Сухова Анна БПИ207

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

1. Описание задания

В рамках задания необходимо:

1. Использовать:

• вычислительную систему с архитектурой x86-64;

• операционную систему Linux;

• язык программирования C++;

2. Разработать программу в виде консольного приложения, ориентированную на объектно-ориентированный подход с использованием статически типизированного универсального языка программирования.

Запуск программы должен осуществляться из командной строки, в которой указываются: имя запускаемой программы; имя файла с исходными данными; имя файла с выходными данными.

Примеры:

command -f infile.txt outfile1.txt outfile2.txt (для ввода из файла)

command -n number_of_elements outfile1.txt outfile2.txt (для случайной генерации)

3. Реализовать в программе следующий функционал (**вариант 13**):

Обобщённый артефакт	Базовые альтернативы (и их отличительные параметры)	Общая для всех альтернатив переменная	Общая для всех альтернатив функция
Растение	1. Деревья (возраст – длинное целое) 2. Кустарники (месяц цветения – перечислимый тип)	Название – строка символов (макс. длина = 20 символов)	Частое от деления числа гласных букв в названии на общую длину названия

	3. Цветы (домашние, садовые, дикие – перечислимый тип)		

4. Поместить данные объекты в контейнер и в соответствии с вариантом задания (**вариант 19**) удалить из контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, меньше чем среднее арифметическое для всех элементов контейнера, полученное с использованием той же функции.

5. Провести отладку и тестирование разработанной программы на заранее подготовленных тестовых наборах данных (не менее 5). Тестовые данные с большим числом элементов должны порождаться программой с использованием генераторов случайных наборов данных. Управление вводом данных задается из командной строки.

6. Создать отчёт по выполненному заданию, описав структуру используемой ВС с наложением на нее обобщённой схемы разработанной программы и зафиксировав основные характеристики программы.

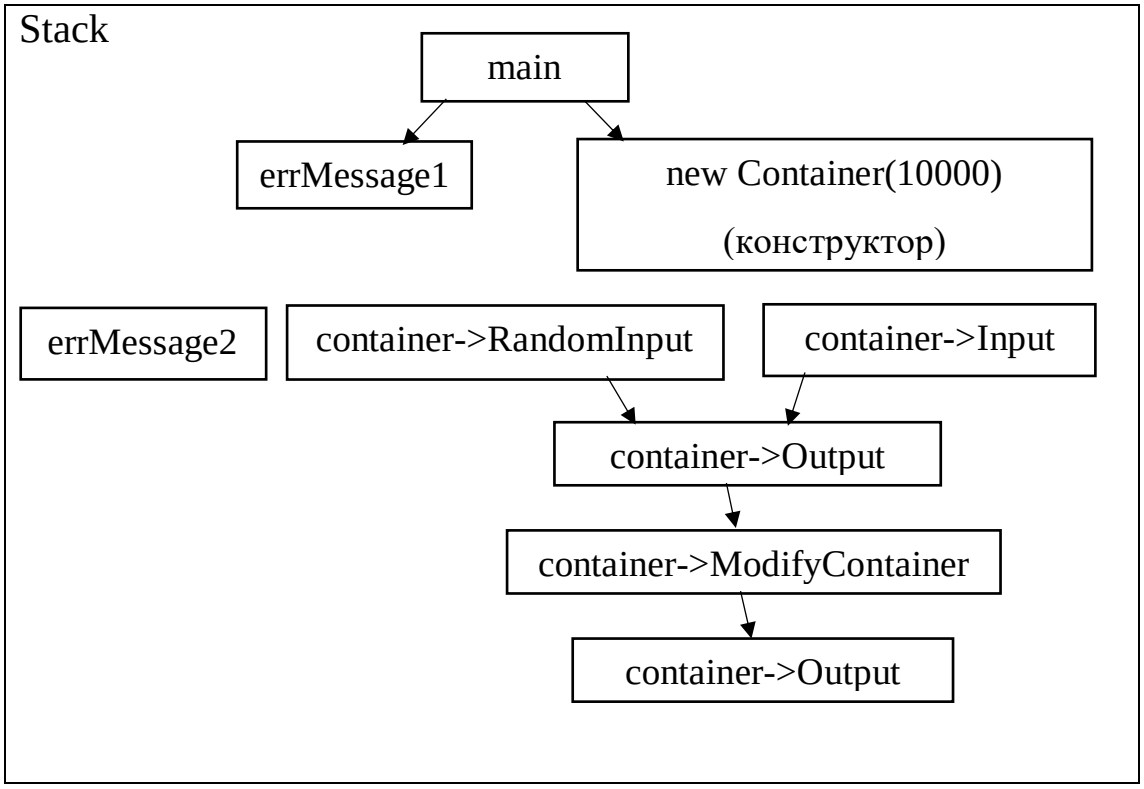
7. Привести результаты сравнительного анализа полученных характеристик с теми, которые были получены для предыдущей программы. Сделать выводы о достоинствах и недостатках этого и предшествующего решения относительно друг друга.

2. Схема архитектуры ВС с размещённой на неё разработанной программой.

1. Схема памяти функции main.

Память программы

int main(int argc, char* argv[]) {
 	
…
 	
}

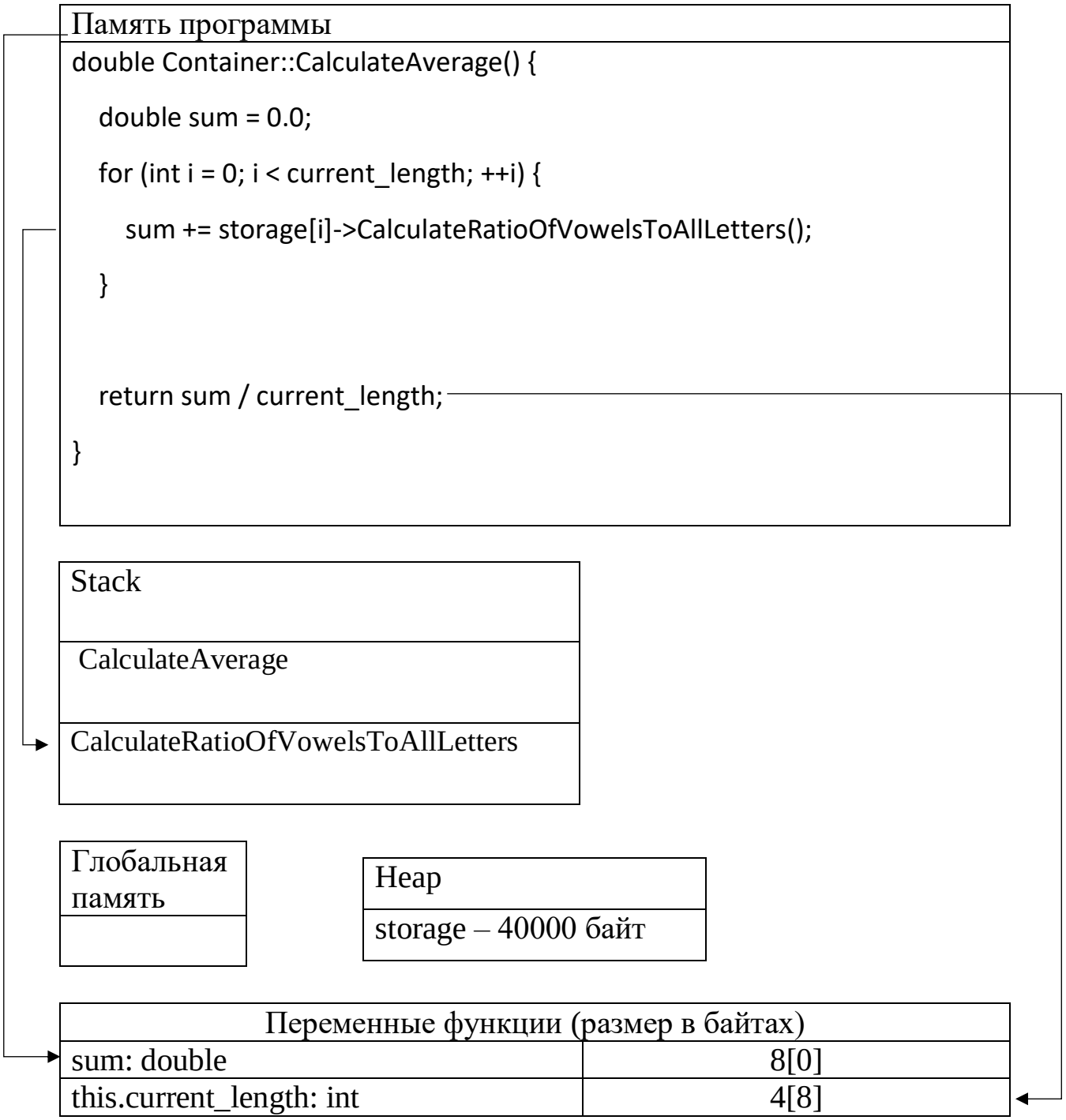


Переменные функции (размер в байтах)	
argv: int	4[0]
argc: char**	4[4]
container: *Container	4[8]
file_mode: string	28[12]
size: int	4[40]
ifstream: ifstream	184[44]
ofstream1: ofstream	176[228]
ofstream2: ofstream	176[404]

Глобальная память

Heap
container – 40000 байт

2. Схема памяти функции по подсчёту среднего значения.



3. Схема памяти функции по модификации контейнера в соответствии с вариантом 19.

Память программы

```
void Container::ModifyContainer() {
   double average = CalculateAverage();
   int i = 0;

   while (i < current_length) {
      if (storage[i]->CalculateRatioOfVowelsToAllLetters() < average) {
         // Удаление элемента.
         for (int j = i; j < current_length - 1; j++) {
            storage[j] = storage[j + 1];
         }
         --current_length;
      } else {
         ++i;
      }
   }
}
```

Stack

ModifyContainer

CalculateAverage

CalculateRatioOfVowelsToAllLetters

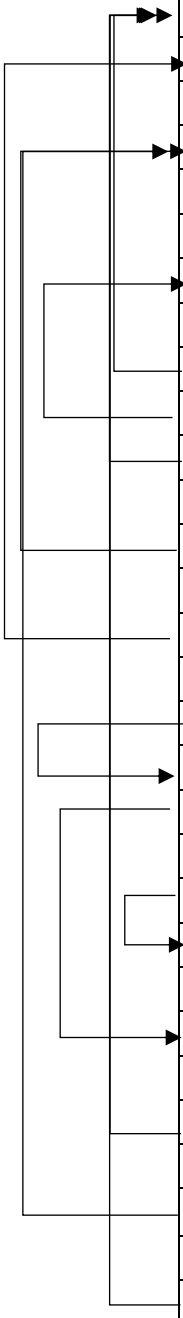
Heap

storage – 40000 байт

Глобальная память

Переменные программы (размер в байтах)	
average: double	8[0]
i: int	4[8]
j: int	4[12]
this.current_length: int	4[16]

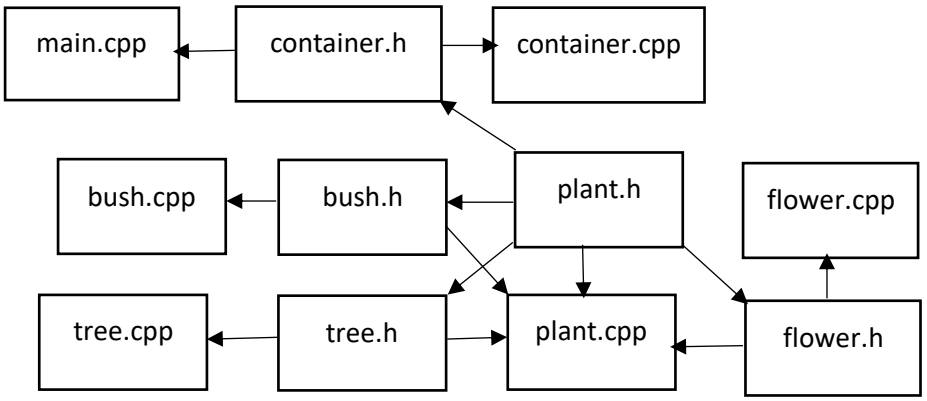
Таблица типов:



Тип	**Размер (байт)**
int	4
long	4
double	8
string	28
Container	12
*Container	4
*Plant	4
class Container	40008
current_length: int	4[0]
storage: **Plant	40000[4]
size: int	4[40004]
class Plant	32
name: string	28[0]
class Tree	36
age: long	4[0]
class Flower	36
placement_type: flower_key	4[0]
enum flower_key {}	4[4]
placement_types: char* [3]	12[8]
class Bush	36
flowering_month: bush_key	4[0]
enum bush_key {}	4[4]
char*	4
char	1
argv: int	4
argc: char**	4
file_mode: string	28
container: *Container	4
size: int	4
ofstream1: ofstream	176

ofstream2: ofstream	176
ifstream: ifstream	184

Файловая схема:



3. Основные характеристики программы:

Число заголовочных файлов (внутренних): 5
Число заголовочных файлов (библиотек): 4 (ctime, string, iostream, fstream)
Число модулей: 6
Общий размер исходных текстов (программы): 14,0125 Кбайт
Полученный размер исполняемого кода:
36,8 Кбайт (исполняемый файл Linux)
112 Кбайт (.exe)

Время выполнения программы для тестовых наборов данных:

№ теста	Время (в секундах)	Время в задании №1 (процедурный подход)
1	0.000982	0.001703
2	0.000843	0.00148
3	0.000945	0.002179
4	0.00132	0.01543
5	0.1245	0.16013
6	0.038996	0.081633
7	0.00443	0.014050
8	0.000752	0.001601
9	0.11506	0.332784

4. Сравнительный анализ с предыдущей программой (из задания №1)

При сравнении программ на C++, решающих одну задачу, но с использованием разных подходов (объектно-ориентированный и процедурный), можно выделить следующие различия:

1. По таблице времени выполнения программы для тестовых наборов данных видно, что на одинаковых тестах программа с объектно-ориентированным подходом выполняется быстрее, чем программа с процедурным подходом, следовательно, объектно-ориентированный подход более эффективен для решения данной задачи.
2. Для описания данных сущностей (растение, дерево, цветок, кустарник, контейнер) при использовании ООП используются классы, между тем как при процедурном подходе используются структуры.
3. Благодаря наследованию и полиморфизму код с объектно-ориентированным подходом получается более читабельным (видно, что цветок, кустарник, дерево наследуются от класса растения => являются растениями и получают функционал родителя, но с разной его реализацией).
4. Размер исполняемого кода при объектно-ориентированном подходе больше примерно на 13 Кбайт (файл .exe больше примерно на 64 Кбайт).
5. Схемы подключения заголовочных файлов различаются (файловая схема, пункт 2), так как при объектно-ориентированном подходе к файлам классов-наследников подключается заголовочный файл класса-родителя.
6. Количества заголовочных файлов и модулей одинаковы у обеих программ.

Таким образом, программа, ориентированная на объектно-ориентированный подход, более эффективна по времени и читабельна, но занимает больше памяти, чем программа, разработанная с использованием процедурного подхода.