Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Кафедра вычислительных систем

ОТЧЕТ

по практической работе 1 по дисциплине
«Программирование»

Выполнил:
студент гр. ИС - 242
«16» февраля 2023 г _____ Денисов М. А.

Проверил:
Старший преподаватель
Кафедры ВС
«16» февраля 2023 г. _____ Фульман В.О.

Оценка **«**_____**»**

Новосибирск 2023

ОГЛАВЛЕНИЕ

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

Функция `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.

```
{
    IntVector *v = NULL;
    v = malloc(sizeof(*v));
    if (!v)
    {
        return NULL;
    }
```

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

```
    v->data = malloc(initial_capacity*sizeof(int));
    if (!v->data)
    {
        free(v);
        return NULL;
    }
```

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.

```
20      if (!v->data)
21      {
22          free(v);
23          return NULL;
24      }
25      v->capacity = initial_capacity;
26      v->size = 0;
27      return v;
28
```

Проверим, как работает функция

```
IntVector array
 data = 0x56049e1bb280
 size = 0
 capacity = 3
```

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

2. 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.

```
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;
}
```

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

```
IntVector array
 data = 0x55fb4f51e280
 size = 0
 capacity = 3
0
1
2
IntVector array
 data = 0x55fb4f51e280
 size = 3
 capacity = 3
IntVector a
 data = 0x55fb4f51e6d0
 size = 3
 capacity = 3
0
1
2
```

Функция работает правильно, и полностью копирует подаваемый вектор.

3. Функция

```
void int_vector_free(IntVector *v)
```

Освобождает выделенную память.

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

```
48  }
49  void int_vector_free(IntVector *v)
50  {
51      free(v->data);
52      free(v);
53  }
```

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.

```
vector a
0x557f794f26b0
0
0
2035228688
```

4. 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.

```
53  }
54  int int_vector_get_item(const IntVector *v, size_t index)
55  {
56      int s = v->data[index];
57      return s;
58  }
```

Попробуем взять 1 индекс элемента, должно вывести один.

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

```
Get Item
1
IntVector array
 data = 0x562024d5a280
 size = 3
 capacity = 3
0
1
```

5. 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.

```
IntVector array
 data = 0x559ae520d280
 size = 3
 capacity = 3
0
1
7
```

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

6. Функция

```
size_t int_vector_get_size(const IntVector *v)
```

Размер вектора.

Возвращаем значение size в поданном указатели на структуру IntVector.

7. 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.

```
IntVector array
 data = 0x56474a890280
 size = 3
 capacity = 3
0
1
7
size array = 3
capacity array = 3
```

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

8. 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.

```c
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*2) * sizeof(int));
        if (!z)
        {
            return -1;
        }
        v->capacity = v->capacity * 2;
        v -> data = z;
        v -> data[v->size] = item;
        v -> size++;
    }
    return 0;
```

Добавим элемент 10 в конец, с помощью функции push_back и выведем значения на экран.

```
IntVector array
 data = 0x563488706280
 size = 3
 capacity = 3
0
1
7
10
IntVector array
 data = 0x563488706280
 size = 4
 capacity = 6
```

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.

9. Функция

```c
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.

```c
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;
    }
}
```

Вызовем функцию и выведем значения на экран.

```
IntVector array
 data = 0x56498209460d0
 size = 3
 capacity = 6
0
1
7
```

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

10. Функция

```c
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.

```c
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;
```

Вызовем функция и выведем значения структуры и массива на экран.

```
IntVector array
 data = 0x56498209 46d0
 size = 3
 capacity = 6
0
1
7
IntVector array
 data = 0x56498209 46d0
 size = 3
 capacity = 3
0
1
7
```

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

11. 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.

```c
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;
}
```

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

```
IntVector array
 data = 0x55b592d03710
 size = 3
 capacity = 40
0
1
7
IntVector array
 data = 0x55b592d03710
 size = 10
 capacity = 40
0
1
7
0
0
0
0
0
0
0
```

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

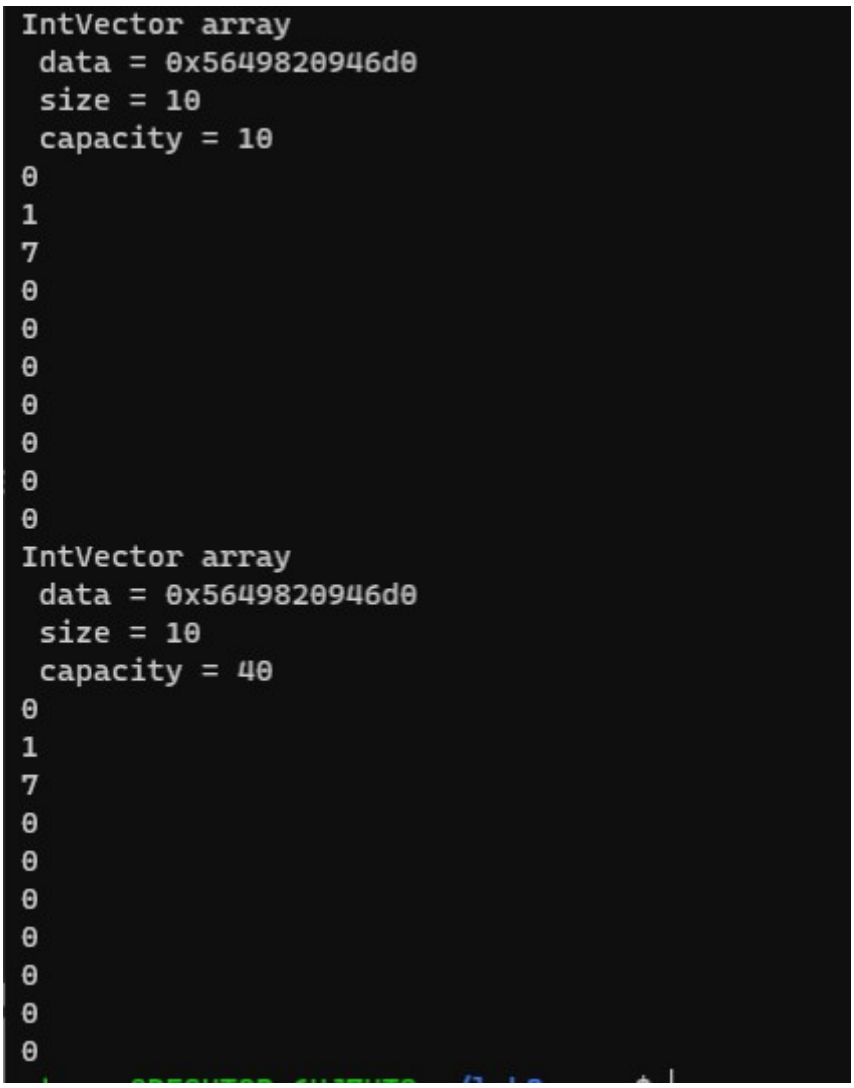
12. Функция `int int_vector_reserve(IntVector *v, size_t new_capacity)` Изменить емкость массива.

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.

```
151 int int_vector_reserve(IntVector *v, size_t new_capacity)
152 {
153     if (new_capacity > v->capacity)
154     {
155         int *z = realloc(v->data,new_capacity*sizeof(int));
156         if (!z)
157         {
158             return -1;
159         }
160         v->capacity = new_capacity;
161         v->data = z;
162         return 0;
163     }
164     else
165     {
166         return -1;
167     }
168 }
```

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   IntVector *int_vector_new(size_t initial_capacity)
 2   {
 3       IntVector *v = NULL;
 4       v = malloc(sizeof(*v));
 5       if (!v)
 6       {
 7           return NULL;
 8        }
 9       v->data = malloc(initial_capacity*sizeof(int));
10       if (!v->data)
11       {
12           free(v);
13           return NULL;
14        }
15       v->capacity = initial_capacity;
16       v->size = 0;
17       return v;
```

2. IntVector *int_vector_copy(const IntVector *v)

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

3. `void int_vector_free(IntVector *v)`

```
1   void int_vector_free(IntVector *v)
2   {
3       free(v->data);
4       free(v);
5   }
```

4. `int int_vector_get_item(const IntVector *v, size_t index)`

```
1   int int_vector_get_item(const IntVector *v, size_t index)
2   {
3       int s = v->data[index];
4       return s;
5   }
```

5. `void int_vector_set_item(IntVector *v, size_t index, int item)`

```
1   void int_vector_set_item(IntVector *v, size_t index, int item)
2   {
3       v->data[index] = item;
4   }
```

6. `size_t int_vector_get_size(const IntVector *v)`

```
1   size_t int_vector_get_size(const IntVector *v)
2   {
3       return v->size;
4   }
```

7. `size_t int_vector_get_capacity(const IntVector *v)`

```
1   size_t int_vector_get_capacity(const IntVector *v)
2   {
3       return v->capacity;
4   }
```

8. `int int_vector_push_back(IntVector *v, int item)`

```
1   int int_vector_push_back(IntVector *v, int item)
2   {
3       if (v->size < v->capacity)
4       {
5           v->data[v->size] = item;
6           v->size++;
7       }
8       else
9       {
10          int *z = realloc(v->data,(v->capacity*2) * sizeof(int));
11          if (!z)
12          {
```

```
13              return -1;
14           }
15          v->capacity = v->capacity * 2;
16          v -> data = z;
17          v -> data[v->size] = item;
18          v -> size++;
19      }
20      return 0;
21  }
```

9. `void int_vector_pop_back(IntVector *v)`

```
1   void int_vector_pop_back(IntVector *v)
2   {
3       if (v->size > 0)
4       {
5           v->size = s;
6        }
```

10. `int int_vector_shrink_to_fit(IntVector *v)`

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

11. `int int_vector_resize(IntVector *v, size_t new_size)`

```c
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;
}
```

13. `int int_vector_reserve(IntVector *v, size_t new_capacity)`

```c
int int_vector_reserve(IntVector *v, size_t new_capacity)
{
    if (new_capacity > v->capacity)
    {
        int *z = realloc(v->data,new_capacity*sizeof(int));
        if (!z)
        {
            return -1;
         }
        v->capacity = new_capacity;
        v->data = z;
        return 0;
     }
    else
    {
        return -1;
     }
}
```

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "IntVector.h"
int main()
{
    IntVector *array = int_vector_new(3);
    print_vector(array);
    for (int i = 0; i < array->capacity; i++)
```

```
10        {
11            int_vector_push_back(array,i);
12            printf("%d\n",array -> data[i]);
13         }
14        print_vector(array);
15        IntVector *a = int_vector_copy(array);
16        print_vector(array);
17        printf("vector a\n");
18        printf("%p\n",a);
19        for (int i = 0; i < array->capacity; i++) {
20                    printf("%d\n", a->data[i]);
21        }
22        printf("\n");
23        printf("Get Item\n");
24        int z;
25        z = int_vector_get_item(array,1);
26        printf("%d\n",z);
27        print_vector(array);
28        printf("set item\n");
29        int_vector_set_item(array,2,7);
30        for (int i = 0; i < array->capacity; i++) {
31                    printf("%d\n", array->data[i]);
32        }
33        size_t s = int_vector_get_size(array);
34        size_t jos = int_vector_get_capacity(array);
35        printf("size array = %ld\n",s);
36        printf("capacity array = %ld\n",jos);
37        printf("\n");
38        print_vector(array);
39        printf("push back\n");
40        int_vector_push_back(array,10);
41        print_vector(array);
42        printf("\n");
43        printf("pop back\n");
44        int_vector_pop_back(array);
45        print_vector(array);
46        printf("shrink to fit\n");
47        int_vector_shrink_to_fit(array);
48        print_vector(array);
49        printf("reserve\n");
50        int_vector_reserve(array,40);
51        print_vector(array);
52        printf("resize\n");
53        int_vector_resize(array,10);
54        print_vector(array);
55        int_vector_free(array);
56        int_vector_free(a);
57        return 0;
58    }
```

intvector.h

```c
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);
void print_vector(IntVector *v);
```