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  «25» февраля 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Тынянов В.С./ |
|  |  |  |
| Проверил:  Старший преподаватель Кафедры ВС  «27» февраля 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Фульман В.О./ |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[**ЗАДАНИЕ** 3](#_Toc35593781)

[**ВЫПОЛНЕНИЕ РАБОТЫ** 4](#_Toc35593782)

[**ПРИЛОЖЕНИЕ** 8](#_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**

Начнем с создания структуры IntVector, имеющей три поля: емкость, размер и указатель на массив – в файле IntVector.h.

|  |  |
| --- | --- |
| 1  2  3  4  5 | typedef struct {  int\* data;  size\_t size;  size\_t capacity;  } IntVector; |

Функция int\_vector\_new, принимающей в качестве аргумента емкость, а возвращающая указатель на созданный вектор. Делаем проверку на то, что память для вектора и для массива была выделена, иначе возвращаем NULL при этом высвобождая память.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | IntVector \*vector = (IntVector\*)malloc(**sizeof**(IntVector));  **if** (!vector) {  **return** NULL;  }  **int** \*data = (**int**\*) malloc(**sizeof**(**int**) \* initial\_capacity);  **if** (!data) {  free(vector);  **return** NULL;  } |

Затем сделаем функцию int\_vector\_copy, принимающую указатель на вектор и возвращающую указатель на копию входного вектора, с проверкой на выделение памяти.

Копируем содержимое входного вектора и присваиваем новому вектору

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **for** (**int** i = **0**; i < v->size; i++) {  copy\_data[i] = (v->data)[i];  }  copy\_vector->data = copy\_data;  copy\_vector->size = v->size;  copy\_vector->capacity = v->capacity;  **return** copy\_vector; |

Напишем функцию int\_vector\_free освобождающую память, выделенную под вектор.

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

Так же напишем функции int\_vector\_get\_size и int\_vector\_get\_capacity, которые получают указатель на вектор и возвращают размер и емкость вектора соответственно.

Функции int\_vector\_get\_item и int\_vector\_set\_item получают и кладут элемент по индексу соответственно. Если индекс вышел за пределы массива, поведение не определенно.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **int** **int\_vector\_get\_item**(**const** IntVector \*v, **size\_t** index)  {  **int** item = (v->data)[index];  **return** item;  }  **void** **int\_vector\_set\_item**(IntVector \*v, **size\_t** index, **int** item)  {  (v->data)[index] = item;  } |

Функция int\_vector\_reserve получает на вход вектор и новую емкость и увеличивает емкость до необходимой, в случае успеха возвращает 0, в обратном -1. Если емкость меньше или равно исходной, то ничего не происходит.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **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) {  **return** -**1**;  }  v->data = new\_data;  v->capacity = new\_capacity;  }  **return** **0**;  } |

Функция int\_vector\_pop\_back принимает на вход указатель на вектор и уменьшает размер вектора на 1.

Функция int\_vector\_resize изменяет размер вектора, при необходимости увеличиваю емкость, если новый размер больше исходного новые ячейки заполняются нулями, в случае успеха возвращает 0, в обратном -1.

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

Функция int\_vector\_shrink\_to\_fit принимает на вход указатель на вектор и уменьшает емкость этого вектора до его размера, в случае успеха возвращает 0, в обратном -1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | int int\_vector\_shrink\_to\_fit(IntVector \*v)  {  int\* new\_data = realloc(v->data, v->size \* sizeof(int));  if (!new\_data) {  return -1;  }  v->data = new\_data;  v->capacity = v->size;  return 0;  } |

Последняя функция int\_victor\_push\_back получает указатель на вектор и значение, которое будет записано в конец вектора, при необходимости, емкость вектора увеличится на 2 при помощи функции int\_vector\_reserve, в случае успеха возвращает 0, в обратном -1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **int** **int\_vector\_push\_back**(IntVector \*v, **int** item)  {  **size\_t** index = int\_vector\_get\_size(v);  **size\_t** capacity = int\_vector\_get\_capacity(v);  **if** (index == capacity) {  **if** (int\_vector\_reserve(v, capacity + **2**)) {  **return** -**1**;  }  }  (v->data)[index] = item;  (v->size)++;  **return** **0**;  } |

В файле main.c сделаем проверку всех функций.

Создадим Makefile и напишим инструкцию для компиляции main.c.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **all**: **main**  **main.o**: **src/main.c**  gcc -c -Wall -Werror -o main.o src/main.c  **IntVector.o**: **src/IntVector.c**  gcc -c -Wall -Werror -o IntVector.o src/IntVector.c  **main**: **IntVector.o main.o**  gcc -Wall -Werror -o main IntVector.o main.o |

Скомпилируем используя команду make и при помощи утилиты valgrind с флагом –leak-cheak=yes проверим нашу программу на факт утечки памяти.

|  |  |
| --- | --- |
| с | vadim@LAPTOP-5NUPNVPD:lab2/$ valgrind --leak-check=yes ./main |

Утечек памяти не обнаружено.

|  |  |
| --- | --- |
| 1  2  3  4  5 | ==12607== HEAP SUMMARY:  ==12607== in use at exit: 0 bytes in 0 blocks  ==12607== total heap usage: 12 allocs, 12 frees, 1,328 bytes allocated  ==12607==  ==12607== All heap blocks were freed -- no leaks are possible |

# **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 | #include <stdio.h>  #include "IntVector.h"  **void** **pritn\_vector**(IntVector \*v) {  **for** (**int** i = **0**; i < v->size; i++) {  printf("%d ", int\_vector\_get\_item(v, i));  }  printf("**\n**");  printf("size-%ld, capacity-%ld**\n**", v->size, v->capacity);  }  **int** **main**()  {  IntVector \*vector1 = int\_vector\_new(**0**);  **if** (!vector1) {  **return** -**1**;  }  printf("vector1 capacity=0**\n**");  pritn\_vector(vector1);  **for** (**int** i = **0**; i < **10**; i++) {  int\_vector\_push\_back(vector1, i);  }  printf("vector1 add 10 itmes**\n**");  pritn\_vector(vector1);  IntVector \*vector2 = int\_vector\_copy(vector1);  **if** (!vector2) {  **return** -**1**;  }  printf("vector2 is copy vector1**\n**");  pritn\_vector(vector2);  printf("vector2 pop and shrink ot fit**\n**");  int\_vector\_pop\_back(vector2);  int\_vector\_shrink\_to\_fit(vector2);  pritn\_vector(vector2);  printf("Set item index = 3 value = 5**\n**");  int\_vector\_set\_item(vector2, **3**, **5**);  pritn\_vector(vector2);  printf("vector1 resize to 5**\n**");  int\_vector\_resize(vector1, **5**);  pritn\_vector(vector1);  printf("vector2 resize to 15**\n**");  int\_vector\_resize(vector2, **15**);  pritn\_vector(vector2);  int\_vector\_free(vector1);  int\_vector\_free(vector2);  **return** **0**;  } | |

**IntVector.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | **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\_reserve**(IntVector \*v, **size\_t** new\_capacity);  **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); |

**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 | #include <stdlib.h>  #include "IntVector.h"  IntVector \***int\_vector\_new**(**size\_t** initial\_capacity)  {  IntVector \*vector = (IntVector\*)malloc(**sizeof**(IntVector));  **if** (!vector) {  **return** NULL;  }  **int** \*data = (**int**\*) malloc(**sizeof**(**int**) \* initial\_capacity);  **if** (!data) {  free(vector);  **return** NULL;  }  vector->data = data;  **return** vector;  }  IntVector \***int\_vector\_copy**(**const** IntVector \*v)  {  IntVector \*copy\_vector = (IntVector\*) malloc(**sizeof**(IntVector));  **if** (!copy\_vector) {  **return** NULL;  }  **int** \*copy\_data = (**int**\*) malloc(**sizeof**(**int**) \* v->capacity);  **if** (!copy\_data) {  free(copy\_vector);  **return** NULL;  }  **for** (**int** i = **0**; i < v->size; i++) {  copy\_data[i] = (v->data)[i];  }  copy\_vector->data = copy\_data;  copy\_vector->size = v->size;  copy\_vector->capacity = v->capacity;  **return** copy\_vector;  }  **void** **int\_vector\_free**(IntVector \*v)  {  free(v->data);  free(v);  }  **int** **int\_vector\_get\_item**(**const** IntVector \*v, **size\_t** index)  {  **int** item = (v->data)[index];  **return** item;  }  **void** **int\_vector\_set\_item**(IntVector \*v, **size\_t** index, **int** item)  {  (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)  {  **size\_t** index = int\_vector\_get\_size(v);  **size\_t** capacity = int\_vector\_get\_capacity(v);  **if** (index == capacity) {  **if** (int\_vector\_reserve(v, capacity + **2**)) {  **return** -**1**;  }  }  (v->data)[index] = item;  (v->size)++;  **return** **0**;  }  **void** **int\_vector\_pop\_back**(IntVector \*v)  {  **if** (v->size > **0**) {  v->size--;  }  }  **int** **int\_vector\_shrink\_to\_fit**(IntVector \*v)  {  **int**\* new\_data = realloc(v->data, v->size \* **sizeof**(**int**));  **if** (!new\_data) {  **return** -**1**;  }  v->data = new\_data;  v->capacity = v->size;  **return** **0**;  }  **int** **int\_vector\_resize**(IntVector \*v, **size\_t** new\_size)  {  **if** (v->capacity < new\_size) {  **if** (int\_vector\_reserve(v, new\_size)) {  **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) {  **return** -**1**;  }  v->data = new\_data;  v->capacity = new\_capacity;  }  **return** **0**;  } | |