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-221  15 марта 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | Гордов Р.С. |
|  |  |  |
| Проверил:  Старший преподаватель кафедры ВС  «20» марта 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | Фульман В.О. |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[**ЗАДАНИЕ** 3](#_Toc130139184)

[**ВЫПОЛНЕНИЕ РАБОТЫ** 4](#_Toc130139185)

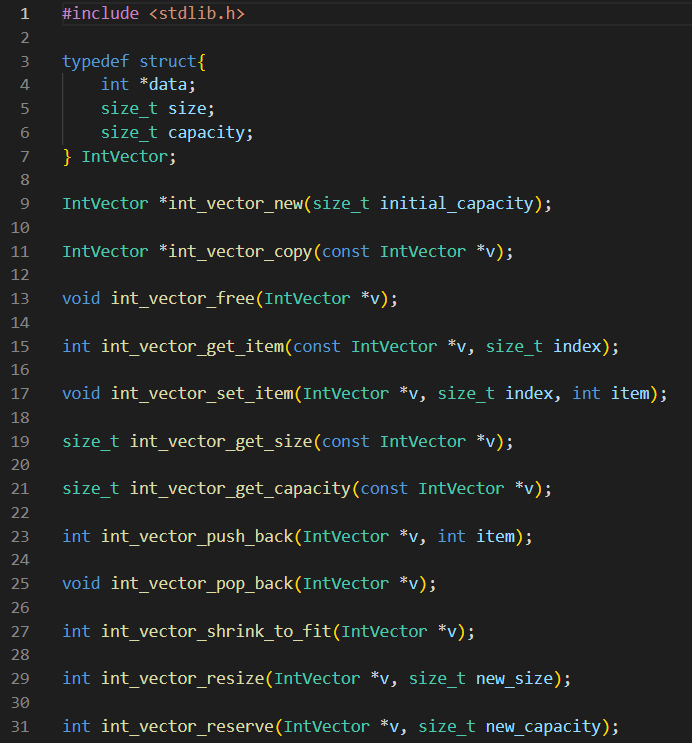
[**ПРИЛОЖЕНИЕ** 10](#_Toc130139186)

# **ЗАДАНИЕ**

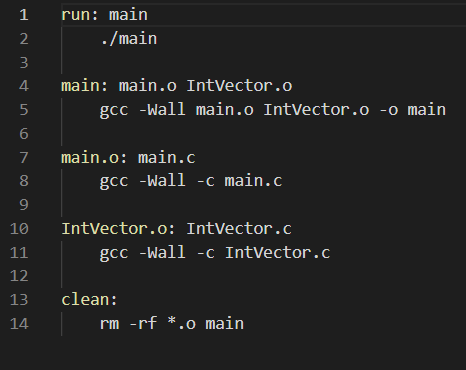
Реализовать тип данных «Динамический массив целых чисел» — IntVector и основные функции для работы с ним. Разработать тестовое приложение для демонстрации реализованных функций.

**ВЫПОЛНЕНИЕ РАБОТЫ**

Создадим файл с расширением .h, в нем будет лежать структура, содержащая три поля — данные, размер и емкость нашего массива, а также прототипы функций.



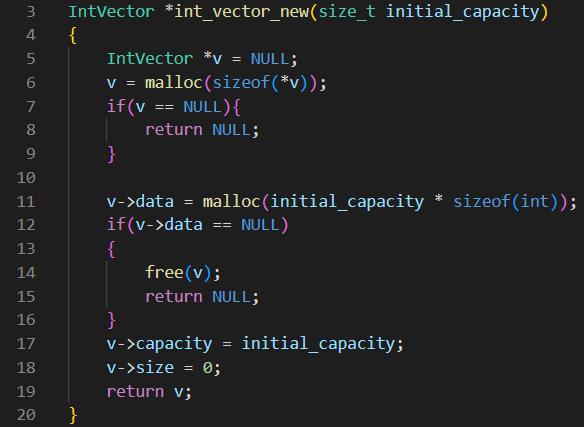
Для удобства запуска программы создадим Makefile.



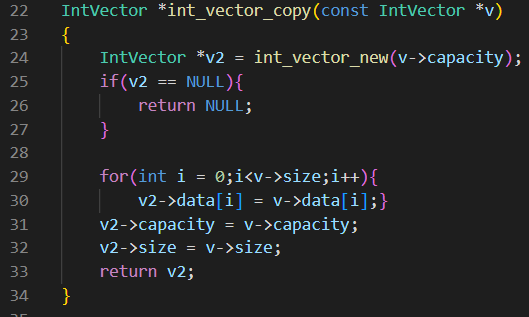
В файле IntVector.c будет реализация функций для работы с нашим вектором.

Напишем функцию, которая создает динамический массив. Она принимает изначальную емкость и возвращает указатель на нашу структуру.

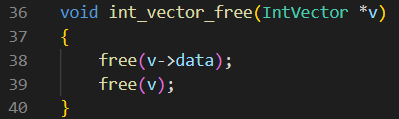
В случае, если не удается выделить место либо под структуру, либо под сами данные, возвращает нулевой указатель, если удалось выделить память, записываем в структуру начальные значение емкости и размера.



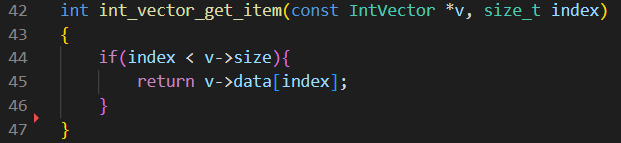
Реализуем функцию, которая возвращает указатель на копию нашего динамического массива. Она создает новый вектор, в который копирует все данные из предыдущего, и возвращает указатель на новый вектор.



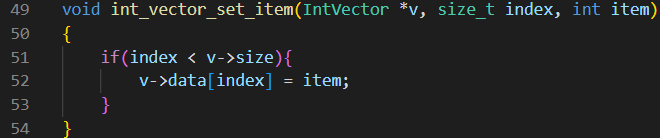
Функция int\_vector\_free принимает указатель на наш массив и освобождает выделенную для него память.



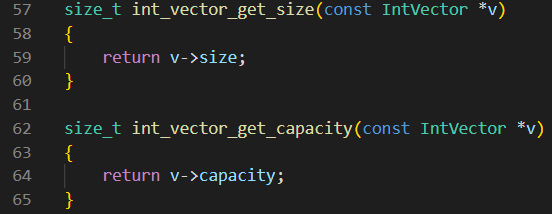
Функция int\_vector\_get\_item возвращает элемент под определенным индексом, если такой индекс существует.



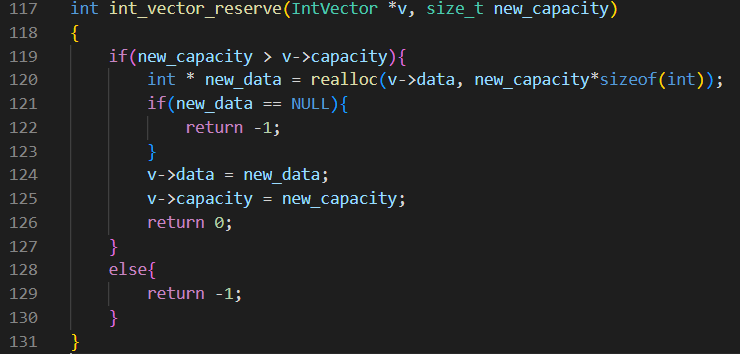
Функция int\_vector\_set\_item изменяет значение под определенным индексом на передаваемое, конечно, если такой индекс существует.



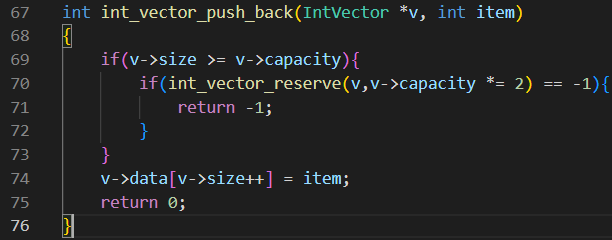
Следующие функции возвращают значения полей size и capacity соответственно.



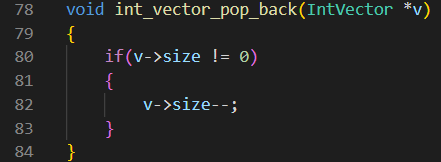
Функция int\_vector\_reserve увеличивает емкость нашего вектора и возвращает 0 в случае успеха. Если память выделить не удалось или новая емкость меньше предыдущей возвращает -1.



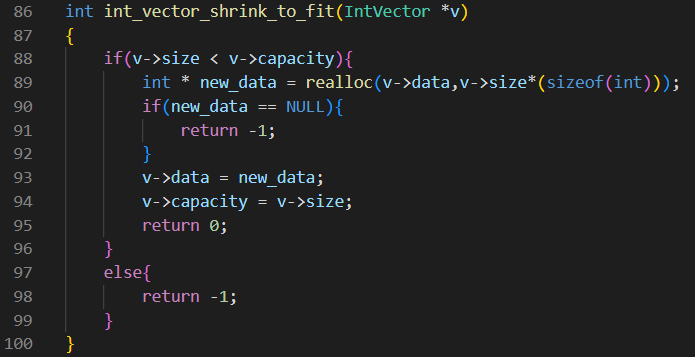
Функция int\_vector\_push\_back, записывает новый элемент в конец массива, в случае если нам не хватает размера, мы увеличиваем емкость в два раза. Возвращает 0 в случае успешной записи, и возвращает -1, если нам не удалось выделить память.



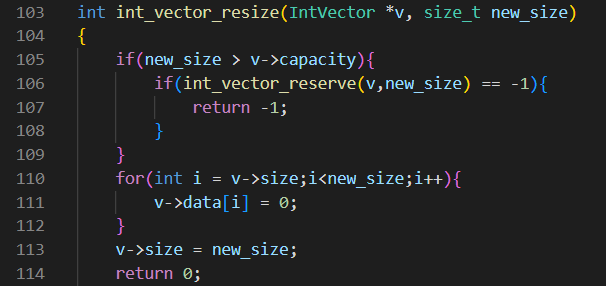
Функция int\_vector\_pop\_back удаляет последний элемент массива.



Функция int\_vector\_shrink\_to\_fit уменьшает емкость массива до его размера и возвращает 0 в случае успеха и -1 в случае ошибки.



Функция int\_vector\_resize изменяет размер массива и новые элементы инициализирует нулями, в случае если новый размер больше емкости, мы увеличиваем емкость в два раза. Возвращает 0 в случае успеха и -1 в случае ошибки.



# **ПРИЛОЖЕНИЕ**

<https://github.com/Herroa/proga2023>

**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 | #include <stdio.h>  #include "IntVector.h"  **void** **print**(IntVector \*array)  {  **for**(**int** i = **0**;i < array->capacity;i++){  printf("%d ", array->data[i]);  } printf("**\n**");  **for**(**int** i = **0**;i < array->size;i++){  printf("%d ", array->data[i]);  } printf("**\n**");  printf("=============================**\n**");  }  **int** **main**()  {  **size\_t** capacity = **15**;  IntVector \*array = int\_vector\_new(capacity);  **for**(**int** i = **0**;i < **8**;i++){  int\_vector\_push\_back(array,i);  } printf("**\n**");  print(array);  // int\_vector\_free(array);  // printf("%d\n", int\_vector\_get\_item(array,4));  // int\_vector\_set\_item(array, 5, 100);  // printf("%zu\n",int\_vector\_get\_size(array));  // printf("%zu\n",int\_vector\_get\_capacity(array));  printf("%d**\n**",int\_vector\_push\_back(array, **111**));  // int\_vector\_pop\_back(array);  // printf("%d\n",int\_vector\_shrink\_to\_fit(array));  // int\_vector\_resize(array, 3);  // printf("%d\n",int\_vector\_reserve(array, 3));  print(array);  // int\_vector\_free(array);  **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  24  25  26  27  28  29  30  31 | #include <stdlib.h>  **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 | #include "IntVector.h"  IntVector \***int\_vector\_new**(**size\_t** initial\_capacity)  {  IntVector \*v = NULL;  v = malloc(**sizeof**(\*v));  **if**(v == NULL){  **return** NULL;  }  v->data = malloc(initial\_capacity \* **sizeof**(**int**));  **if**(v->data == NULL)  {  free(v);  **return** NULL;  }  v->capacity = initial\_capacity;  v->size = **0**;  **return** v;  }  IntVector \***int\_vector\_copy**(**const** IntVector \*v)  {  IntVector \*v2 = int\_vector\_new(v->capacity);  **if**(v2 == NULL){  **return** NULL;  }    **for**(**int** i = **0**;i<v->size;i++){  v2->data[i] = v->data[i];}  v2->capacity = v->capacity;  v2->size = v->size;  **return** v2;  }  **void** **int\_vector\_free**(IntVector \*v)  {  free(v->data);  free(v);  }  **int** **int\_vector\_get\_item**(**const** IntVector \*v, **size\_t** index)  {  **if**(index < v->size){  **return** v->data[index];  }  }  **void** **int\_vector\_set\_item**(IntVector \*v, **size\_t** index, **int** item)  {  **if**(index < v->size){  v->data[index] = 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**(v->size >= v->capacity){  **if**(int\_vector\_reserve(v,v->capacity \*= **2**) == -**1**){  **return** -**1**;  }  }  v->data[v->size++] = item;  **return** **0**;  }  **void** **int\_vector\_pop\_back**(IntVector \*v)  {  **if**(v->size != **0**)  {  v->size--;  }  }  **int** **int\_vector\_shrink\_to\_fit**(IntVector \*v)  {  **if**(v->size < v->capacity){  **int** \* new\_data = realloc(v->data,v->size\*(**sizeof**(**int**)));  **if**(new\_data == NULL){  **return** -**1**;  }  v->data = new\_data;  v->capacity = v->size;  **return** **0**;  }  **else**{  **return** -**1**;  }  }  **int** **int\_vector\_resize**(IntVector \*v, **size\_t** new\_size)  {  **if**(new\_size > v->capacity){  **if**(int\_vector\_reserve(v,new\_size) == -**1**){  **return** -**1**;  }  }  **for**(**int** i = v->size;i<new\_size;i++){  v->data[i] = **0**;  }  v->size = new\_size;  **return** **0**;  }  **int** **int\_vector\_reserve**(IntVector \*v, **size\_t** new\_capacity)  {  **if**(new\_capacity > v->capacity){  **int** \* new\_data = realloc(v->data, new\_capacity\***sizeof**(**int**));  **if**(new\_data == NULL){  **return** -**1**;  }  v->data = new\_data;  v->capacity = new\_capacity;  **return** **0**;  }  **else**{  **return** -**1**;  } |

**Makefile**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **run:** main  ./main    **main:** main.o IntVector.o  gcc -Wall main.o IntVector.o -o main  main.o: main.c  gcc -Wall -c main.c  IntVector.o: IntVector.c  gcc -Wall -c IntVector.c  **clean:**  rm -rf \*.o main |