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: Гердележов Д.Д.
студент гр. ИВ-122
«3» апреля 2022 г.

Проверил: Фульман В.О.
Старший преподаватель
кафедры ВС.
«6» апреля 2022 г.

Оценка «_____»

Новосибирск 2022

Оглавление:

Задание:

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

Базовые операции с вектором:

IntVector *int_vector_new(size_t initial_capacity)

Создает массив нулевого размера.

Параметры:	**initial_capacity** (*size_t*) – исходная емкость массива
Результат:	указатель на **IntVector**, если удалось выделить память. Иначе **NULL**.

Implementation note: поскольку функция возвращает указатель, в реализации ожидается выделение двух участков памяти: для структуры **IntVector** и для массива внутри структуры. Функция должна корректно обрабатывать ошибку при выделении любого из участков памяти, не должна возвращать указатель частично сформированный объект и не должна приводить к утечкам памяти в случае ошибки.

IntVector *int_vector_copy(const IntVector *v)

Результат:	Указатель на копию вектора **v**. **NULL**, если не удалось выделить память.

void int_vector_free(IntVector *v)

Освобождает память, выделенную для вектора **v**.

int int_vector_get_item(const IntVector *v, size_t index)

Результат:	элемент под номером **index**. В случае выхода за границы массива поведение не определено.

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

Присваивает элементу под номером **index** значение **item**. В случае выхода за границы массива поведение не определено.

size_t int_vector_get_size(const IntVector *v)

Результат:	размер вектора.

size_t int_vector_get_capacity(const IntVector *v)

Результат:	емкость вектора.

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.

Результат: 0 в случае успешного добавления элемента, -1 в случае ошибки.

void int_vector_pop_back(IntVector *v)

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

int int_vector_shrink_to_fit(IntVector *v)

Уменьшает емкость массива до его размера.

Результат: 0 в случае успешного изменения емкости, -1 в случае ошибки.

int int_vector_resize(IntVector *v, size_t new_size)

Изменяет размер массива.

Если новый размер массива больше исходного, то добавленные элементы заполняются нулями.

Если новый размер массива меньше исходного, то перевыделение памяти не происходит. Для уменьшения емкости массива в этом случае следует использовать функцию **int_vector_shrink_to_fit**.

Результат: 0 в случае успеха, -1 в случае ошибки. Если не удалось изменить размер, массив остается в исходном состоянии.

int int_vector_reserve(IntVector *v, size_t new_capacity)

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.

Выполнение работы:

Структура проекта:

```
.
|-- Makefile
`-- src
    |-- IntVector.c
    |-- IntVector.h
    `-- main.c
```

Содержимое Makefile:

```
all:
    gcc -Wall -o main src/*.c -g
```

Файл IntVector.h содержит структуру IntVector и прототипы всех функций (приложение 1).

В файле IntVector.c (приложение 2) содержатся описания функций
1)

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

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

    v->size = 0;
    v->capacity = initial_capacity;
    return v;
}
```

Данная функция создает массив размером 0 с емкостью initial_capacity и возвращает указатель на IntVector (NULL, если выделить память не удалось)

2)

```
IntVector* int_vector_copy(const IntVector* v)
{
    IntVector* t = malloc(sizeof(IntVector));
    if (t == NULL)
        return NULL;

    t->data = malloc(v->capacity * sizeof(int));
    if (t->data == NULL) {
        free(t);
        return NULL;
    }

    memcpy(t->data, v->data, sizeof(int) * v->capacity);
```

```
14      t->size = v->size;
15      t->capacity = v->capacity;
16      return t;
17  }
```

Данная функция создаёт копию вектора v и возвращает указатель на неё (**NULL**, если не удалось выделить память)

3)

```
1  void int_vector_free(IntVector* v)
2  {
3      if (!v) {
4          return;
5      }
6      free(v->data);
7      free(v);
8  }
```

Данная функция освобождает память, выделенную для вектора **v**.

4)

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

Данная функция элемент под номером **index**.

5)

```
1  void int_vector_set_item(IntVector* v, size_t index, int item)
2  {
3      if (index <= v->capacity)
4          v->data[index] = item;
5  }
```

Данная функция присваивает элементу под номером **index** значение **item**.

6)

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

Данная функция возвращает размер вектора v.

7)

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

Данная функция возвращает ёмкость вектора v.

8)

```
int int_vector_push_back(IntVector* v, int item)
{
    if (v->size < v->capacity) {
        v->data[v->size] = item;
        v->size++;
    } else {
        v->capacity = v->capacity * 2;
        v->data = realloc(v->data, v->capacity * sizeof(int));

        v->data[v->size] = item;
        v->size++;
    }
    return 0;
}
```

Данная функция добавляет элемент в конец массива. При необходимости увеличивает емкость массива в 2 раза.

9)

```
void int_vector_pop_back(IntVector* v)
{
    if (v->size != 0) {
        v->size--;
    }
}
```

Данная функция удаляет последний элемент массива.

10)

```
int int_vector_shrink_to_fit(IntVector* v)
{
    if (v->size < v->capacity) {
        v->capacity = v->size;
        int* t = realloc(v->data, v->size * sizeof(int));
        if (t == NULL)
            return -1;
        v->data = t;
        return 0;
    }
    return -1;
}
```

Данная функция уменьшает емкость массива до его размера.

11)

```
int int_vector_resize(IntVector* v, size_t new_size)
{
    if (new_size == v->size)
        return 0;
    if (new_size > v->size) {
        int* t = realloc(v->data, new_size * sizeof(int));
```

```
 7         if (t == NULL)
 8             return -1;
 9         v->data = t;
10         for (int i = new_size - v->size; i < new_size; i++)
11             v->data[i] = 0;
12     }
13     v->size = new_size;
14     v->capacity = new_size;
15     return 0;
16 }
```

Данная функция изменяет размер массива.

Если новый размер массива больше исходного, то добавленные элементы заполняются нулями.

Если новый размер массива меньше исходного, то перевыделение памяти не происходит.

12)

```
 1 int int_vector_reserve(IntVector* v, size_t new_capacity)
 2 {
 3     if (new_capacity > v->capacity) {
 4         v->capacity = new_capacity;
 5         int* t = realloc(v->data, new_capacity * sizeof(int));
 6         if (t == NULL)
 7             return -1;
 8         v->data = t;
 9         return 0;
10     }
11     return -1;
12 }
```

Данная функция изменяет емкость массива.

Нет эффекта, если новая емкость меньше либо равна исходной.

Файл main.c (приложение 3) содержит как минимум один пример использования каждой из данных функций.

В результате компиляции и запуска программы будет выведено следующее:

```
nothomepc@DESKTOP-BFFKO6N:~/prog/Study/Laby/2$ ./app
Создаем массив v ёмкостью 7 с помощью функции int_vector_new:
vector v:
data: 0x55f37369c6d0
size: 0
capacity: 7
-----
Заполняем массив, используя функцию int_vector_get_capacity что бы узнать его ёмкость
и функцию int_vector_push_back что бы добавить элемент в конец массива.
Выводим массив, используя функцию int_vector_get_size что бы узнать его размер
и функцию int_vector_get_item что бы получать элементы массива.
0 = 34  1 = 37  2 = 28  3 = 16  4 = 44  5 = 36  6 = 37
-----
```

```
Изменим третий элемент массива на 999 с помощью фунции int_vector_set_item
и добавим в последниий элемент массива 15 с помощью функции int_vector_push_back:
vector v:
data: 0x55f37369c6d0
size: 8
capacity: 14
0 = 34  1 = 37  2 = 999 3 = 16  4 = 44  5 = 36  6 = 37  7 = 15
Так как изначальной ёмкости (7) не хватило для добавления нового элемента
она удвоилась
-----
Попробуем уменьшить емкость ёмкость массива до 5 с помощью функции int_vector_reserve
vector v:
data: 0x55f37369c6d0
size: 8
capacity: 14
Так как ёмкость меньше исходной ничего не произошло.
-----
Увеличим ёмкость массива до 15 с помощью функции int_vector_reserve
vector v:
data: 0x55f37369c6d0
size: 8
capacity: 15
-----
Уменьшим ёмкость массива до его размера с помощью функции int_vector_shrink_to_fit
vector v:
data: 0x55f37369c6d0
size: 8
capacity: 8
-----
Увеличим ёмкость массива до 15 после чего
увеличим размер до 11 с помощью функции int_vector_resize:
vector v:
data: 0x55f37369c720
size: 11
capacity: 11
-----
Попробуем уменьшить размер массива до 4 с помошью функции int_vector_resize
vector v:
data: 0x55f37369c720
size: 4
capacity: 4
Размер массива уменьшился, но память не перевыделилась,
значит функция отработала правильно
-----
```

```
Создаем копию массива v - массив t с помощью функции int_vector_copy
vector v:
data: 0x55f37369c720
size: 4
capacity: 4
элементы вектора v: 0 = 34  1 = 37  2 = 999  3 = 0

vector t:
data: 0x55f37369c770
size: 4
capacity: 4
элементы вектора t: 0 = 34  1 = 37  2 = 999  3 = 0
-----
Затем освобождаем паямять выделенную под массив t с помощью функции int_vector_free
vector t:
data: 0x55f37369c770
size: 94504101724176
capacity: 4
```

Приложение:

1)

```c
#pragma once
#include <stddef.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);
```

2)

```c
#include "IntVector.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

IntVector* int_vector_new(size_t initial_capacity)
{
    IntVector* v = malloc(sizeof(IntVector));
    if (!v) {
        return NULL;
    }

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

    v->size = 0;
    v->capacity = initial_capacity;
    return v;
}

IntVector* int_vector_copy(const IntVector* v)
{
    IntVector* t = malloc(sizeof(IntVector));
    if (t == NULL)
        return NULL;

    t->data = malloc(v->capacity * sizeof(int));
```

```
31      if (t->data == NULL) {
32          free(t);
33          return NULL;
34      }
35
36      memcpy(t->data, v->data, sizeof(int) * v->capacity);
37      t->size = v->size;
38      t->capacity = v->capacity;
39      return t;
40  }
41
42  void int_vector_free(IntVector* v)
43  {
44      if (!v) {
45          return;
46      }
47      free(v->data);
48      free(v);
49  }
50
51  int int_vector_get_item(const IntVector* v, size_t index)
52  {
53      return v->data[index];
54  }
55
56  void int_vector_set_item(IntVector* v, size_t index, int item)
57  {
58      if (index <= v->capacity)
59          v->data[index] = item;
60  }
61
62  size_t int_vector_get_size(const IntVector* v)
63  {
64      return v->size;
65  }
66
67  size_t int_vector_get_capacity(const IntVector* v)
68  {
69      return v->capacity;
70  }
71
72  int int_vector_push_back(IntVector* v, int item)
73  {
74      if (v->size < v->capacity) {
75          v->data[v->size] = item;
76          v->size++;
77      } else {
78          v->capacity = v->capacity * 2;
79          v->data = realloc(v->data, v->capacity * sizeof(int));
80
81          v->data[v->size] = item;
82          v->size++;
83      }
84      return 0;
85  }
86
87  void int_vector_pop_back(IntVector* v)
88  {
```

```
89      if (v->size != 0) {
90          v->size--;
91      }
92  }
93
94  int int_vector_shrink_to_fit(IntVector* v)
95  {
96      if (v->size < v->capacity) {
97          v->capacity = v->size;
98          int* t = realloc(v->data, v->size * sizeof(int));
99          if (t == NULL)
100             return -1;
101         v->data = t;
102         return 0;
103     }
104     return -1;
105 }
106
107 int int_vector_resize(IntVector* v, size_t new_size)
108 {
109     if (new_size == v->size)
110         return 0;
111     if (new_size > v->size) {
112         int* t = realloc(v->data, new_size * sizeof(int));
113         if (t == NULL)
114             return -1;
115         v->data = t;
116         for (int i = new_size - v->size; i < new_size; i++)
117             v->data[i] = 0;
118     }
119     v->size = new_size;
120     v->capacity = new_size;
121     return 0;
122 }
123
124 int int_vector_reserve(IntVector* v, size_t new_capacity)
125 {
126     if (new_capacity > v->capacity) {
127         v->capacity = new_capacity;
128         int* t = realloc(v->data, new_capacity * sizeof(int));
129         if (t == NULL)
130             return -1;
131         v->data = t;
132         return 0;
133     }
134     return -1;
135 }
```

3)

```
1  #include "IntVector.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main()
6  {
7      printf("Создаем массив v ёмкостью 7 с помощью функции int_vector_new:\n");
```

```
 8      IntVector* v = int_vector_new(7);
 9      printf("vector v:\ndata: %p\nsize: %ld\ncapacity: %ld\n",
10             v->data,
11             v->size,
12             v->capacity);
13
14      printf("-----\nЗаполняем массив, используя функцию int_vector_get_capacity "
15             "что "
16             "бы "
17             "узнать его ёмкость\nи функцию int_vector_push_back что бы добавить "
18             "элемент в конец массива.\n");
19      for (int i = 0; i < int_vector_get_capacity(v); i++) {
20          int_vector_push_back(v, (rand() % 50 + 1));
21      }
22      printf("Выводим массив, используя функцию int_vector_get_size что бы "
23             "узнать "
24             "его размер\nи функцию int_vector_get_item что бы получать элементы "
25             "массива.\n");
26      for (int i = 0; i < int_vector_get_size(v); i++) {
27          printf("%d = %d\t", i, int_vector_get_item(v, i));
28      }
29
30      printf("\n-----\nИзменим третий элемент массива на 999 с помощью фунции "
31             "int_vector_set_item\nи добавим в последниий элемент массива 15 с "
32             "помощью "
33             "функции int_vector_push_back:\n");
34      int_vector_set_item(v, 2, 999);
35      int_vector_push_back(v, 15);
36      printf("vector v:\ndata: %p\nsize: %ld\ncapacity: %ld\n",
37             v->data,
38             v->size,
39             v->capacity);
40      for (int i = 0; i < int_vector_get_size(v); i++) {
41          printf("%d = %d\t", i, int_vector_get_item(v, i));
42      }
43      printf("\nТак как изначальной ёмкости (7) не хватило для добавления нового "
44             "элемента \nона удвоилась");
45
46      printf("\n-----\nПопробуем уменьшить емкость ёмкость массива до 5 с "
47             "помощью "
48             "функции int_vector_reserve\n");
49      int_vector_reserve(v, 5);
50      printf("vector v:\ndata: %p\nsize: %ld\ncapacity: %ld\n",
51             v->data,
52             v->size,
53             v->capacity);
54      printf("Так как ёмкость меньше исходной ничего не произошло.");
55
56      printf("\n-----\nУвеличим ёмкость массива до 15 с помощью функции "
57             "int_vector_reserve");
58      int_vector_reserve(v, 15);
59      printf("\nvector v:\ndata: %p\nsize: %ld\ncapacity: %ld\n",
60             v->data,
61             v->size,
62             v->capacity);
63
64      printf("-----\nУменьшим ёмкость массива до его размера с помощью функции "
65             "int_vector_shrink_to_fit");
```

```
66      int_vector_shrink_to_fit(v);
67      printf("\nvector v:\ndata: %p\nsize: %ld\ncapacity: %ld\n",
68             v->data,
69             v->size,
70             v->capacity);
71
72      printf("-----\nУвеличим ёмкость массива до 15 после чего\nувеличим размер "
73             "до 11 "
74             "с помощью функции int_vector_resize:");
75      int_vector_reserve(v, 15);
76      int_vector_resize(v, 11);
77      printf("\nvector v:\ndata: %p\nsize: %ld\ncapacity: %ld\n",
78             v->data,
79             v->size,
80             v->capacity);
81
82      printf("-----\nПопробуем уменьшить размер массива до 4 с помошью функции "
83             "int_vector_resize");
84      int_vector_resize(v, 4);
85      printf("\nvector v:\ndata: %p\nsize: %ld\ncapacity: %ld\n",
86             v->data,
87             v->size,
88             v->capacity);
89      printf("Размер массива уменьшился, но память не перевыделилась, \nзначит "
90             "функция отработала правильно");
91
92      printf("\n-----\nСоздаем копию массива v - массив t с помощью функции "
93             "int_vector_copy");
94      printf("\nvector v:\ndata: %p\nsize: %ld\ncapacity: %ld\n",
95             v->data,
96             v->size,
97             v->capacity);
98      printf("элементы вектора v: ");
99      for (int i = 0; i < int_vector_get_size(v); i++) {
100         printf("%d = %d  ", i, int_vector_get_item(v, i));
101     }
102     IntVector* t = int_vector_copy(v);
103     printf("\n\nvector t: \ndata: %p\nsize: %ld\ncapacity: %ld\n",
104            t->data,
105            t->size,
106            t->capacity);
107     printf("элементы вектора t: ");
108     for (int i = 0; i < int_vector_get_size(t); i++) {
109         printf("%d = %d  ", i, int_vector_get_item(t, i));
110     }
111
112     printf("\n-----\nЗатем освобождаем паямять выделенную под массив t с "
113            "помощью "
114            "функции int_vector_free\n");
115     int_vector_free(t);
116     printf("vector t: \ndata: %p\nsize: %ld\ncapacity: %ld\n",
117            t->data,
118            t->size,
119            t->capacity);
120     return 0;
121 }
```