**Описание полученного задания**

Необходимо написать программу, оформленную в виде консольного приложения. Запуск программы осуществляется из командной строки, в которой указываются имя запускаемой программы, имя файла с исходными данными, имя файла с выходными данными.

Для каждого программного объекта, загружаемого в контейнер, исходный файл с тестовым набором должен содержать признак альтернативы, а также список параметров, необходимых альтернативе. В выходной файл необходимо вывести введенные в контейнер данные. Помимо этого, необходимо вывести информацию об общем количестве объектов, содержащихся в контейнере. После этого в тот же файл необходимо вывести новые данные в соответствии с результатами, полученными в ходе работы программы.

Основное задание для составления программы – создание обобщенного артефакта – языки программирования – со следующими базовыми альтернативами, имеющими уникальные параметры: процедурные языки программирования (уникальные параметры: наличие/отсутствие абстрактных типов данных [булевская величина]), объектно-ориентированные языки программирования (уникальные параметры: виды наследования: одинарное, множественное, интерфейс [перечислимый тип]), функциональные языки программирования (уникальные параметры: типизация: строгая, динамическая [перечислимый тип]; поддержка «ленивых» вычислений [булевская величина]). Все базовые альтернативы имеют общие переменные: название [строка символов], популярность в процентах [действительное число], год создания [целое число]. Также все базовые альтернативы имеют общую для всех функцию: частное от деления года создания на количество символов в названии.

После размещения данных в контейнер необходимо упорядочить элементы контейнера по убыванию, используя сортировку с помощью прямого слияния (Straight Merge). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив. В программе должны поддерживаться следующие операции: заполнение контейнера данными, поступающими из входного потока (файла); вывод значений всех элементов в выходной поток (файл), выводятся параметры элементов, размещенных в контейнере.

**Таблица типов**

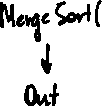
|  |  |
| --- | --- |
| **Тип данных** | **Размер** |
| int | 4 |
| float | 4 |
| char | 1 |
| class Procedural | 1 |
| bool:abstract\_data\_types | 1 [0] |
| class Functional | 5 |
| tipization:k | 4 [0] |
| bool:lazy\_calculations | 1 [4] |
| class Object-oriented | 4 |
| inheritance:k | 4 [0] |
| class Container | 1080004 |
| int:len | 4 [0] |
| Language\*:cont[max\_len] | 10000\*108 [4] |
| class Language | 108 |
| int:creat\_year | 4 [0] |
| float popularity\_perc | 4 [4] |
| char[]:name [100] | 100\*1=100 [8] |

Глобальная память не используется.

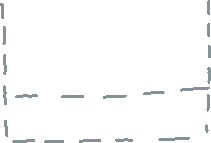
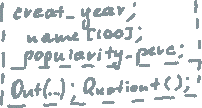
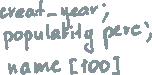
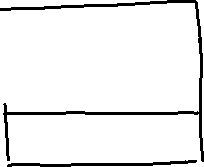
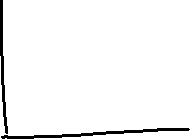
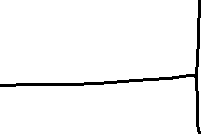
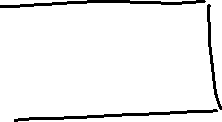
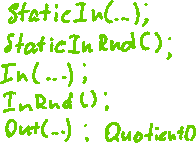
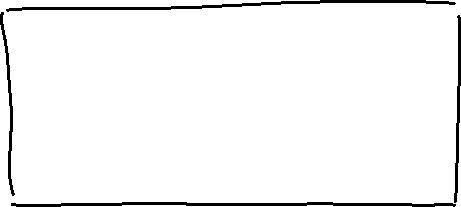
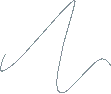
**Таблица данных функции main**

|  |  |
| --- | --- |
| **Данные** | **Размер** |
| int:argc | 4 [0] |
| char\*[]:argv | 1 [4] |
| Container:c | 40004 [5] |
| ifstream:ifst | 184 [40009] |
| int:size | 4 [40193] |
| ofstream:ofst1 | 176 [40197] |
| ofstream:ofst1 | 176 [40373] |

**Стек**



**Иерархия классов**



**Таблица данных функции StaticInRnd()**

|  |  |
| --- | --- |
| **Данные** | **Размер** |
| Language\*:sp | 4 [0] |
| int:k | 4 [4] |

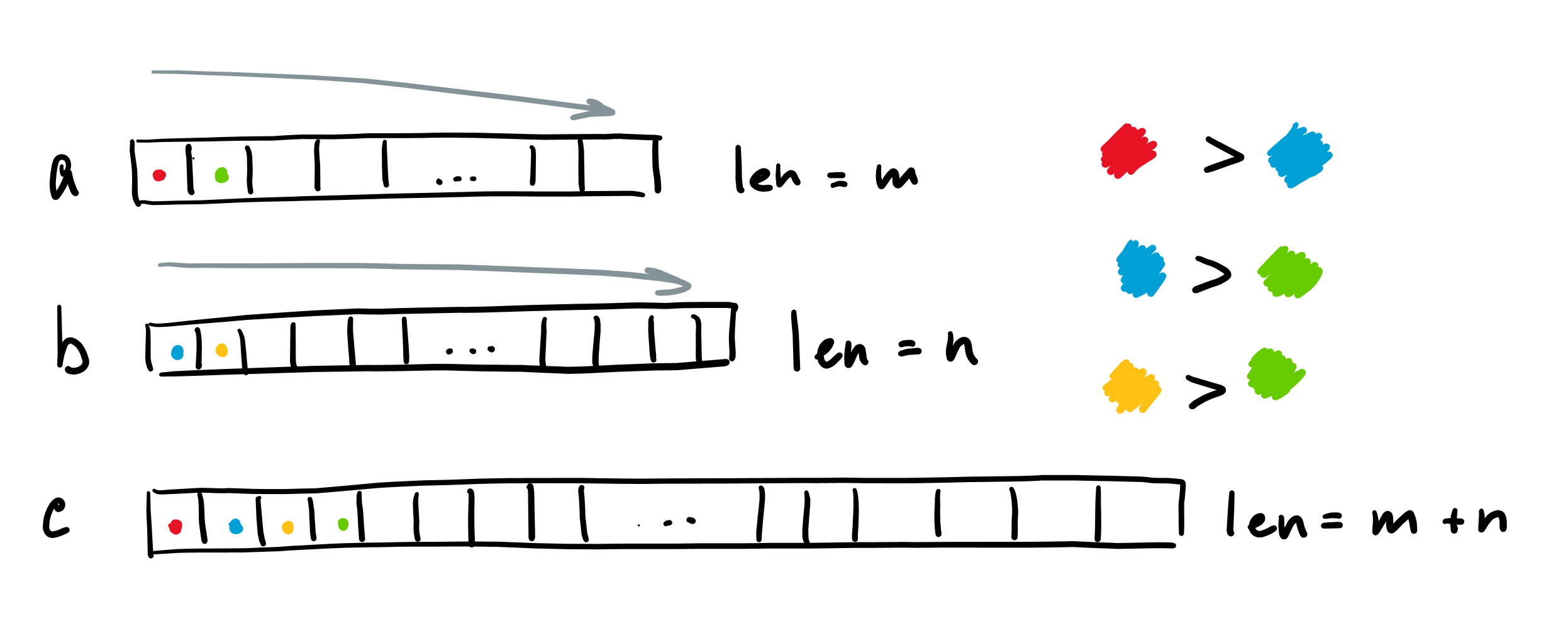
**Таблица данных функции StaticIn(int k, ifstream &ifstr)**

|  |  |
| --- | --- |
| **Данные** | **Размер** |
| int:k | 4 [0] |
| ifstream:ifstr | 184 [4] |
| Language\*:sp | 4 [188] |

**Таблица данных функции Merge(container \*c, int first, int last)**

|  |  |
| --- | --- |
| **Данные** | **Размер** |
| container:c | 4 [0] |
| int:first | 4 [4] |
| int:last | 4 [8] |
| int:middle | 4 [12] |
| int:start | 4 [16] |
| int:final | 4 [20] |
| int:j | 4 [24] |
| Container\*:mas | 4 [28] |

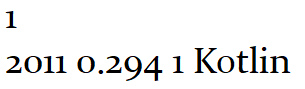
**Принцип алгоритма сортировки контейнера методом слияния:**



**Требуемые метрики, определяющие характеристики программы, для различных тестовых прогонов:**

*Пример* тестовых входных данных:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



*Изображение 1*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Замечание*: в первой строке указывается номер базовой альтернативы в диапазоне от 1 до 3 (1 – процедурный язык программирования; 2 – объектно-ориентированный язык программирования; 3 – функциональный язык программирования). В следующей строке через пробел вводятся параметры альтернативы (уникальные для каждой) в следующем порядке:

1. *процедурный язык программирования*: год создания (целочисленный тип в диапазоне от 1940 до 2020), процент популярности (вещественный тип в диапазоне от 0 до 100), наличие абстрактных типов данных (булевский тип, который может принимать значение 1 – true, 0 – false), имя (строка, начинающаяся с заглавной буквы, не превышающая длину 100).
2. *объектно-ориентированный язык программирования*: вид наследования (целочисленный ключ в диапазоне от 1 до 3, где 1 – одиночное наследование, 2 – множественное наследование, 3 – наследование интерфейса), год создания (целочисленный тип в диапазоне от 1940 до 2020), процент популярности (вещественный тип в диапазоне от 0 до 100), имя (строка, начинающаяся с заглавной буквы, не превышающая длину 100).
3. *функциональный язык программирования*: вид типизации (целочисленный ключ в диапазоне от 1 до 2, где 1 – строгая типизация, 2 – динамическая типизация), год создания (целочисленный тип в диапазоне от 1940 до 2020), процент популярности (вещественный тип в диапазоне от 0 до 100), наличие «ленивых» вычислений (булевский тип, который может принимать значение 1 – true, 0 – false), имя (строка, начинающаяся с заглавной буквы, не превышающая длину 100).

В примере (*Изображение 1*) приведен процедурный язык программирования, год создания 2011, процент популярности = 0.294, наличие абстрактных типов данных = true, название – Kotlin.

Все составленные тесты лежат в .txt файлах test1.txt, test2.txt, test3.txt, test4.txt, test5.txt, test6.txt, test7.txt, test8.txt, test9.txt, test10.txt. Файлы для вывода результатов работы программы – result.txt.out и result2.txt.out

**Основные характеристики программы**

Время работы тестовых данных:

1. **Тест 1:**

*2*

*2 1991 9.7 Python*

*3*

*2 1958 0.3 1 Lisp*

*1*

*2011 0.294 1 Kotlin*

Время работы = 3ms

1. **Тест 2:**

*3*

*1 2005 0.232 1 F#*

*2*

*2 1991 9.7 Python*

*2*

*1 2000 5.3 C#*

*3*

*2 2004 0.378 1 Scala*

Время работы = 3ms

1. **Тест 3:**

*1*

*2009 0.9 1 Go*

*2*

*1 2000 5.3 C#*

*1*

*2011 0.294 1 Kotlin*

*3*

*2 2004 0.378 1 Scala*

Время работы = 3ms

1. **Тест 4**

*1*

*2011 0.294 1 Kotlin*

*2*

*1 2000 5.3*

*C#*

Время работы = 3ms

1. **Тест 5**

*3*

*2 2004 0.378 1 Scala*

*3*

*1 2005 0.232 1 F#*

*2*

*2 1991 9.7 Python*

*3*

*2 1958 0.3 1 Lisp*

*1*

*2011 0.294 1 Kotlin*

Время работы = 8ms

1. **Тест 6:**

*2*

*2 1991 9.7 Python*

*1*

*1980 0.62 1 Ada*

*3*

*2 1958 0.3 1 Lisp*

*3*

*2 1995 11.68 1 Java*

*1*

*2011 0.294 1 Kotlin*

Время работы = 4ms

1. **Тест 7:**

*1*

*1980 0.62 1 Ada*

*2*

*1 1995 1.2 Ruby*

*2*

*2 1991 9.7 Python*

*3*

*2 1958 0.3 1 Lisp*

*3*

*2 2004 0.378 1 Scala*

*3*

*1 2005 0.232 1 F#*

Время работы = 5ms

1. **Тест 8:**

*3*

*1 2005 0.232 1 F#*

*2*

*1 1995 1.2 Ruby*

*2*

*1 2000 5.3 C#*

*3*

*2 2004 0.378 1 Scala*

*1*

*1980 0.62 1 Ada*

Время работы = 4ms

1. **Тест 9:**

*1*

*1980 0.62 1 Ada*

*3*

*2 1995 11.68 1 Java*

*2*

*1 1995 1.2 Ruby*

Время работы = 3ms

1. **Тест 10:**

*1*

*2020 30.33 1 NewLang2.0*

*3*

*1 1999 1.003 1 Lang2.0*

*3*

*2 1998 2.0333 0 Flang*

*2*

*1 2017 3.8752 OOP-new-Lang*

*1*

*2019 20.22 1 NewLang2.0*

Время работы = 4ms

Время работы для рандомного формирования входынх данных в зависимости от их количества:

1: 1ms

2: 4ms

3: 4ms

4: 4ms

5: 5ms

Число интерфейсных модулей = 6

Число модулей реализации = 6

Общий размер исходных текстов = 26Кб

Полученный размер исполняемого кода = 110Кб

**Сравнительный анализ по полученным характеристикам с реализациями, сделанными в других заданиях**

В текущей работе код реализуется на основе парадигмы объектно-ориентированного программирования, в то время как в предыдущей работе рассматривался процедурный подход. Это значит, что в данной работе используется наследование, которое не использовалось в прошлой работе. Наследование реализовано в классе Language, от которого наследуются классы Object-oriented, Functional и Procedural, они же и реализуют функции класса-родителя. При процедурном подходе все данные процедуры доступны лишь внутри нее, их нельзя вызвать из другого места программы. В данном случае преимущество объектно-ориентированного подхода заключается в том, что нет необходимости копировать аналогичный код в разные места программы (например, общие для всех альтернатив параметры не приходилось копировать в каждый класс, достаточно было описать их в классе-родителе).

Стоит отметить, что формат тестовых входных данных не изменился.

Время работы тестовых данных изменилось следующим образом:

|  |  |  |
| --- | --- | --- |
| Время работы для теста № | Работа 1 | Работа 2 |
| 1 | 3 | 3 |
| 2 | 5 | 3 |
| 3 | 4 | 3 |
| 4 | 6 | 3 |
| 5 | 18 | 8 |

Также к данной работе были добавлены новые тесты. После тестирования работы №1 на новых (добавленных) тестах 6, 7, 8, 9, 10 была составлена следующая сравнительная таблица:

|  |  |  |
| --- | --- | --- |
| Время работы для теста № | Работа 1 | Работа 2 |
| 6 | 4 | 4 |
| 7 | 6 | 5 |
| 8 | 4 | 4 |
| 9 | 3 | 3 |
| 10 | 5 | 4 |

Также было отмечено, что число интерфейсных модулей и модулей реализации не изменились, в то время как общий размер исходных текстов уменьшился на 5Кб, а полученный размер исполняемого кода увеличился на 60Кб.

Также можно отметить, что параметры *int creat\_year*, *float popularity\_perc*, *char name[100]* были проинициализированы в классе-родителе Language, и теперь можно явно выделить общие параметры базовых альтернатив и их уникальные параметры. Также общие для всех альтернатив функции вывода (*void Out(FILE \*outfile)*) и функция расчета частного от деления года создания на количество символов (*float Quotient()*) также содержатся в классе-родителе, так как их суть для каждой альтернативы одинакова.