Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Отчёт по лабораторной работе № 5

Дисциплина: Низкоуровневое программирование

Тема: программирование на языке C

Вариант: 5

Выполнил студент гр. 3530901/90002 _____Е. В. Бурков
(подпись)

Принял преподаватель ___ _____ Д. С. Степанов
(подпись)

“_____” _____ 2021 г.

Санкт-Петербург

2021

Содержание

Формулировка задачи

1.Разработать статическую библиотеку, реализующую определенный вариантом задания абстрактный тип данных.

2. Разработать демонстрационную программу – консольное приложение, обеспечивающее ввод данных из файла (файлов), их обработку и вывод в файл (файлы); имена файлов передаются в качестве параметров командной строки.

Требования к ПО

1. Язык разработки – C.

2. Реализация абстрактного типа данных должна использовать динамическое выделение памяти, при этом должна быть предусмотрена функция деинициализации, обеспечивающая освобождение всей выделенной памяти.

3. Библиотека и демонстрационная программа должны быть снабжены модульными тестами.

4. Разработанный исходный код должен компилироваться gcc без ошибок и предупреждений со следующими параметрами: -std=c11 -pedantic –Wall -Wextra.

5. Сборка библиотеки, демонстрационной программы и модульных тестов должна осуществляться утилитой make.

Вариант задания

Вариант 5: двоичная куча.

Двоичная куча или пирамида (англ. Binary heap) — такое двоичное подвешенное дерево, для которого выполнены следующие три условия:

1. Значение в любой вершине не больше (если куча для минимума), чем значения её потомков.
2. На i-ом слое 2^i вершин, кроме последнего. Слои нумеруются с нуля.
3. Последний слой заполнен слева направо (как показано на рисунке)

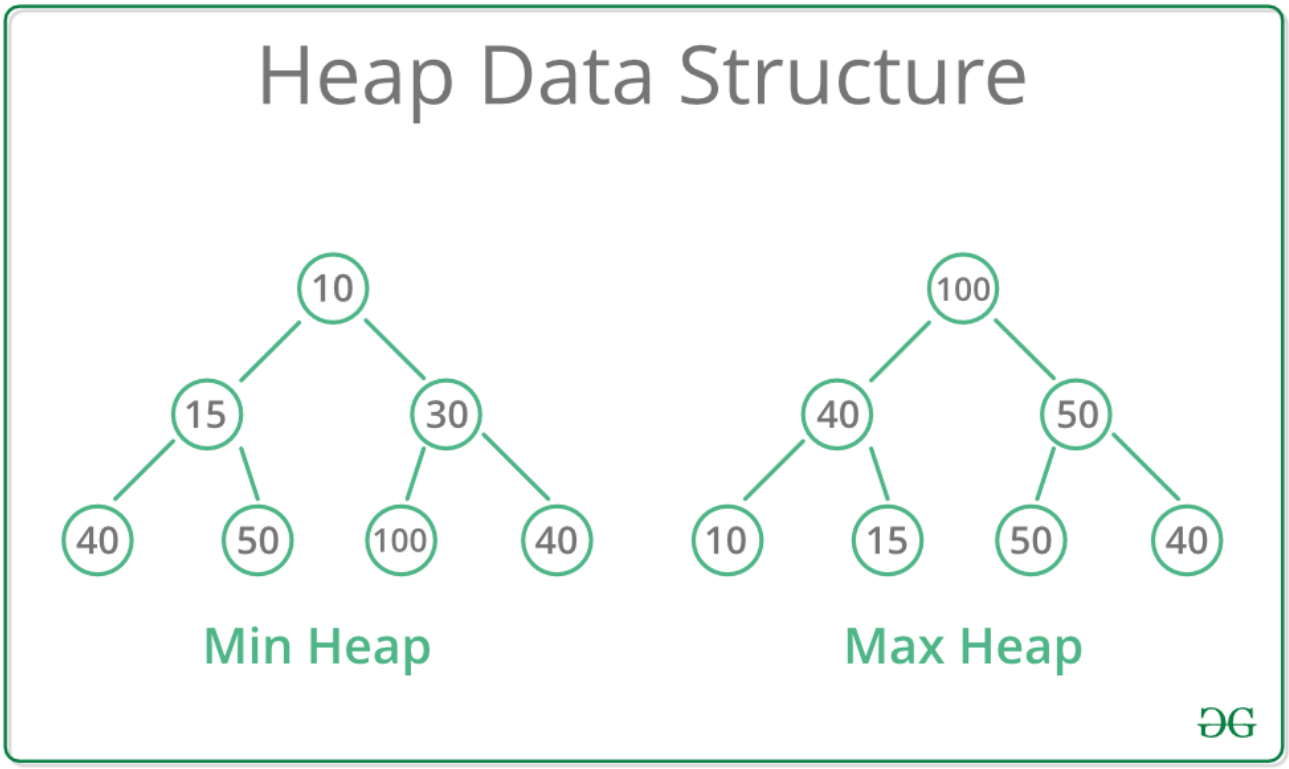

Рис. 1 Куча минимума и куча максимума

В библиотеке представлены все базовые процедуры, такие как: добавление в кучу, исключение минимального/максимального элемента и функции для восстановления свойств кучи. Используя данные базовые операции возможно реализовать что-то поинтересней. Например: пирамидальная сортировка, формирование кучи из массива за линейное время O(len(array)). Чаще всего кучи используют для извлечения минимума/максимума из набора значений, то есть частный случай приоритетной очереди.

Описание реализованной библиотеки

Реализуемая куча представляет из себя динамическую структуру данных. Хранение элементов происходит в динамическом массиве. Адресация будет следующая:

a[0] – корень (min/max element)

a[i] – предок элемента a[(i-1/2)]

a[2i+1], a[2i+2] – предки элемента a[i]

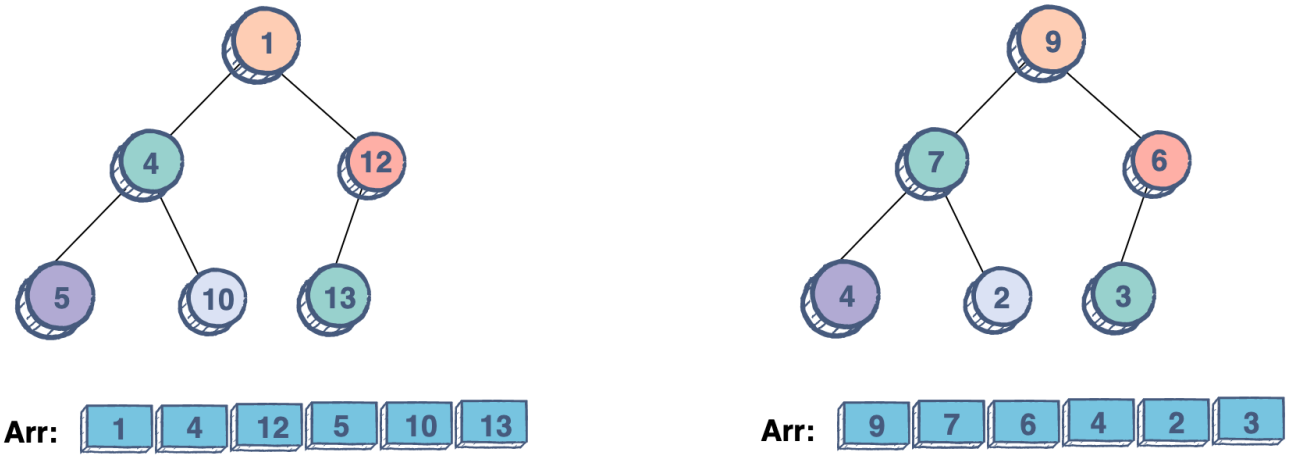

Рис. 2 Кучи и их представление в виде массива

В начале написания программы в заголовочном файле libHeap.h были написаны следующие строки:

```
typedef int key_heap;
typedef unsigned int value_heap;
```

В данном куске кода используется ключевое слово **typedef.** Данное слово позволяет задать нам собственное имя типа. В данном случае мы вводим синонимы для типов **int** и **unsigned int**. Может показаться что данная функция работает как директива #DEFINE, но это не так. Разница заключается в том, что директива #DEFINE выполняется препроцессором, а функция typedef выполняется на этапе компиляции.

Далее определим основные структуры.

```
typedef struct {
key_heap key; // ключ
value_heap value; // значение
} pair_heap;

typedef struct {
pair_heap *array; // указатель на массив пар
size_t size; // размер массива
unsigned int data; // размер инициализированной памяти для массива
bool max; // тип кучи
} heap;
```

Тут используется объединение нескольких объектов под одним именем – **struct**. По большому счёту структура позволяет расположить в памяти все объекты последовательно.

Также используется слово typedef. Мы могли бы опустить эту конструкцию, но тогда столкнулись бы с проблемой. На языке С есть два разных пространств имен типов: пространство имен тегов struct/union/enum, и пространство имен typedef. Если мы опустим typedef, то структура будет определена только в пространстве тегов, и каждый раз при декларации переменных, относящихся к этой структуре, необходимо было бы писать **struct pair_heap,** это может вызвать некоторые сложности и неудобство. Поэтому было выбрана именно эта структура, где мы декларируем безымянную структуру и определяем для нее имя в пространстве typedef.

Базовые операции над кучей

```
heap *heapInit(const unsigned int start_data)
```

Инициализация кучи. Выделение памяти для структуры heap, выделение памяти для массива пар с учётом пожелания пользователя (start data), установка размера в ноль. На выходе получаем указатель на кучу. Используется для других функций библиотеки

```
heap *maxHeap(const unsigned int start_data)
heap *minHeap(const unsigned int start_data)
```

Создание максимальной/минимальной бинарной кучи. Использует функцию heapInit. Дополнительно устанавливается тип кучи. На выходе получаем указатель на кучу.

P.S. Далее для удобства изложения представим что работаем с min кучей.

```
void heapShiftUp(heap *h, size_t i)
```

Всплывание элементов. Указывается индекс элемента в массиве кучи. Для него проверяется – если он больше отца, то свойство 1 сохраняется. Иначе меняем его местами с родителем и продолжаем итерироваться пока не дойдём до корня или соблюдения свойства кучи. Используется для извлечения корня из кучи.

```
void heapShiftDown(heap *h, size_t i)
```

Данный метод необходим для восстановления свойств кучи, когда при добавлении элемента в конец кучи его надо поднять наверх. В цикле сравниванием i-ый элемент с потомками. Если он больше потомка, то меняем местами. Также учтено свойство 3 (приоритет для левого потомка). Используется при добавлении элемента.

```
void heapAdd(heap *h, pair_heap p)
```

Добавление элемента в кучу. Алгоритм крайне прост – добавляем элемент в конец массива и вызываем для него всплывание, тем самым восстанавливаем

свойство кучи.

pair_heap heapRoot(heap *h)

Извлечение минимального элемента из кучи. Берём первый элемент из массива, сохраняем его в переменную. После перемещаем последний элемент из массива в начало и уменьшаем размер массива. После для восстановления свойств кучи используем просеивание для корня. После возвращаем переменную, в которую записан минимальный элемент.

void buildHeap(heap *h)

Внутренний метод библиотеки для формирования кучи из массива. Имеет асимптотику O(N). Инициализируем кучу с массивом. Свойство 1 не соблюдено. Вызываем heapShiftDown для всех пар, у которых есть потомки (от 0 до len(array) /2). Тем самым мы можем гарантированно сказать, что куча построена верно.

```
heap *minHeapArray(pair_heap *p, size_t size)
heap *maxHeapArray(pair_heap *p, size_t size)
```

Построение min/max кучи из массива пар.

void heapSort(pair_heap *array, size_t size)

Пирамидальная сортировка. Данная сортировка довольно проста: мы извлекаем корень (то есть max элемент) и ставим его в конец массива. Уменьшаем размер кучи на 1 и повторяем N раз. Тем самым получается что максимальные элементы каждую итерацию переходят в конец массив и в конце получаем отсортированный массив. Может рассматриваться как усовершенствованная сортировка пузырьком, в которой элемент всплывает (min-heap) / тонет (max-heap) по многим путям. Реализация получилась довольно лёгкий из-за того, что основные операции над кучей уже реализованы.

```
void heapRemove(heap *h)
```

Освобождение памяти, выделенной для кучи. Делаем проверки на существование выделенной памяти (указатель != NULL). Делается это для того, чтобы функция не освободила не выделенную нами память, что может привести к повреждению данным в куче памяти или сильной дефрагментации.

```
void printHeap(heap *h , FILE *file)
```

Печать кучи. Необходимо указать выходной поток.

Консольное приложение

Для демонстрации работы библиотеки была разработана консольная утилита heap. Она имеет следующий синтаксис для вызова:

$ heap [-min/-max] (operation) [-i input file] [-o output file]
Или
$ heap –help (-h)

При вызове приложения с аргументом –help на экран выводятся подсказки для пользователя:

```
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>heap -h
Heap sort [-i] [input file] [-o] [output file]
Sorting heap of integers - integers

Heap (-max|-min) tree [-i] [input file] [-o] [output file]
Print heap

Heap (-max|-min) extract [-i] [input file] [-o] [output file]
Deleting max or min key in heap
```

Рис. 3 Вызов печати доступных команд

При вызове tree и extract необходимо указывать какой тип кучи требуется. Так же во всех команда используются ключи -i и -o для указывания входных и выходных файлов. Если их опустить, то будет произведена работа со стандартными потоками (с консоли).При вводе с консоли необходимо заканчивать ввод любой буквой.

Heap sort

Отсортировать исходный целых чисел массив и вывести его. Используется пирамидальная сортировка. Значения в массиве могут разделяться или пробелами, или отступами.

```
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>type heap.txt
42 2
3 4
5 4
4 4
32 4
31 4
1024 4
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>heap.exe sort -i heap.txt
3 4
4 4
5 4
31 4
32 4
42 2
1024 4
```

Рис. 4 Примеры работы Heap sort

Heap tree

Сформировать кучу из массива данных вида (ключ + значение) и вывести дерево данной кучи. Ключ и значение отделяются пробелами, пары отделяются отступами или пробелами.

```
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>type heap.txt
42 2
3 4
5 4
4 4
32 4
31 4
1024 4
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>heap.exe -max tree -i heap.txt
\_1024 (4)_
    |_32 (4)_
    |   |_3 (4)_
    |   |
    |   |
    |   \_4 (4)_
    |
    |
    \_42 (2)_
        |_5 (4)_
        |
        |
        \_31 (4)_
```

Рис. 5 Пример работы Heap tree

Heap extract

Изъять корень из кучи. Может использоваться для нахождения максимума минимума набора данных.

```
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>type heap.txt
42 2
3 4
5 4
4 4
32 4
31 4
1024 4
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>heap.exe -max extract -i heap.txt
key=1024 value=4 was extracted
42 2
32 4
31 4
3 4
4 4
5 4

D:\projects\lowlevelprog\lowlevelprog\lab5\heap>heap.exe -min extract -i heap.txt
key=3 value=4 was extracted
4 4
32 4
5 4
42 2
1024 4
31 4
```

Рис. 6 Пример работы Heap extract

Модульные тесты

Для написания тестов отдельных функций и тестов на просторах гитхаба был найден фреймворк CuTest. Он представляет собой заголовочный файл и файл с методами. Использование оказалось крайне простым, необходимо переместить эти два файла в каталог проекта и включить их в компиляцию. Структура запусков тестов следующая:

```
#include "CuTest.h"

void Test (CuTest *tc) {
    CuAssertStrEquals(…);
}

CuSuite* StrUtilGetSuite() {
    CuSuite* suite = CuSuiteNew();
    SUITE_ADD_TEST(suite, Test);
    return suite;
}

CuSuite* StrUtilGetSuite();

void RunAllTests(void) {
    CuString *output = CuStringNew();
    CuSuite* suite = CuSuiteNew();

    CuSuiteAddSuite(suite, StrUtilGetSuite());

    CuSuiteRun(suite);
    CuSuiteSummary(suite, output);
    CuSuiteDetails(suite, output);
    printf("%s\n", output->buffer);
}

int main(void) {
    RunAllTests();
}
```

При помощи данного фреймворка были составлены тесты для основных функций библиотеки:

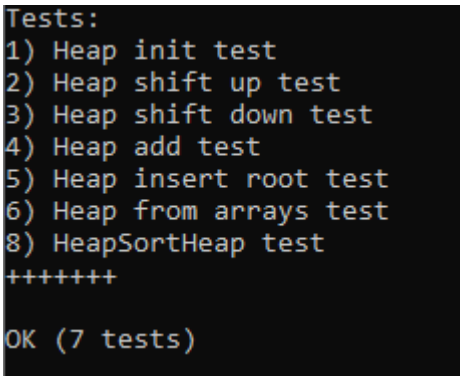

Рис. 7 Тестирование программы

Написание make-файлов

Для автоматизации сборки библиотеки, приложения и тестирования были написаны make-файлы. Основной файл лежит в корневом каталоге проекта.

Для сборки библиотеки необходимо выполнить:

```
$ make
или
$ make build
```

В ходе выполнения цели происходит выполнение make-файла из папки “src” компиляция файла “heap.o” и сборка статической библиотеки “libHeap.a”.

```
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>make
make -C src
make[1]: Entering directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
gcc -std=c11 -Wall -pedantic -Wextra -I../include  -c -o heap.o heap.c
ar rsc libHeap.a heap.o
make[1]: Leaving directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
```

Рис. 8 Вызов make

Можно заметить, что компиляция проходит с помощью тулчейна GNU Compiler Collection. Все флаги из условия работы указаны и предупреждений не наблюдается:

-std=cc1	Стандарт языка СИ
-Wall	"Агрегатор" базовых предупреждений
-pedantic	Проверяет соответствие кода стандарту ISO C++, сообщает об использовании запрещённых расширений, о наличии лишних точек с запятой, нехватке переноса строки в конце файла и прочих полезных штуках.
-Wextra	"Агрегатор" дополнительных предупреждений.

Для проверки правильно написания make-файла проведём эксперимент. Попробуем вызвать цель build дважды. По-хорошему, если файлы исходного кода не были изменены, то библиотеке не следует собираться ещё раз. Для этого используется опция **s**.

- s - Записывает индекс объектного файла в архив или, если он существует, обновляет его, даже если нет других изменений в архиве.

```
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>make build
make -C src
make[1]: Entering directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
gcc -std=c11 -Wall -pedantic -Wextra -I../include  -c -o heap.o heap.c
ar rsc libHeap.a heap.o
make[1]: Leaving directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'

D:\projects\lowlevelprog\lowlevelprog\lab5\heap>make build
make -C src
make[1]: Entering directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
make[1]: 'libHeap.a' is up to date.
make[1]: Leaving directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
```

Рис. 9 Вызов make без изменения исходников

Можно заметить строку “ 'libHeap.a' is up to date. ”, что свидетельствует о том, что собирать заново библиотеку не потребовалось. Изменим исходный файл (добавим глобальную переменную или уберём модификатор static где-нибудь) и посмотрим на действия мейка:

```
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>make build
make -C src
make[1]: Entering directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
make[1]: 'libHeap.a' is up to date.
make[1]: Leaving directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'

D:\projects\lowlevelprog\lowlevelprog\lab5\heap>make build
make -C src
make[1]: Entering directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
gcc -std=c11 -Wall -pedantic -Wextra -I../include  -c -o heap.o heap.c
ar rsc libHeap.a heap.o
make[1]: Leaving directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
```

Рис. 10 Вызов make с изменением исходников

Как видно, исходный файл скомпилировался заново, и библиотека была пересобрана. Далее рецепты были построены похожим образом и проблемы повторной ненужной компиляции отпадает.

Сборка консольного приложения:

```
$ make app
```

```
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>make app
make -C src Heap
make[1]: Entering directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
gcc -std=c11 -Wall -pedantic -Wextra -I../include  -c -o main.o main.c
gcc -std=c11 -Wall -pedantic -Wextra -I../include  -c -o heap.o heap.c
ar rsc libHeap.a heap.o
gcc -std=c11 -Wall -pedantic -Wextra -I../include main.o libHeap.a -o Heap
move Heap.exe ../
Перемещено файлов:         1.
make[1]: Leaving directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
```

Рис. 11 Результат вызова make app

Можно подметить, что для сборки приложения сначала необходимо создать библиотеку, после скомпилировать объектный файл main.o в .exe . В конце приложение перемещается в корневую папку для удобства.

Модульные тесты:

```
$ make test
```

```
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>make test
make -C test
make[1]: Entering directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/test'
gcc -std=c11 -Wall -pedantic -Wextra -I../include -c -o test.o test.c
gcc -std=c11 -Wall -pedantic -Wextra -I../include -c -o CuTest.o CuTest.c
gcc -std=c11 -Wall -pedantic -Wextra -I../include -c -o ../src/heap.o ../src/heap.c
gcc -std=c11 -Wall -pedantic -Wextra -I../include -o test.exe test.o CuTest.o ../src/heap.o
test.exe
Tests:
1) Heap init test
2) Heap shift up test
3) Heap shift down test
4) Heap add test
5) Heap insert root test
6) Heap from arrays test
8) HeapSortHeap test
+++++++

OK (7 tests)
```

Рис. 12 Прохождение тестов

Для запуска тестов не надо собирать библиотеку. Это связано с тем, что тестируются функции, которые не входят в её API.

Вызов демонстрационной программы:

```
$ make demo
```

Тут было решено просто собрать программу и выполнить с ней пару действий:

```
D:\projects\lowlevelprog\lowlevelprog\lab5\heap>make demo
make -C src Heap
make[1]: Entering directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
gcc -std=c11 -Wall -pedantic -Wextra -I../include  -c -o main.o main.c
gcc -std=c11 -Wall -pedantic -Wextra -I../include  -c -o heap.o heap.c
ar rsc libHeap.a heap.o
gcc -std=c11 -Wall -pedantic -Wextra -I../include main.o libHeap.a -o Heap
move Heap.exe ../
Перемещено файлов:         1.
make[1]: Leaving directory 'D:/projects/lowlevelprog/lowlevelprog/lab5/heap/src'
Working with file "samples/heap"
Max heap
```

Рис. 13 Демонстрационная программа

Руководство программиста

Для использования библиотеки и приложения необходимо клонировать git репозиторий проекта к себе на машину.

```
$ git clone https://github.com/wooftown/lowlevelprog.git
```

При необходимости изменить в переменную $(RM) в мейкфайле (для пользователей UNIX).

Для сборки библиотеки:

```
$ make
```

Для сборки приложения:

```
$ make app
```

Для запуска демонстрационной программы:

```
$ make demo
```

Для выполнения модульных тестов:

```
$ make test
```

Для отчистки от файлов созданных в ходе работы мейк-файлов:

```
$ make clean
```

Clean так же можно вызывать из отдельных мейк-файлов в каталогах проекта.

Вывод

В процессе выполнения лабораторной работы была реализована статически линкуемая библиотека с функциональностью бинарной кучи. Была реализована консольная утилита, использующая нашу библиотеку. Были написаны модульные тесты. Реализованы мейк-файлы для сборки библиотеки, приложения, выполнения тестов и демонстрации работы программы. Все файлы представлены в репозитории https://github.com/wooftown/lowlevelprog .

Список использованной литературы и источников

- Язык программирования Си 3-е издание Брайан Керниган, Деннис Ритчи
- C Interfaces and Implementations: Techniques for Creating Reusable Software by David Hanson
- https://neerc.ifmo.ru/wiki/index.php?title=Двоичная_куча
- https://github.com/ennorehling/cutest
- https://habr.com/ru/post/490850