Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра вычислительных систем

**ОТЧЕТ**

по практической работе 3

по дисциплине «**Программирование**»

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. ИС-241  «17» марта 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Дмитрюк В.В./ |
|  |  |  |
| Проверил:  ст.преп. Кафедры ВС  «\_\_» марта 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Фульман В.О./ |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[ЗАДАНИЕ 3](#__RefHeading___Toc2367_3548133034)

[ВЫПОЛНЕНИЕ РАБОТЫ 4](#__RefHeading___Toc2371_3548133034)

[ПРИЛОЖЕНИЕ 9](#__RefHeading___Toc2369_3548133034)

# **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

Прототипы необходимых для реализации функций даны в требованиях к работе и были записаны в заголовочный файл IntVector.h. Функции были реализованы в файле IntVector.c.

Реализация функций происходила в порядке, отличном от того, в котором даны их прототипы – это позволяет использовать раннее реализованные функции в последующих.

Структура вектора выглядит следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5 | **typedef** **struct** intVector {  **size\_t** size; // длина массива, она же размер **size\_t** cap; // ёмкость массива  **int**\* arr; // массив чисел  } IntVector; |

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

Реализация: выделение памяти под структуру и массив целых чисел данной ёмкости с проверкой на нулевой указатель.

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

Реализация: создаёт новый вектор с ёмкостью, равной ёмкости данного вектора при помощи предыдущей функции, результат работы которой проверяется, и копирует массив целых чисел вместе со значением размера в новый вектор.

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

Реализация: освобождает память массива целых чисел, а затем – память всей структуры.

|  |  |
| --- | --- |
| 1 | **static** **void** **\_\_throw\_index\_OOB\_error**(**size\_t** index, **size\_t** size) |

Эта функция – следствие попытки решить задачу «Advanced». Однако, задача решена не полностью – функция не предоставляет возможности обработать ошибку в пользовательском коде, а аварийно завершает программу при попытке обратиться к индексу за пределами размера вектора.

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

Реализация: возвращает элемент массива целых чисел с запрошенным индексом, осуществляя проверку индекса на принадлежность пределу (0..size), вызывая предыдущую функцию при выходе за пределы.

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

Реализация: аналогично предыдущей функции, но вместо чтения осуществляется запись значения item.

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

Реализация: возвращает значение поля size.

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

Реализация: возвращает значение поля cap.

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

Реализация: при помощи функции realloc, возвращаемое значение которой проверяется, под массив целых чисел выделяется больше памяти, а значение ёмкости увеличивается. При попытке уменьшить размер массива этой функцией не происходит ничего.

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

Реализация: если новый размер не укладывается в ёмкость, вызывается предыдущая функция для её увеличения, после чего происходит инициализация новых элементов нулём и изменение значения размера в структуре.

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

Реализация: записывает в массив целых чисел элемент item по индексу size. Если такая запись приведёт к выходу за ёмкость массива, ёмкость увеличивается функцией **int\_vector\_reserve** до 2n+1, где n – старое значение ёмкости. Это отходит от рекомендуемого коэффициента роста и от реализации вектора в C++, поскольку позволяет успешно добавить элемент даже при нулевом размере.

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

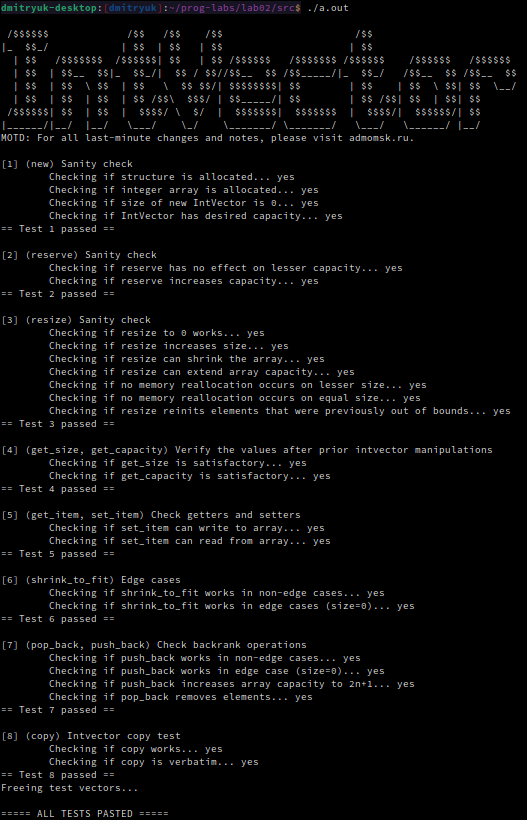
Реализация: записывает 0 в элемент под индексом size и декрементирует размер.

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

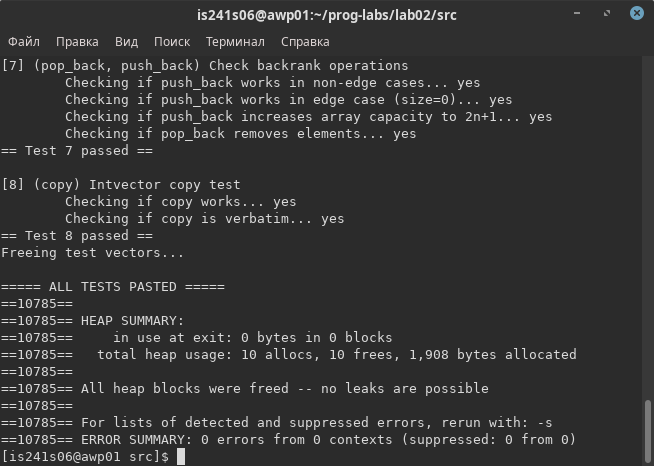
Реализация: при помощи realloc выделяет минимальное количество памяти для хранения количества целых чисел, соответствующего размеру. Значение возврата realloc проверяется. Если размер нулевой, память массива целых чисел освобождается. Если размер равен текущей ёмкости, не происходит ничего.

В ходе работы была реализована тестовая программа, проверяющая правильность работы всех реализованных функций. Значения после вызова функций сравниваются со значениями, известными на этапе компиляции и соответствующие ожидаемым. Программа выводит результаты тестов в стандартный поток ввода-вывода и уведомляет пользователя о прохождении всех тестов. Если какое-то из значений не соответствует ожидаемым, тест считается проваленным и работа тестовой программы прерывается.

Для удобства сборки тестовой программы был написан Makefile.



1. Снимок экрана после выполнения тестовой программы

2. Снимок экрана после выполнения valgrind

*Подводя итоги, можно утверждать, что наша реализация удволетворяет требованиям — реализована обработка ошибок выделения памяти, утечки памяти отсутствуют, граничные случаи проверяются тестовой программой.*

# **ПРИЛОЖЕНИЕ**

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  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 <stddef.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  **typedef** **struct** intVector {  **size\_t** size; // длина массива, она же размер  **size\_t** cap; // ёмкость массива  **int** \*arr; // массив чисел  } IntVector;  /\*  \* Создает массив нулевого размера.  \* Параметры: initial\_capacity (size\_t) – исходная емкость массива  \* Результат: указатель на IntVector, если удалось выделить память. Иначе  \* NULL.  \*/  IntVector \***int\_vector\_new**(**size\_t** initial\_capacity);  /\* Результат: Указатель на копию вектора v. NULL, если не удалось выделить  \* память. \*/  IntVector \***int\_vector\_copy**(**const** IntVector \*v);  /\* Освобождает память, выделенную для вектора v. \*/  **void** **int\_vector\_free**(IntVector \*v);  /\* Результат: элемент под номером index.  \* В случае выхода за границы массива поведение не определено. \*/  **int** **int\_vector\_get\_item**(**const** IntVector \*v, **size\_t** index);  /\* Присваивает элементу под номером index значение item.  \* В случае выхода за границы массива поведение не определено. \*/  **void** **int\_vector\_set\_item**(IntVector \*v, **size\_t** index, **int** item);  /\* Результат: размер вектора. \*/  **size\_t** **int\_vector\_get\_size**(**const** IntVector \*v);  /\* Результат: емкость вектора. \*/  **size\_t** **int\_vector\_get\_capacity**(**const** IntVector \*v);  /\*  \* Добавляет элемент в конец массива.  \* При необходимости увеличивает емкость массива.  \* Результат: 0 в случае успешного добавления элемента, -1 в случае ошибки.  \*/  **int** **int\_vector\_push\_back**(IntVector \*v, **int** item);  /\* Удаляет последний элемент из массива. Нет эффекта, если размер массива равен  \* 0. \*/  **void** **int\_vector\_pop\_back**(IntVector \*v);  /\* Уменьшает емкость массива до его размера.  \* Результат: 0 в случае успешного изменения емкости, -1 в случае ошибки. \*/  **int** **int\_vector\_shrink\_to\_fit**(IntVector \*v);  /\*  \* Изменяет размер массива.  \* Если новый размер массива больше исходного, то добавленные элементы  \* заполняются нулями. Результат: 0 в случае успеха, -1 в случае ошибки.  \* Если не удалось изменить размер, массив остается в исходном состоянии.  \*/  **int** **int\_vector\_resize**(IntVector \*v, **size\_t** new\_size);  /\* Изменить емкость массива.  \* Нет эффекта, если новая емкость меньше либо равна исходной.  \* Результат: 0 в случае успеха, -1 в случае ошибки.  \* Если не удалось изменить емкость, массив остается в исходном состоянии.  \*/  **int** **int\_vector\_reserve**(IntVector \*v, **size\_t** new\_capacity); |

IntVector.c

|  |  |
| --- | --- |
| 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  153  154  155  156  157  158  159  160  161  162  163 | #include "IntVector.h"  IntVector \***int\_vector\_new**(**size\_t** initial\_capacity) {  /\* Выделяем память под массив чисел и структуру.  \* Возвращает:  \* указатель на вектор. \*/  **int** \*arr = (**int** \*)malloc(**sizeof**(**int**) \* initial\_capacity);  IntVector \*iv = (IntVector \*)malloc(**sizeof**(IntVector));  **if** (arr == (**int** \*)NULL || iv == (IntVector \*)NULL)  **return** (IntVector \*)NULL;  /\* Если всё получилось,  \* ставим размер вектора и возвращаем адрес на структуру. \*/  iv->size = **0**;  iv->cap = initial\_capacity;  iv->arr = arr;  **return** iv;  }  IntVector \***int\_vector\_copy**(**const** IntVector \*v) {  /\* Выделяем память под массив чисел и стркутуру копии..  \* Возвращаем NULL, если не получилось. \*/  IntVector \*iv = int\_vector\_new(v->cap);  **if** (iv == NULL)  **return** (IntVector \*)NULL;  /\* Если всё получилось,  \* ставим те же длину и ёмкость, копируем содержимое исходного  \* вектора в копию и возвращаем адрес копии. \*/  memcpy(iv->arr, v->arr, v->size);  iv->cap = v->cap;  iv->size = v->size;  **return** iv;  }  **void** **int\_vector\_free**(IntVector \*v) {  /\* Освобождаем память, выделенную под массив числел  \* и под структуру. \*/  free(v->arr);  free(v);  }  **static** **void** **\_\_throw\_index\_OOB\_error**(**size\_t** index, **size\_t** size) {  /\* Выводим ошибку  \* и аварийно завершаем программу при выходе за границы \*/  // TODO: сделать вывод имени фунции, в которой произошла ошибка, вместе с её  // параметрами  printf("[IV] Guru meditaton.**\n\t**Attempted to access index (%zu) that is out "  "of bounds (max index is %zu).",  index, size - **1**);  exit(-**1**);  }  **int** **int\_vector\_get\_item**(**const** IntVector \*v, **size\_t** index) {  /\* Возвращаем элемент под номером index  \* Advanced: не даём обратиться к элементу за пределами массива \*/  **if** (index < v->size)  **return** v->arr[index];  \_\_throw\_index\_OOB\_error(index, v->size);  **return** **0**;  }  **void** **int\_vector\_set\_item**(IntVector \*v, **size\_t** index, **int** item) {  /\* Ставим элемент под номером index  \* Advanced: не даём обратиться к элементу за пределами массива \*/  **if** (index < v->size)  v->arr[index] = item;  **else**  \_\_throw\_index\_OOB\_error(index, v->size);  **return**;  }  **size\_t** **int\_vector\_get\_size**(**const** IntVector \*v) {  // Просто возвращаем размер  **return** v->size;  }  **size\_t** **int\_vector\_get\_capacity**(**const** IntVector \*v) {  // Просто возвращаем ёмкость  **return** v->cap;  }  **int** **int\_vector\_reserve**(IntVector \*v, **size\_t** new\_capacity) {  // Ёмкость не растёт в другую сторону  **if** (new\_capacity <= v->cap)  **return** **0**;  // Проверяем, сможем ли мы выделить больше памяти  // убился головой об стену когда 3 дня не мог отдебажить забытый множитель  // sizeof(int)  // int\* old\_arr = v->arr;  **int** \*newarr = (**int** \*)realloc(v->arr, new\_capacity \* **sizeof**(**int**));  **if** (!newarr)  **return** -**1**; // не можем :(  v->arr = newarr;  v->cap = new\_capacity;  **return** **0**;  }  **int** **int\_vector\_resize**(IntVector \*v, **size\_t** new\_size) {  // Размер МОЖЕТ расти в другую сторону  // но перевыделения памяти не происходит - просто уменьшаем размер  **if** (new\_size <= v->size) {  v->size = new\_size;  **return** **0**;  }  // Увеличиваем ёмкость, если новому размеру её не хватает  **if** (new\_size > v->cap)  **if** (int\_vector\_reserve(v, v->cap + (new\_size - (v->cap))))  **return** -**1**; // не получилось  // инициализация новых ячеек  **size\_t** old\_size = v->size;  v->size = new\_size;  **for** (; old\_size < v->size; old\_size++) {  int\_vector\_set\_item(v, old\_size, **0**);  }  **return** **0**;  }  **int** **int\_vector\_push\_back**(IntVector \*v, **int** item) {  // увеличиваем ёмкость, если нам её не хватает  **if** (int\_vector\_get\_size(v) == int\_vector\_get\_capacity(v)) {  ;  **if** (int\_vector\_reserve(v, v->cap \* **2** + **1**))  **return** -**1**;  }  v->arr[v->size] = item;  v->size += **1**;  **return** **0**;  }  **void** **int\_vector\_pop\_back**(IntVector \*v) {  // Нет эффекта, если размер равен нулю  **if** (v->size <= **0**)  **return**;  // Занулить последний элемент и уменьшить размер  int\_vector\_set\_item(v, v->size - **1**, **0**);  v->size--;  **return**;  }  **int** **int\_vector\_shrink\_to\_fit**(IntVector \*v) {  // Если размер и ёмкость равны, нет смысла что-то делать  **if** (v->size >= v->cap)  **return** **0**;  // Выделяем новую память  // если размер 0 просто освобождаем массив  **if** (v->size != **0**) {  **int** \*newarr = (**int** \*)realloc(v->arr, v->size \* **sizeof**(**int**));  **if** (!newarr)  **return** -**1**; // не получилось  v->arr = newarr;  v->cap = v->size;  } **else** {  free(v->arr);  v->arr = NULL;  }  **return** **0**;  } |

main.c (тестовая программа; удалена заглавная надпись в начале)

|  |  |
| --- | --- |
| 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 | #include "IntVector.h"  **static** **void** **testCase**(**int** test) {  **if** (test) {  printf("no**\n**");  exit(**0**);  } **else**  printf("yes**\n**");  }  **int** **main**(**void**) {  printf("**\n**[1] (new) Sanity check**\n**");  IntVector \*testVec = int\_vector\_new(**8**);  printf("**\t**Checking if structure is allocated... ");  testCase(!testVec);  printf("**\t**Checking if integer array is allocated... ");  testCase(!testVec->arr);  printf("**\t**Checking if size of new IntVector is 0... ");  testCase(testVec->size != **0**);  printf("**\t**Checking if IntVector has desired capacity... ");  testCase(testVec->cap != (**size\_t**)**8**);  printf("== Test 1 passed == **\n\n**");  printf("[2] (reserve) Sanity check**\n**");  printf("**\t**Checking if reserve has no effect on lesser capacity... ");  testCase(!int\_vector\_reserve(testVec, **4**) && testVec->cap != **8**);  printf("**\t**Checking if reserve increases capacity... ");  int\_vector\_reserve(testVec, **16**);  testCase(testVec->cap != **16**);  printf("== Test 2 passed == **\n\n**");  printf("[3] (resize) Sanity check**\n**");  printf("**\t**Checking if resize to 0 works... ");  int\_vector\_resize(testVec, **0**);  testCase(testVec->size != **0**);  int\_vector\_resize(testVec, **8**);  printf("**\t**Checking if resize increases size... ");  testCase(testVec->size != **8**);  printf("**\t**Checking if resize can shrink the array... ");  int\_vector\_resize(testVec, **4**);  testCase(testVec->size != **4**);  printf("**\t**Checking if resize can extend array capacity... ");  **size\_t** \_newCap = (testVec->cap) + **1**;  // size\_t \_oldCap = (testVec->cap);  int\_vector\_resize(testVec, \_newCap);  testCase(testVec->size != testVec->cap || testVec->cap != \_newCap);  printf("**\t**Checking if no memory reallocation occurs on lesser size... ");  **int** \*old\_addr = testVec->arr;  int\_vector\_resize(testVec, **2**);  testCase(old\_addr != testVec->arr);  printf("**\t**Checking if no memory reallocation occurs on equal size... ");  old\_addr = testVec->arr;  int\_vector\_resize(testVec, **2**);  testCase(old\_addr != testVec->arr);  printf("**\t**Checking if resize reinits elements that were previously "  "out of bounds... ");  **size\_t** oldSize = testVec->size;  int\_vector\_resize(testVec, **8**);  **int** flag = **0**;  **for** (**size\_t** i = oldSize; i < testVec->size; i++)  **if** (testVec->arr[i] != **0**)  flag = **1**;  testCase(flag == **1**);  printf("== Test 3 passed ==**\n\n**");  printf("[4] (get\_size, get\_capacity) Verify the values after prior"  "intvector manipulations**\n**");  printf("**\t**Checking if get\_size is satisfactory... ");  **size\_t** \_size = int\_vector\_get\_size(testVec);  testCase(\_size != **8**);  printf("**\t**Checking if get\_capacity is satisfactory... ");  **size\_t** \_cap = int\_vector\_get\_capacity(testVec);  testCase(\_cap != **17**);  printf("== Test 4 passed ==**\n\n**");  printf("[5] (get\_item, set\_item) Check getters and setters**\n**");  printf("**\t**Checking if set\_item can write to array... ");  **for** (**size\_t** i = **0**; i < testVec->size; i++)  int\_vector\_set\_item(testVec, i, (**int**)i + **90**);  flag = **0**;  **for** (**size\_t** i = **0**; i < testVec->size; i++)  **if** (testVec->arr[i] != (**int**)(i + **90**))  flag = **1**;  testCase(flag);  printf("**\t**Checking if set\_item can read from array... ");  flag = **0**;  **for** (**size\_t** i = **0**; i < testVec->size; i++)  **if** (int\_vector\_get\_item(testVec, i) != (**int**)(i + **90**))  flag = **1**;  testCase(flag);  printf("== Test 5 passed ==**\n\n**");  printf("[6] (shrink\_to\_fit) Edge cases **\n**");  printf("**\t**Checking if shrink\_to\_fit works in non-edge cases... ");  int\_vector\_resize(testVec, **6**);  int\_vector\_shrink\_to\_fit(testVec);  testCase(int\_vector\_get\_capacity(testVec) != **6**);  printf("**\t**Checking if shrink\_to\_fit works in edge cases (size=0)... ");  int\_vector\_resize(testVec, **0**);  int\_vector\_shrink\_to\_fit(testVec);  testCase(testVec->arr != NULL);  printf("== Test 6 passed ==**\n\n**");  printf("[7] (pop\_back, push\_back) Check backrank operations**\n**");  int\_vector\_reserve(testVec, **32**);  int\_vector\_resize(testVec, **31**);  printf("**\t**Checking if push\_back works in non-edge cases... ");  int\_vector\_push\_back(testVec, **25565**);  testCase(int\_vector\_get\_item(testVec, **31**) != **25565**);  printf("**\t**Checking if push\_back works in edge case (size=0)... ");  int\_vector\_resize(testVec, **0**);  int\_vector\_push\_back(testVec, **1000** - **7**);  testCase(int\_vector\_get\_item(testVec, **0**) != **993**);  printf("**\t**Checking if push\_back increases array capacity to 2n+1... ");  int\_vector\_resize(testVec, **32**);  int\_vector\_push\_back(testVec, **11037**);  testCase(int\_vector\_get\_capacity(testVec) != **65** &&  int\_vector\_get\_item(testVec, **32**) != **11037** &&  int\_vector\_get\_size(testVec) != **33**);  printf("**\t**Checking if pop\_back removes elements... ");  int\_vector\_pop\_back(testVec);  testCase(int\_vector\_get\_capacity(testVec) != **65** &&  int\_vector\_get\_item(testVec, **31**) != **25565** &&  int\_vector\_get\_size(testVec) != **32**);  printf("== Test 7 passed ==**\n\n**");  printf("[8] (copy) Intvector copy test**\n**");  IntVector \*testVec2;  printf("**\t**Checking if copy works... ");  testVec2 = int\_vector\_copy(testVec);  testCase(testVec2 == NULL);  printf("**\t**Checking if copy is verbatim... ");  testCase(testVec2->cap != testVec->cap || testVec2->size != testVec->size ||  memcmp(testVec->arr, testVec2->arr, testVec->size));  printf("== Test 8 passed ==**\n**");  printf("Freeing test vectors... **\n**");  int\_vector\_free(testVec);  int\_vector\_free(testVec2);  printf("**\n**===== ALL TESTS PASTED =====**\n**");  **return** **0**;  } |