МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №5b

по дисциплине: Основы программирования
тема: Реализация структуры данных «Вектор»

Выполнил: ст. группы ПВ-211
Чувилко Илья Романович

Проверили:
Притчин Иван Сергеевич
Черников Сергей Викторович

Белгород 2021 г.

Цель работы: усовершенствование навыков в создании библиотек, получение навыков работы с системой контроля версий `git`

Содержание отчета:

- Тема лабораторной работы.
- Цель лабораторной работы.
- Ссылка на открытый репозиторий с решением.
- Исходный код файлов:
  - vector.h / vector.c
  - *vectorVoid.h / vectorVoid.c – main.c
- Результат выполнения команд
- Выводы по работе

Требования:

Обратите особое внимание на задания к лабораторной работе. В частности, на требование к коммитам в процессе выполнения работы. Если работа будет выполнена без них или будут отсутствовать промежуточные коммиты, она не будет засчитана. С целью улучшения умения чтения текста лабораторной, будет выдано дополнительное задание.

Выполнение работы:

1. `vector createVector(size_t n)` – возвращает структуру-дескриптор вектор из n значений.

Код программы:

```
// возвращает структуру-дескриптор вектор из n значений.
vector createVector(size_t n) {
  vector v;
  v.data = (int *) malloc(n * sizeof(int));
  if (v.data == 0) {
    fprintf(stderr, "bad alloc ");
    exit(1);
  }
  v.size = 0;
  v.capacity = n;
  return v;
}
```

После выполнения следующего кода:

```
int main() {
  vector v = createVector(SIZE_MAX);

  return 0;
}
```

Функция завершает выполнение с кодом 1 и выводит ошибку bad alloc:

```
"D:\BGTU\Programming Basics\Lab 5b\cmake-build-debug\Lab_5b.exe"
bad alloc
Process finished with exit code 1
```

Дополнительные тесты:

```
void test_createVector_emptyVector() {
  vector v = createVector(0);

  assert(v.size == 0);
  assert(v.capacity == 0);

  deleteVector(&v);
}
```

```
void test_createVector_notEmptyVector() {
  vector v = createVector(5);

  assert(v.size == 0);
  assert(v.capacity == 5);

  deleteVector(&v);
}
```

2. `void reserve(vector *v, size_t newCapacity)` – изменяет количество памяти, выделенное под хранение элементов вектора. Пример, когда новая вместимость больше:

Код программы:

```
// изменяет количество памяти, выделенное под хранение элементов вектора.
void reserve(vector *v, size_t newCapacity) {
  if (newCapacity == 0) {
    v->data = NULL;
    v->size = 0;
  } else {
    if (newCapacity < v->size)
      v->size = newCapacity;
    v->data = (int *) realloc(v->data, newCapacity * sizeof(int));
    if (v->data == 0) {
      fprintf(stderr, "bad alloc ");
      exit(1);
    }
  }
  v->capacity = newCapacity;
}
```

Дополнительные тесты:

```
void test_reserve_newCapacityZero() {
  vector v = createVector(5);

  reserve(&v, 0);

  assert(v.data == 0);
  assert(v.size == 0);
  assert(v.capacity == 0);
  deleteVector(&v);
}

void test_reserve_newCapacitySmallerSize() {
  vector v = createVector(5);

  pushBack(&v, 0);
  pushBack(&v, 1);
  pushBack(&v, 2);
  pushBack(&v, 3);
  reserve(&v, 3);

  int resArray[] = {0, 1, 2};

  assert(isEqualArray(v.data, v.size, resArray, 3));
  assert(v.size == 3);
  assert(v.capacity == 3);
  deleteVector(&v);
}
```

```
void test_reserve_newCapacityBiggerSize() {
  vector v = createVector(5);

  pushBack(&v, 0);
  pushBack(&v, 1);
  pushBack(&v, 2);
  pushBack(&v, 3);
  reserve(&v, 8);

  int resArray[] = {0, 1, 2, 3};

  assert(isEqualArray(v.data, v.size, resArray, 4));
  assert(v.size == 4);
  assert(v.capacity == 8);
  deleteVector(&v);
}
```

3. void clear(vector *v) – удаляет элементы из контейнера, но не освобождает выделенную память.

```
// void clear(vector *v) - удаляет элементы из контейнера,
// но не освобождает выделенную память.
void clear(vector *v) {
  v->size = 0;
}
```

4. void shrinkToFit(vector *v) – освобождает память, выделенную под неиспользуемые элементы

```
// освобождает память, выделенную под неиспользуемые элементы
void shrinkToFit(vector *v) {
  v->data = (int *) realloc(v->data, v->size * sizeof(int));
  v->capacity = v->size;
}
```

Дополнительные тесты:

```
void test_shrinkToFit() {
  vector v = createVector(6);

  pushBack(&v, 0);
  pushBack(&v, 1);
  pushBack(&v, 2);
  pushBack(&v, 3);
  shrinkToFit(&v);
  int resArray[] = {0, 1, 2, 3};

  assert(isEqualArray(v.data, v.size, resArray, 4));
  assert(v.size == 4);
  assert(v.capacity == 4);
  deleteVector(&v);
}
```

5. void deleteVector(vector *v) – освобождает память, выделенную вектору

```
// Освобождает память, выделенную вектору
void deleteVector(vector *v) {
  free(v->data);
}
```

6. bool isEmpty(vector *v) - Возвращает true, если вектор пустой. Иначе false.

```
// Возвращает true, если вектор пустой. Иначе false.
bool isEmpty(vector *v) {
  return v->size == 0 ? true : false;
}
```

7. bool isFull(vector *v) - Возвращает true, если вектор пустой. Иначе false.

```
// Возвращает true, если вектор полный. Иначе false.
bool isFull(vector *v) {
  return v->size == v->capacity ? true : false;
}
```

8. int getVectorValue(vector *v, size_t I) - Функция которая возвращает i-ый элемент вектора v. Тело функции – одна строка кода.

```
// Возвращает true, если вектор полный. Иначе false.
bool isFull(vector *v) {
  return v->size == v->capacity ? true : false;
}
```

9. void pushBack(vector *v, int x)– добавляет элемент x в конец вектора v. Если вектор заполнен, увеличивает количество выделенной ему памяти в 2 раза, используя reserve.

```
// Добавляет элемент x в конец вектора v. Если вектор заполнен,
// увеличивает количество выделенной ему памяти в 2 раза, используя reserve.
void pushBack(vector *v, int x) {
  if (isFull(v))
    reserve(v, v->capacity == 0 ? 1 : v->capacity * 2);
  v->data[v->size++] = x;
}
```

Дополнительные тесты:

```
void test_pushBack_emptyVector1() {
  vector v = createVector(2);
  pushBack(&v, 1);

  assert(v.data[0] == 1);
  assert(v.capacity == 2);
  deleteVector(&v);
}

void test_pushBack_emptyVector2() {
  vector v = createVector(0);
  pushBack(&v, 1);

  assert(v.data[0] == 1);
  assert(v.capacity == 1);
  deleteVector(&v);
}

void test_pushBack_fullVector() {
  vector v = createVector(2);
  pushBack(&v, 1);
  pushBack(&v, 2);
  pushBack(&v, 3);
  assert(v.data[2] == 3);

  deleteVector(&v);
}
```

10. void popBack(vector *v) – удаляет последний элемент из вектора. Функция должна ’выкидывать’ в поток ошибок сообщение, если вектор пуст и закончить выполнение с кодом 1.

```
// Удаляет последний элемент из вектора. Функция ’выкидывает’ в поток
// ошибок сообщение, если вектор пуст и заканчивает выполнение с кодом 1.
void popBack(vector *v) {
  if (isEmpty(v)) {
    fprintf(stderr, "vector is empty");
    exit(1);
  }
  v->size--;
}
```

11. int* atVector(vector *v, size_t index) – возвращает указатель на index-ый элемент вектора. Если осуществляется попытка получить доступ вне пределов используемых элементов вектора, в поток ошибок должна выводиться ошибка: "IndexError: a[index] is not exists", где в качестве index указывается позиция элемента, к которому пытались осуществить доступ

```
// Возвращает указатель на index-ый элемент вектора. Если осуществляется
// попытка получить доступ вне пределов используемых элементов вектора,
// в поток ошибок выводится ошибка: "IndexError: a[index] is not exists",
// где в качестве index указывается позиция элемента, к которому пытались
// осуществить доступ.
int* atVector(vector *v, size_t index) {
  if (index > v->size)
    fprintf(stderr, "a[%zu] is not exist", index);
  return &v->data[index];
}
```

12. int* back(vector *v) – возвращает указатель на последний элемент вектора.

```
// Возвращает указатель на последний элемент вектора
int* back(vector *v) {
  if (isEmpty(v))
    fprintf(stderr, "vector is empty");
  return &v->data[v->size - 1];
}
```

13. int* front(vector *v) – возвращает указатель на первый элемент вектора.

```
int* front(vector *v) {
  if (isEmpty(v))
    fprintf(stderr, "vector is empty");
  return &v->data[0];
}
```

Коммиты vector.c:

```
commit 00b296e874323a1830d53a29ff5122a724ee9a46
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Sun Feb 6 12:27:37 2022 +0300

    append access functions

 libs/data_structures/vector/vector.c | 5 +++++
 libs/data_structures/vector/vector.h | 2 ++
 2 files changed, 7 insertions(+)

commit 45d49b6fc51121dca63510e0c3c82141caafcb0f
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Sun Feb 6 00:21:08 2022 +0300

    index vector

 libs/data_structures/vector/vector.c | 19 ++++++++++++++++---
 libs/data_structures/vector/vector.h |  4 ++++
 2 files changed, 20 insertions(+), 3 deletions(-)

commit cbcca423c28ebfab4f1b4f4dffcd91fe75a1a3a2
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Sat Feb 5 19:52:36 2022 +0300
```

```
    +lib array

 libs/data_structures/vector/vector.c | 26 ++++++++++++++++++--------
 libs/data_structures/vector/vector.h |  2 ++
 2 files changed, 20 insertions(+), 8 deletions(-)

commit 5f3a00a39fec6151a9d2a14ae0a844f91c9f296d
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Sat Feb 5 18:14:56 2022 +0300

    №8

 libs/data_structures/vector/vector.c | 20 ++++++++++++++++---
 libs/data_structures/vector/vector.h |  6 ++++++
 2 files changed, 23 insertions(+), 3 deletions(-)

commit 1d180d157c8be5f277f2aa078fc1a07dde892a43
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Wed Feb 2 19:43:55 2022 +0300

    second

 libs/data_structures/vector/vector.c | 19 ++++++++++++++++-
 libs/data_structures/vector/vector.h |  3 +++
 2 files changed, 21 insertions(+), 1 deletion(-)
```

```
commit b55fc52df732873d7865ab9215596f30c1dedb16
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Wed Feb 2 17:37:01 2022 +0300

    memory usage of vector

 libs/data_structures/vector/vector.c | 35 ++++++++++++++++++++++++++++++++++----
 libs/data_structures/vector/vector.h | 20 ++++++++++++++++++++
 2 files changed, 51 insertions(+), 4 deletions(-)

commit 1d98baac84d15675447877703797a98acadd0032
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Wed Feb 2 16:49:20 2022 +0300

    second

 libs/data_structures/vector/vector.c | 5 +++++
 libs/data_structures/vector/vector.h | 4 ++++
 2 files changed, 9 insertions(+)
```

1. `vectorVoid createVectorV(size_t n, size_t baseTypeSize)` – возвращает структуру-дескриптор вектор из n значений.

Код программы:

```
// возвращает структуру-дескриптор вектор из n значений.
vectorVoid createVectorV(size_t n, size_t baseTypeSize) {
  vectorVoid v;
  v.data = (void *) malloc(n * baseTypeSize);
  if (v.data == 0) {
    fprintf(stderr, "bad alloc ");
    exit(1);
  }
  v.size = 0;
  v.capacity = n;
  v.baseTypeSize = baseTypeSize;
  return v;
}
```

2. `void reserveV(vector *v, size_t newCapacity)` – изменяет количество памяти, выделенное под хранение элементов вектора. Пример, когда новая вместимость больше:

Код программы:

```
// изменяет количество памяти, выделенное под хранение элементов вектора.
void reserveV(vectorVoid *v, size_t newCapacity) {
  if (newCapacity == 0) {
    v->data = NULL;
    v->size = 0;
  } else {
    if (newCapacity < v->size)
      v->size = newCapacity;
    v->data = (void *) realloc(v->data, newCapacity * v->baseTypeSize);
  }
  v->capacity = newCapacity;
}
```

3. `void clear(vector *v)` – удаляет элементы из контейнера, но не освобождает выделенную память.

```
// Удаляет элементы из контейнера, но не освобождает выделенную память.
void clear(vector *v) {
  v->size = 0;
}
```

4. `void shrinkToFitV(vector *v)` – освобождает память, выделенную под неиспользуемые элементы

```
// освобождает память, выделенную под неиспользуемые элементы
void shrinkToFitV(vectorVoid *v) {
  v->data = (void *) realloc(v->data, v->size * v->baseTypeSize);
  v->capacity = v->size;
}
```

5. `void deleteVector(vector *v)` – освобождает память, выделенную вектору

```
// Освобождает память, выделенную вектору
void deleteVectorV(vectorVoid *v) {
  free(v->data);
  v->size = 0;
  v->capacity = 0;
  v->baseTypeSize = 0;
}
```

6. bool isEmptyV(vectorVoid *v) — Возвращает true, если вектор пустой. False в ином случае.

```
// Возвращает true, если вектор пустой. False в ином случае.
bool isEmptyV(vectorVoid *v) {
  return v->size == 0 ? true : false;
}
```

7. bool isEmptyV(vectorVoid *v) — Возвращает true, если вектор полный. False в ином случае.

```
// Возвращает true, если вектор полный. False в ином случае.
bool isFullV(vectorVoid *v) {
  return v->size == v->capacity ? true : false;
}
```

8. void getVectorValueV(vectorVoid *v, size_t index, void *destination) – записывает по адресу destination index-ый элемент вектора v.

```
void getVectorValueV(vectorVoid *v, size_t index, void *destination) {
  if (index >= v->capacity)
    fprintf(stderr, "index > capacity :'( ");
  char *source = (char *) v->data + index * v->baseTypeSize;
  memcpy(destination, source, v->baseTypeSize);
}
```

Дополнительные тесты:

```
void test_getVectorValueV() {
  vectorVoid v = createVectorV(5, sizeof(int));
  int source[] = {1, 2, 3, 4, 5};
  for (int i = 0; i < 5; i++)
    setVectorValueV(&v, i, &source[i]);

  assert(isEqualArray(v.data, v.size, source, 5));
  deleteVectorV(&v);
}
```

9. void setVectorValueV(vectorVoid *v, size_t index, void *source) – записывает на index-ый элемент вектора v значение, расположенное по адресу source;

```
void setVectorValueV(vectorVoid *v, size_t index, void *source) {
  if (index >= v->capacity)
    fprintf(stderr, "index > capacity :'( ");
  char *destination = (char *) v->data + index * v->baseTypeSize;
  memcpy(destination, source, v->baseTypeSize);
}
```

Дополнительные тесты:

```
void test_setVectorValueV() {
  vectorVoid v = createVectorV(5, sizeof(int));
  int source[] = {1, 2, 3, 4, 5};
  for (int i = 0; i < 5; i++)
    setVectorValueV(&v, i, &source[i]);

  assert(isEqualArray(v.data, v.size, source, 5));
  deleteVectorV(&v);
}
```

10. void popBackV(vectorVoid *v)

```
void popBackV(vectorVoid *v) {
  if (isEmptyV(v)) {
    fprintf(stderr, "vector is empty");
    exit(1);
  }
  (v->size)--;
}
```

11. void pushBackV(vectorVoid *v, void *source)

```c
void pushBackV(vectorVoid *v, void *source) {
  if (isFullV(v))
    reserveV(v, v->capacity == 0 ? v->capacity + 1 : v->capacity * 2);
  setVectorValueV(v, v->size++, source);
}
```

```
commit 041dfed530822ba48b638418f08b728d0e52303e (HEAD -> master, origin/master)
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Wed Feb 9 00:32:23 2022 +0300

    Specification

 libs/data_structures/vector/voidVector.c | 10 ++++++++--
 1 file changed, 8 insertions(+), 2 deletions(-)

commit 1da3c493317c797f06a51c5947bff54ce2f02602
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 8 20:22:40 2022 +0300

    refactoring + tests

 libs/data_structures/vector/voidVector.c | 14 ++++++++++----
 1 file changed, 10 insertions(+), 4 deletions(-)

commit b35665eed7cc98605fae01935ee35892424c18c5
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Sun Feb 6 20:32:37 2022 +0300

    append push / pop functions

 libs/data_structures/vector/voidVector.c | 51 ++++++++++++++++++++++++++------
 1 file changed, 42 insertions(+), 9 deletions(-)

commit 929c54b8af9092df3554d83669e8655caf8b245d
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Sun Feb 6 14:35:14 2022 +0300

    memory usage of vectorVoid

 libs/data_structures/vector/voidVector.c | 42 ++++++++++++++++++++++++++++++
 1 file changed, 42 insertions(+)

commit 7c7cd919e3bb10fa8a91a303efeba4f480673f70
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Sun Feb 6 14:28:03 2022 +0300

    init vectorVoid

 libs/data_structures/vector/voidVector.c | 1 +
 1 file changed, 1 insertion(+)
```

Вывод: Благодаря этой прекрасной лабораторной работе разобрался с GitHub-ом и теперь я профессиональный разработчик ChuvilkoDEV ”типа черные очки →” B-).