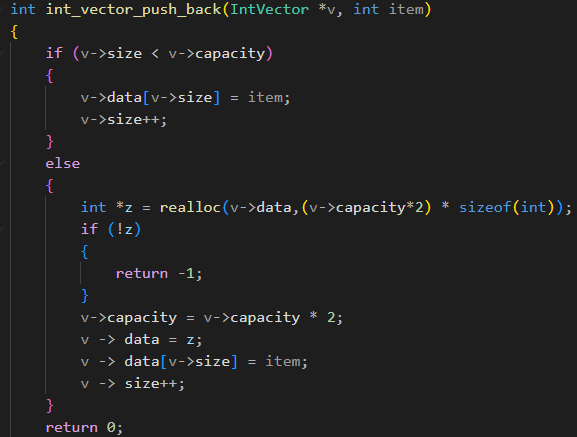
Как видим, size и capacity были равны значениям 3, функция и вернула эти значения.

1. Функция

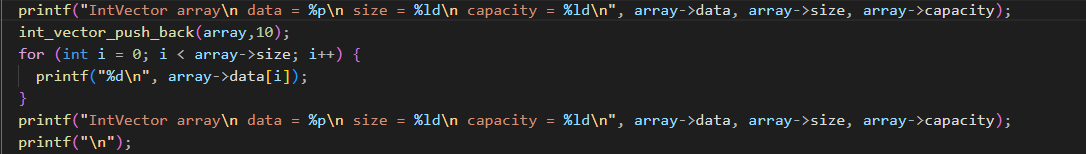
int int\_vector\_push\_back(IntVector \*v, int item)

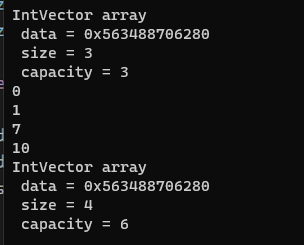
Добавляет элемент в конец массива. При необходимости увеличивает емкость массива. Для простоты в качестве коэффициента роста можно использовать 2.

Проверяем, есть ли место для нашего элемента, изначально, в поданном указателе.  
Если есть, то просто присваиваем и увеличиваем размерность на 1 , т.к добавлен 1 элемент.  
А иначе, увеличиваем емкость вдвое. Создаем новый указатель и присваиваем в него добавленную память, сохраняя с прошлыми значения(realloc). Если выделить не поучилось, вернем нулевой указатель. Кладем другой указатель на память в v data, после присваиваем элемент и прибавляем к размеру 1, т.к размер увеличился на единицу.



Добавим элемент 10 в конец, с помощью функции push\_back и выведем значения на экран.





Как видим, элемент 10 добавился в массив и емкость увеличилась вдвое и размер увеличился на 1, как и прописано в функции. Указатель не изменился, т.к. была последовательная память для выделения.

1. Функция

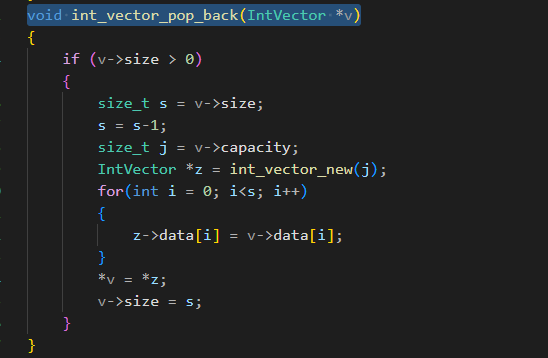
void int\_vector\_pop\_back(IntVector \*v)

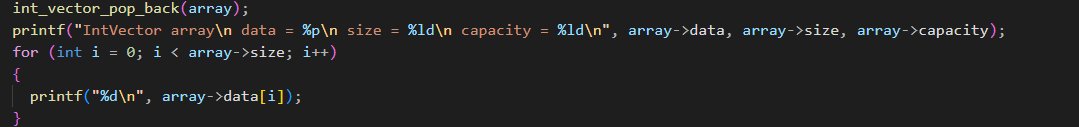
Удаляет последний элемент из массива. Нет эффекта, если размер массива равен 0

j, после, заполняем его до предпоследнего включительно значений data, а далее

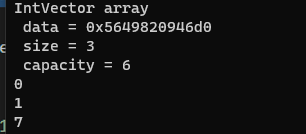
переприсваем указатель \*v и кладем в v – size значение размера, измененного на

единицу, т.к нужно удалить 1 элемент массива.





Вызовем функцию и выведем значения на экран.



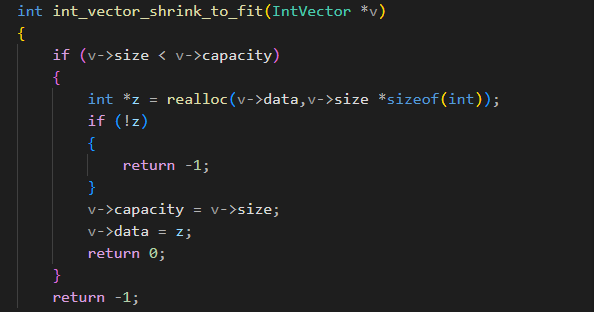
Как видим, размер уменьшился на единицу и удалился элемент 10.

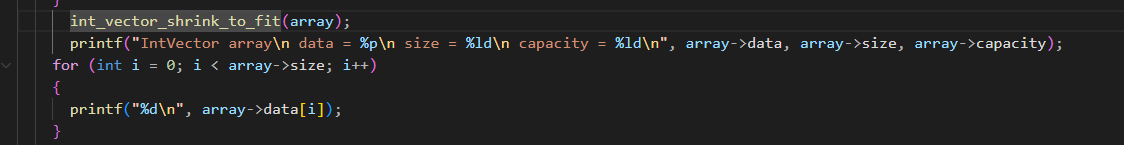
1. Функция

int int\_vector\_shrink\_to\_fit(IntVector \*v)

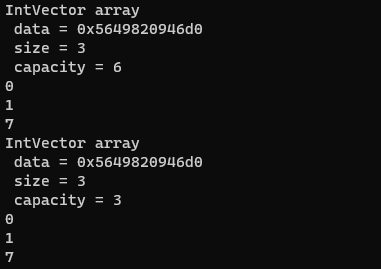
Уменьшает емкость массива до его размера.

Если наш размер меньше, чем емкость, то присваиваем значение емкости size и выделяем память и проверяем, удалось ли это сделать, после присваиваем указателю data новый указатель(не всегда новый указатель, если есть возможность выделить последовательно память) на новую выделенную память, сохранившая все значения.





Вызовем функция и выведем значения структуры и массива на экран.



Как видим, значения остались те же, емкость уменьшилась до размера.

1. Функция

int int\_vector\_resize(IntVector \*v, size\_t new\_size)

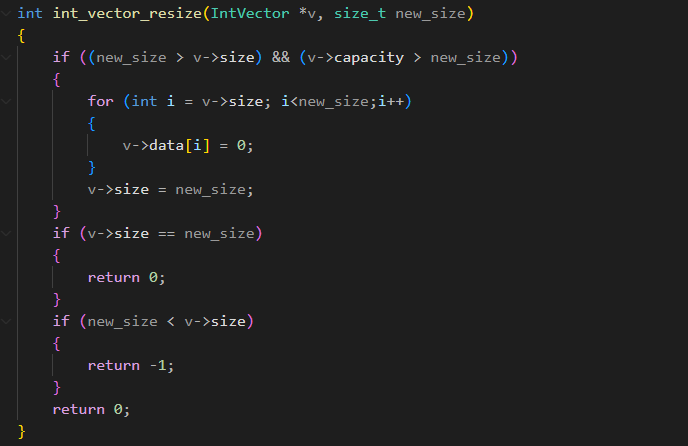
Изменяет размер массива.

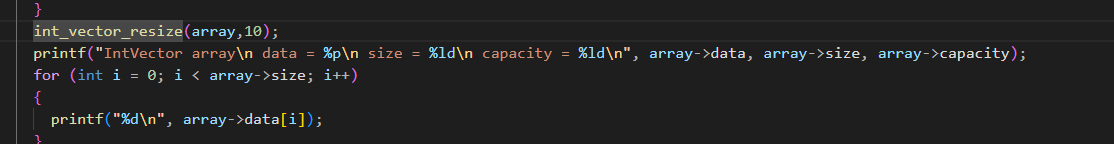
Если новый размер больше, чем нынешний. Создаем новый указатель на выделенную память, сохраняя нынешние значения. Если не получилось выделить память, возвращаем NULL. Записываем новый указатель в указатель data. После зануляем все значения до new\_size(новой размерности).

Далее если емкость больше либо равна размерности, оставляем емкость, какая и была.

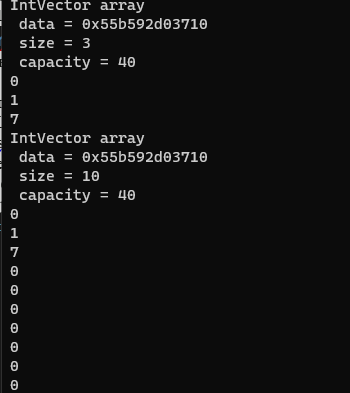
Если емкость меньше, чем размерность, присваиваем емкости, размер.

Если размер равен новому размеру, ничего не делаем, а если меньше, то вернем ошибку.





Вызовем функцию и выведем поля структуры, и все значения, лежащие в data.



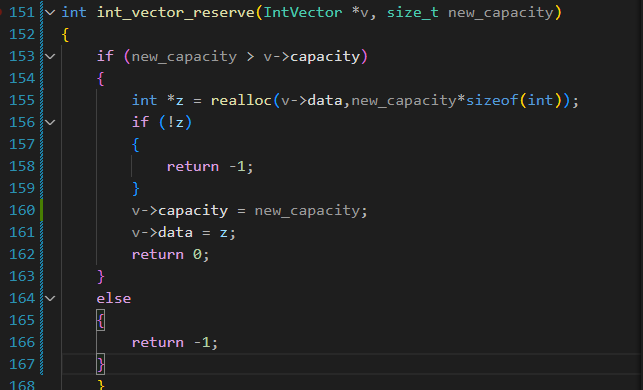
Как видим, емкость увеличилась до размерности, т.к. размерность превышала емкость, и все элементы, которые не заданы, обращены в ноль.

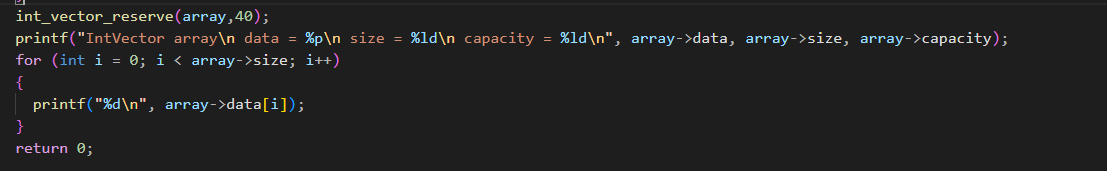
1. Функция

int int\_vector\_reserve(IntVector \*v, size\_t new\_capacity)

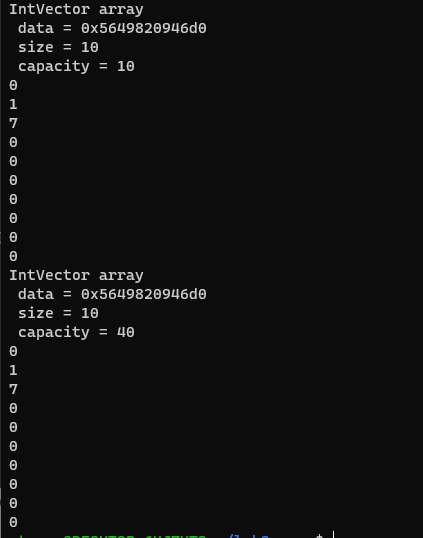
Изменить емкость массива.

Если новая емкость, больше нынешней, то перезаписываем значение capacity. После создаем указатель на новую увеличенную память, сохраняя изменения прошлой. Если не удалось выделить, вернем -1. После присвоим новую память в старую v->data. В любых других случаях вернем ошибку.





Вызовем функцию и увеличим емкость до 40, выведем все поля структуры и посмотрим, сохранились ли все наши значения.



Как видим, емкость увеличилась до 40, и все значения, лежащие в data сохранены.

# **ПРИЛОЖЕНИЕ**

Исходный код с комментариями;

1. IntVector \***int\_vector\_new**(**size\_t** initial\_capacity)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | IntVector \***int\_vector\_new**(**size\_t** initial\_capacity)  {  IntVector \*v = NULL;  v = malloc(**sizeof**(\*v));  **if** (!v)  {  **return** NULL;  }  v->data = malloc(initial\_capacity\***sizeof**(**int**));  **if** (!v->data)  {  free(v);  **return** NULL;  }  v->capacity = initial\_capacity;  v->size = **0**;  **return** v; |

2. IntVector \***int\_vector\_copy**(**const** IntVector \*v)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | IntVector \***int\_vector\_copy**(**const** IntVector \*v)  {  IntVector \*z = NULL;  z = malloc(**sizeof**(\*z));  **if** (!z)  {  **return** NULL;  }  z->data = malloc(v->capacity \* **sizeof**(**int**));  **if** (!z->data)  {  free(z);  **return** NULL;  }  memcpy(z->data,v->data, **sizeof**(**int**) \* v->capacity);  z->size = v->size;  z->capacity = v->capacity;  **return** z;  } |

3. **void** **int\_vector\_free**(IntVector \*v)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **void** **int\_vector\_free**(IntVector \*v)  {  free(v->data);  free(v);  } |

4. **int** **int\_vector\_get\_item**(**const** IntVector \*v, **size\_t** index)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **int** **int\_vector\_get\_item**(**const** IntVector \*v, **size\_t** index)  {  **int** s = v->data[index];  **return** s;  } |

5. **void** **int\_vector\_set\_item**(IntVector \*v, **size\_t** index, **int** item)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **void** **int\_vector\_set\_item**(IntVector \*v, **size\_t** index, **int** item)  {  v->data[index] = item;  } |

6. **size\_t** **int\_vector\_get\_size**(**const** IntVector \*v)

|  |  |
| --- | --- |
| 1  2  3  4  5 | **size\_t** **int\_vector\_get\_size**(**const** IntVector \*v)  {  **return** v->size;  } |

7. **size\_t** **int\_vector\_get\_capacity**(**const** IntVector \*v)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **size\_t** **int\_vector\_get\_capacity**(**const** IntVector \*v)  {  **return** v->capacity;  } |

8. **int** **int\_vector\_push\_back**(IntVector \*v, **int** item)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | **int** **int\_vector\_push\_back**(IntVector \*v, **int** item)  {  **if** (v->size < v->capacity)  {  v->data[v->size] = item;  v->size++;  }  **else**  {  **int** \*z = realloc(v->data,v->capacity \* **sizeof**(**int**));  **if** (!z)  {  **return** -**1**;  }  v->capacity = v->capacity \* **2**;  v -> data = z;  v -> data[v->size] = item;  v -> size++;  }  **return** **0**;  } |

9. **void** **int\_vector\_pop\_back**(IntVector \*v)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **void** **int\_vector\_pop\_back**(IntVector \*v)  {  **if** (v->size > **0**)  {  **size\_t** s = v->size;  s = s-**1**;  **size\_t** j = v->capacity;  IntVector \*z = int\_vector\_new(j);  **for**(**int** i = **0**; i<s; i++)  {  z->data[i] = v->data[i];  }  \*v = \*z;  v->size = s;  }  } |

10. **int** **int\_vector\_shrink\_to\_fit**(IntVector \*v)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **int** **int\_vector\_shrink\_to\_fit**(IntVector \*v)  {  **if** (v->size < v->capacity)  {  **int** \*z = realloc(v->data,v->size \***sizeof**(**int**));  **if** (!z)  {  **return** -**1**;  }  v->capacity = v->size;  v->data = z;  **return** **0**;  }  **return** -**1**;  } |

11. **int** **int\_vector\_resize**(IntVector \*v, **size\_t** new\_size)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **int** **int\_vector\_resize**(IntVector \*v, **size\_t** new\_size)  {  **if** ((new\_size > v->size) && (v->capacity > new\_size))  {  **for** (**int** i = v->size; i<new\_size;i++)  {  v->data[i] = **0**;  }  v->size = new\_size;  }  **if** (v->size == new\_size)  {  **return** **0**;  }  **if** (new\_size < v->size)  {  **return** -**1**;  }  **return** **0**;  } |

1. **int** **int\_vector\_reserve**(IntVector \*v, **size\_t** new\_capacity)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | **int** **int\_vector\_reserve**(IntVector \*v, **size\_t** new\_capacity)  {  **if** (new\_capacity > v->capacity)  {  v->capacity = new\_capacity;  **int** \*z = realloc(v->data,new\_capacity\***sizeof**(**int**));  **if** (!z)  {  **return** -**1**;  }  v->data = z;  **return** **0**;  }  **else**  {  **return** -**1**;  }  } |

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  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87 | #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include "IntVector.h"  **int** **main**()  {  IntVector \*array = int\_vector\_new(**3**);  printf("IntVector array **\n** data = %p**\n** size = %ld**\n** capacity = %ld**\n**", array->data, array->size, array->capacity);  **for** (**int** i = **0**; i < array->capacity; i++)  {  array -> data[i] = i;  array -> size++;  printf("%d**\n**",array -> data[i]);  }  printf("IntVector array**\n** data = %p**\n** size = %ld**\n** capacity = %ld**\n**", array->data, array->size, array->capacity);  IntVector \*a = int\_vector\_copy(array);  printf("IntVector a**\n** data = %p**\n** size = %ld**\n** capacity = %ld**\n**", a->data, a->size, a->capacity);  **for** (**int** i = **0**; i < array->capacity; i++) {  printf("%d**\n**", a->data[i]);  }  printf("vector a**\n**");  printf("%p**\n**",a);  **for** (**int** i = **0**; i < array->capacity; i++) {  printf("%d**\n**", a->data[i]);  }  printf("**\n**");  printf("Get Item**\n**");  **int** z;  z = int\_vector\_get\_item(array,**1**);  printf("%d**\n**",z);  printf("IntVector array**\n** data = %p**\n** size = %ld**\n** capacity = %ld**\n**", array->data, array->size, array->capacity);  int\_vector\_set\_item(array,**2**,**7**);  **for** (**int** i = **0**; i < array->capacity; i++) {  printf("%d**\n**", array->data[i]);  }  **size\_t** s = int\_vector\_get\_size(array);  **size\_t** jos = int\_vector\_get\_capacity(array);  printf("size array = %ld**\n**",s);  printf("capacity array = %ld**\n**",jos);  printf("**\n**");  printf("IntVector array**\n** data = %p**\n** size = %ld**\n** capacity = %ld**\n**", array->data, array->size, array->capacity);  int\_vector\_push\_back(array,**10**);  **for** (**int** i = **0**; i < array->size; i++) {  printf("%d**\n**", array->data[i]);  }  printf("IntVector array**\n** data = %p**\n** size = %ld**\n** capacity = %ld**\n**", array->data, array->size, array->capacity);  printf("**\n**");  printf("gb**\n**");  int\_vector\_pop\_back(array);  printf("IntVector array**\n** data = %p**\n** size = %ld**\n** capacity = %ld**\n**", array->data, array->size, array->capacity);  **for** (**int** i = **0**; i < array->size; i++)  {  printf("%d**\n**", array->data[i]);  }  int\_vector\_shrink\_to\_fit(array);  printf("IntVector array**\n** data = %p**\n** size = %ld**\n** capacity = %ld**\n**", array->data, array->size, array->capacity);  **for** (**int** i = **0**; i < array->size; i++)  {  printf("%d**\n**", array->data[i]);  }  int\_vector\_reserve(array,**40**);  printf("IntVector array**\n** data = %p**\n** size = %ld**\n** capacity = %ld**\n**", array->data, array->size, array->capacity);  **for** (**int** i = **0**; i < array->size; i++)  {  printf("%d**\n**", array->data[i]);  }  int\_vector\_resize(array,**10**);  printf("IntVector array**\n** data = %p**\n** size = %ld**\n** capacity = %ld**\n**", array->data, array->size, array->capacity);  **for** (**int** i = **0**; i < array->size; i++)  {  printf("%d**\n**", array->data[i]);  }  int\_vector\_free(array);  int\_vector\_free(a);  **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 | **typedef** **struct**  {  **size\_t** size;  **size\_t** capacity;  **int** \*data;  } 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); |

**Ссылка на репозиторий:** [**https://github.com/Flar1/lab2\_prog**](https://github.com/Flar1/lab2_prog)