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-222
«4» мая 2023 г. _____ Очнев А.Д.

Проверил:
Старший преподаватель кафедры
вычислительных систем
«5» мая 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.

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.

Пример:

```
data                capacity = 6
  |                       |
  v                       v
  +---+---+---+---+---+---+
  | 3 | 1 | 4 | 1 | 5 |   |
  +---+---+---+---+---+---+
                    ^
                    |
                  size = 5

int_vector_push_back(v, 9)


data                capacity = 6
  |                       |
  v                       v
  +---+---+---+---+---+---+
  | 3 | 1 | 4 | 1 | 5 | 9 |
  +---+---+---+---+---+---+
                        ^
                        |
                      size = 6

int_vector_push_back(v, 2)


data                                    capacity = 12
  |                                           |
  v                                           v
  +---+---+---+---+---+---+---+---+---+---+---+---+
  | 3 | 1 | 4 | 1 | 5 | 9 | 2 |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+---+---+---+
                            ^
                            |
                          size = 7
```

`void int_vector_pop_back(IntVector *v)`

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

```
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

Создаём заголовочный файл IntVector.h, где объявляем шаблон структуры и прототипы функций:

```
#pragma once

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);
```

Реализовываем функции в файле IntVector.c:

int_vector_new

Выделяем память для структуры с помощью функции malloc, если выделение не произошло, то возвращаем NULL.Выделяем память для массива data и заполняем нулями с помощью функции calloc, если выделение не произошло, то освобождаем память выделенную под структуру и возвращаем NULL Приравниваем полю структуры capacity значение переданной ёмкости. Возвращаем указатель на структуру.

```
intvector *int_vector_new(size_t initial_capacity) {
  intvector *v = NULL;
  v = malloc(sizeof(*v));
  if (!v)
    return NULL;
  v->data = calloc(initial_capacity, sizeof(int));
  if (!v->data) {
    free(v);
    return NULL;
  }
  v->capacity = initial_capacity;
  return v;
}
```

int_vector_copy

Выделяем память для структуры с помощью функции malloc, если выделение не

произошло, то возвращаем NULL.Выделяем память для массива data и заполняем нулями с помощью функции calloc, если выделение не произошло, то освобождаем память выделенную под структуру и возвращаем NULL. Приравниваем полям структуры capacity и size значения переданной структуры. Возвращаем указатель на структуру.

```
intvector *int_vector_copy(const intvector *v) {
  intvector *cpy = NULL;
  cpy = malloc(sizeof(*v));
  cpy->capacity = int_vector_get_capacity(v);
  if (!cpy)
    return NULL;
  cpy->data = calloc(cpy->capacity, sizeof(int));
  if (!cpy->data) {
    free(cpy);
    return NULL;
  }
  for (int i = 0; i < cpy->capacity; i++) {
    cpy->data[i] = v->data[i];
  }
  cpy->capacity = v->capacity;
  cpy->size = v->size;
  return cpy;
}
```

int_vector_free

Чтобы избежать утечек памяти, сначала очищаем память для массива, затем для структуры.

```
void int_vector_free(intvector *v) {
  free(v->data);
  free(v);
}
```

int_vector_get_item

Возвращаем значение массива по индексу index.

```
int int_vector_get_item(const intvector *v, size_t index) {
  return v->data[index];
}
```

int_vector_set_item

Если размер массива больше или равен index, тогда мы приравниваем элемент массива по индексу index к значению item. Если переданное значение index лежит в диапазоне от размера до ёмкости, то тогда мы приравниваем элемент массива по индексу index к значению item, после чего приравниваем размер к index. Если index больше ёмкости, то мы добавляем элемент item в конец массива с помощью функции push_back.

```
void int_vector_set_item(intvector *v, size_t index, int item) {
  if (int_vector_get_size(v) >= index) {
    v->data[index] = item;
  } else if (int_vector_get_size(v) < index && int_vector_get_capacity(v) >= index) {
    v->data[index] = item;
    v->size = index+1;
  } else {
    int_vector_push_back(v, item);
  }
}
```

int_vector_get_size

Возвращаем значение поля size из структуры.

```
size_t int_vector_get_size(const intvector *v) { return v->size; }
```

int_vector_get_capacity

Возвращаем значение поля capacity из структуры.

```
size_t int_vector_get_capacity(const intvector *v) { return v->capacity; }
```

int_vector_push_back

Если размер массива меньше его емкости, то мы приравниваем элемент массива по индексу size к значению item, после чего увеличиваем размер на один. Если размер равен емкости, то мы увеличиваем емкость на один, после чего довыделяем память для массива. Если выделение не произошло, то освобождаем память выделенную под структуру и возвращаем -1. Приравниваем элемент массива по индексу size к значению item, после чего увеличиваем размер на один. При успешной инициализации возвращаем 0, -1 если произошла ошибка.

```
int int_vector_push_back(intvector *v, int item) {
  if (int_vector_get_size(v) < int_vector_get_capacity(v)) {
    v->data[int_vector_get_size(v)] = item;
    v->size++;
  } else {
    v->capacity += 1;
    v->data = realloc(v->data, v->capacity * sizeof(int));
    if (!v->data) {
      free(v);
      return -1;
    }
    v->data[int_vector_get_size(v)] = item;
    v->size++;
  }
  if (v->data[int_vector_get_size(v)-1] == item)
    return 0;
  else
    return -1;
}
```

int_vector_pop_back

Если размер массива больше 0, то мы приравниваем элемент массива по индексу size-1 к нулю, после чего уменьшаем размер на один.

```
void int_vector_pop_back(intvector *v) {
  if (int_vector_get_size > 0){
    v->data[int_vector_get_size(v)-1] = 0;
    v->size = int_vector_get_size(v)-1;
  }
}
```

int_vector_shrink_to_fit

Приравниваем емкость массива к его размеру, после чего довыделяем память для массива. Если выделение не произошло, то освобождаем память выделенную под структуру и возвращаем -1. При успешном уменьшении ёмкости возвращаем 0, -1 если произошла ошибка.

```
int int_vector_shrink_to_fit(intvector *v) {
  v->capacity = int_vector_get_size(v);
  v->data = realloc(v->data, int_vector_get_capacity(v) * sizeof(int));
  if (!v->data) {
    free(v);
    return -1;
  }
  if (int_vector_get_capacity(v) == int_vector_get_size(v))
    return 0;
  else
    return -1;
}
```

int_vector_resize

Если новый размер больше текущего, но меньше емкости, то размер массива приравниваем к новому размеру. Если новый размер меньше текущего, то приравниваем каждый элемент массива по индексам от нового размера до старого к нулю, после чего размер массива приравниваем к новому размеру. Если новый размер больше ёмкости, то довыделяем память для массива функцией realloc, проверяем, выделилась ли память, после чего размер массива приравниваем к новому размеру. При успешном изменении размера возвращаем 0, -1 если произошла ошибка.

```c
int int_vector_resize(intvector *v, size_t new_size) {
  if (int_vector_get_size(v) < new_size &&
      new_size < int_vector_get_capacity(v)) {
    v->size = new_size;
  } else if (new_size < int_vector_get_size(v)) {
    for (int i = new_size; i < int_vector_get_size(v); i++) {
      v->data[i] = 0;
    }
    v->size = new_size;
  } else if (new_size > int_vector_get_capacity(v)) {
    v->data = realloc(v->data, new_size * sizeof(int));
    if (!v->data) {
      free(v);
      return -1;
    }
    v->size = new_size;
  }
  if (int_vector_get_size(v) == new_size)
    return 0;
  else
    return -1;
}
```

int_vector_reserve

Если новая емкость меньше или равна старой емкости массива, то возвращаем 0. Иначе довыделяем память для массива функцией realloc, проверяем, выделилась ли память. Изменяем значение емкости на новое.

```c
int int_vector_reserve(intvector *v, size_t new_capacity) {
  if (new_capacity <= int_vector_get_capacity(v))
    return 0;
  v->data = realloc(v->data, new_capacity * sizeof(int));
  if (!v->data) {
    free(v);
    return -1;
  }
  v->capacity = new_capacity;
  return 0;
}
```

В файле main.c проверяем работу функций:

```c
#include <stdio.h>
#include <stdlib.h>

#include "IntVector.h"

void print_vector(intvector *v) {
  printf("Size=%ld Capacity=%ld\nElements:\n", int_vector_get_size(v),
         int_vector_get_capacity(v));
  for (int i = 0; i < int_vector_get_capacity(v); i++) {
    printf("%d ", int_vector_get_item(v, i));
  }
  printf("\n\n");
}
```

```c
int main() {
  intvector *v = int_vector_new(10);
  printf("Создаём массив v с ёмкостью 10:\n");
  print_vector(v);

  for (int i = 0; i < 10; i++) {
    int_vector_set_item(v, i, i);
  }
  printf("Заполняем массив, размер массива теперь также теперь равен 10:\n");
  print_vector(v);

  intvector *t = int_vector_copy(v);
  int_vector_free(v);
  printf("Создаём копию структуры v и освобождаем память для вектора v:\n");
  print_vector(t);

  printf("Пятый элемент массива: %d\n\n", int_vector_get_item(t, 4));

  printf("Добавляем один элемент в конец заполненного массива:\n");
  int_vector_push_back(t, 10);
  print_vector(t);

  printf("Удаляем последний элемент массива:\n");
  int_vector_pop_back(t);
  print_vector(t);

  printf("Увеличиваем ёмкость до 14:\n");
  int_vector_reserve(t, 14);
  print_vector(t);

  printf("Увеличиваем размер до 12:\n");
  int_vector_resize(t, 12);
  print_vector(t);

  printf("Уменьшаем ёмкость до 10:\n");
  int_vector_reserve(t, 10);
  print_vector(t);

  printf("Уменьшаем размер до 8:\n");
  int_vector_resize(t, 8);
  print_vector(t);

  printf("Уменьшаем ёмкость до размера, после чего очищаем память для структуры:\n");
  int_vector_shrink_to_fit(t);
  int_vector_set_item(t, 4, 30);
  print_vector(t);

  int_vector_free(t);

  return 0;
}
```

Вывод:

```
[artem@fedora second]$ ./main
Создаём массив v с ёмкостью 10:
Size=0 Capacity=10
Elements:
0 0 0 0 0 0 0 0 0 0

Заполняем массив, размер массива теперь также теперь равен 10:
Size=10 Capacity=10
Elements:
0 1 2 3 4 5 6 7 8 9

Создаём копию структуры v и освобождаем память для вектора v:
Size=10 Capacity=10
Elements:
0 1 2 3 4 5 6 7 8 9

Пятый элемент массива: 4

Добавляем один элемент в конец заполненного массива:
Size=11 Capacity=11
Elements:
0 1 2 3 4 5 6 7 8 9 10

Удаляем последний элемент массива:
Size=10 Capacity=11
Elements:
0 1 2 3 4 5 6 7 8 9 0

Увеличиваем ёмкость до 14:
Size=10 Capacity=14
Elements:
0 1 2 3 4 5 6 7 8 9 0 0 0 0

Увеличиваем размер до 12:
Size=12 Capacity=14
Elements:
0 1 2 3 4 5 6 7 8 9 0 0 0 0
```

```
Уменьшаем ёмкость до 10:
Size=12 Capacity=14
Elements:
0 1 2 3 4 5 6 7 8 9 0 0 0 0

Уменьшаем размер до 8:
Size=8 Capacity=14
Elements:
0 1 2 3 4 5 6 7 0 0 0 0 0 0

Уменьшаем ёмкость до размера, после чего очищаем память для структуры:
Size=8 Capacity=8
Elements:
0 1 2 3 30 5 6 7
```

Проверим на наличие утечек с помощью команды valgrind:

```
==12608==
==12608== HEAP SUMMARY:
==12608==     in use at exit: 0 bytes in 0 blocks
==12608==   total heap usage: 8 allocs, 8 frees, 1,284 bytes allocated
==12608==
==12608== All heap blocks were freed -- no leaks are possible
==12608==
==12608== Use --track-origins=yes to see where uninitialised values come from
==12608== For lists of detected and suppressed errors, rerun with: -s
==12608== ERROR SUMMARY: 51 errors from 16 contexts (suppressed: 0 from 0)
```

Утечек не обнаружено.

ПРИЛОЖЕНИЕ

main.c

```c
#include <stdio.h>
#include <stdlib.h>

#include "IntVector.h"

void print_vector(intvector *v) {
  printf("Size=%ld Capacity=%ld\nElements:\n", int_vector_get_size(v),
         int_vector_get_capacity(v));
  for (int i = 0; i < int_vector_get_capacity(v); i++) {
    printf("%d ", int_vector_get_item(v, i));
  }
  printf("\n\n");
}

int main() {
  intvector *v = int_vector_new(10);
  printf("Создаём массив v с ёмкостью 10:\n");
  print_vector(v);

  for (int i = 0; i < 10; i++) {
    int_vector_set_item(v, i, i);
  }
  printf("Заполняем массив, размер массива теперь также теперь равен
10:\n");
  print_vector(v);

  intvector *t = int_vector_copy(v);
  int_vector_free(v);
  printf("Создаём копию структуры v и освобождаем память для вектора
v:\n");
  print_vector(t);

  printf("Пятый элемент массива: %d\n\n", int_vector_get_item(t, 4));

  printf("Добавляем один элемент в конец заполненного массива:\n");
  int_vector_push_back(t, 10);
  print_vector(t);

  printf("Удаляем последний элемент массива:\n");
  int_vector_pop_back(t);
  print_vector(t);

  printf("Увеличиваем ёмкость до 14:\n");
  int_vector_reserve(t, 14);
  print_vector(t);

  printf("Увеличиваем размер до 12:\n");
  int_vector_resize(t, 12);
  print_vector(t);

  printf("Уменьшаем ёмкость до 10:\n");
  int_vector_reserve(t, 10);
  print_vector(t);

  printf("Уменьшаем размер до 8:\n");
  int_vector_resize(t, 8);
```

```
57      print_vector(t);
58
59      printf("Уменьшаем ёмкость до размера, после чего очищаем память для
60    структуры:\n");
61      int_vector_shrink_to_fit(t);
62      int_vector_set_item(t, 4, 30);
63      print_vector(t);
64
65      int_vector_free(t);
66
67      return 0;
68    }
69
```

IntVector.h

```
1   #pragma once
2
3   typedef struct
4   {
5       int* data;
6       size_t size;
7       size_t capacity;
8   } intvector;
9
10  intvector *int_vector_new(size_t initial_capacity);
11  intvector *int_vector_copy(const intvector *v);
12  void int_vector_free(intvector *v);
13  int int_vector_get_item(const intvector *v, size_t index);
14  void int_vector_set_item(intvector *v,size_t index, int item);
15  size_t int_vector_get_size(const intvector *v);
16  size_t int_vector_get_capacity(const intvector *v);
17  int int_vector_push_back(intvector *v, int item);
18  void int_vector_pop_back(intvector *v);
19  int int_vector_shrink_to_fit(intvector *v);
20  int int_vector_resize(intvector *v,size_t new_size);
21  int int_vector_reserve(intvector *v,size_t new_capacity);
22
23
```

IntVector.c

```c
#include <stdlib.h>

#include "IntVector.h"

intvector *int_vector_new(size_t initial_capacity) {
  intvector *v = NULL;
  v = malloc(sizeof(*v));
  if (!v)
    return NULL;
  v->data = calloc(initial_capacity, sizeof(int));
  if (!v->data) {
    free(v);
    return NULL;
  }
  v->capacity = initial_capacity;
  return v;
}

intvector *int_vector_copy(const intvector *v) {
  intvector *cpy = NULL;
  cpy = malloc(sizeof(*v));
  cpy->capacity = int_vector_get_capacity(v);
  if (!cpy)
    return NULL;
  cpy->data = calloc(cpy->capacity, sizeof(int));
  if (!cpy->data) {
    free(cpy);
    return NULL;
  }
  for (int i = 0; i < cpy->capacity; i++) {
    cpy->data[i] = v->data[i];
  }
  cpy->capacity = v->capacity;
  cpy->size = v->size;
  return cpy;
}

void int_vector_free(intvector *v) {
  free(v->data);
  free(v);
}

int int_vector_get_item(const intvector *v, size_t index) {
  return v->data[index];
}

void int_vector_set_item(intvector *v, size_t index, int item) {
  if (int_vector_get_size(v) >= index) {
    v->data[index] = item;
  } else if (int_vector_get_size(v) < index &&
int_vector_get_capacity(v) >= index) {
    v->data[index] = item;
    v->size = index+1;
  } else {
```

```
61        int_vector_push_back(v, item);
62      }
63    }
64
65    size_t int_vector_get_size(const intvector *v) { return v->size; }
66
67    size_t int_vector_get_capacity(const intvector *v) { return
68    v->capacity; }
69
70    int int_vector_push_back(intvector *v, int item) {
71      if (int_vector_get_size(v) < int_vector_get_capacity(v)) {
72        v->data[int_vector_get_size(v)] = item;
73        v->size++;
74      } else {
75        v->capacity += 1;
76        v->data = realloc(v->data, v->capacity * sizeof(int));
77        if (!v->data) {
78          free(v);
79          return -1;
80        }
81        v->data[int_vector_get_size(v)] = item;
82        v->size++;
83      }
84      if (v->data[int_vector_get_size(v)-1] == item)
85        return 0;
86      else
87        return -1;
88    }
89
90    void int_vector_pop_back(intvector *v) {
91      if (int_vector_get_size > 0){
92        v->data[int_vector_get_size(v)-1] = 0;
93        v->size = int_vector_get_size(v)-1;
94      }
95    }
96
97    int int_vector_shrink_to_fit(intvector *v) {
98      v->capacity = int_vector_get_size(v);
99      v->data = realloc(v->data, int_vector_get_capacity(v) *
100   sizeof(int));
101     if (!v->data) {
102       free(v);
103       return -1;
104     }
105     if (int_vector_get_capacity(v) == int_vector_get_size(v))
106       return 0;
107     else
108       return -1;
109   }
110
111   int int_vector_resize(intvector *v, size_t new_size) {
112     if (int_vector_get_size(v) < new_size &&
113         new_size < int_vector_get_capacity(v)) {
114       v->size = new_size;
115     } else if (new_size < int_vector_get_size(v)) {
116       for (int i = new_size; i < int_vector_get_size(v); i++) {
117
118
119
120
121
122
```

```
123         v->data[i] = 0;
124       }
125       v->size = new_size;
126     } else if (new_size > int_vector_get_capacity(v)) {
127       v->data = realloc(v->data, new_size * sizeof(int));
128       if (!v->data) {
129         free(v);
130         return -1;
131       }
132       v->size = new_size;
133     }
134     if (int_vector_get_size(v) == new_size)
135       return 0;
136     else
137       return -1;
138   }
139
140   int int_vector_reserve(intvector *v, size_t new_capacity) {
141     if (new_capacity <= int_vector_get_capacity(v))
142       return 0;
143     v->data = realloc(v->data, new_capacity * sizeof(int));
144     if (!v->data) {
145       free(v);
146       return -1;
147     }
148     v->capacity = new_capacity;
149     return 0;
150   }
151
152
```