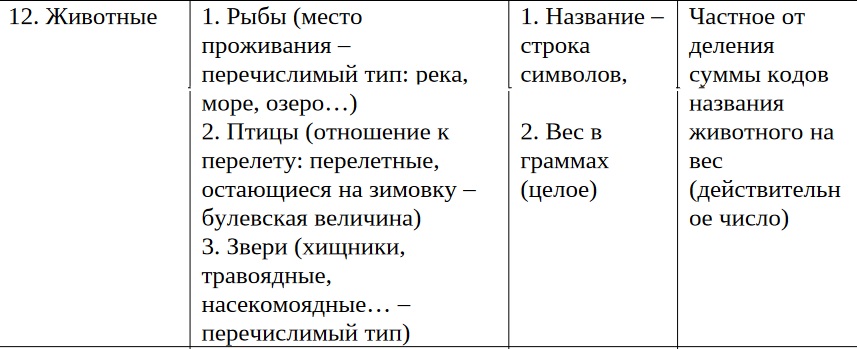
**Пояснительная записка для 2 домашнего задания по предмету «Архитектура вычислительных систем»**

**Выполнил: Ланин Георгий Михайлович БПИ202**

* **Вариант задания: 54**
* **Номер задачи: 12**
* **Номер функции: 4**



Функция: 4. Упорядочить элементы контейнера по возрастанию используя шейкерную сортировку (Shaker Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

Общий вид изложения данных в тесте: (вес – целое число!)

1. <тип животного> <имя> <вес> <доп. поле>

(Тип указывается цифрой: <Bird> = 1 <Fish> = 2 <Beast> 3)

1. Доп.поле для Fish: 1 (RIVER), 2 (SEA) или 3 (LAKE)
2. Доп.поле для Bird: true (Может летать), если не true, то не может летать
3. Доп.поле для Beast: 1 (DOMESTIC), 2(WILD) или 3 (GARDEN)

**Проект поддерживает сборку в cmake, iostream и fstream не используются**.

**Таблица типов**

|  |  |
| --- | --- |
| Типы | Память, (байты) и стартовые точки индексов |
| int | 4 |
| double | 8 |
| class Fish  enum key | 4  4[0] |
| class Beast  enum key | 4  4[0] |
| class Bird  bool isFlight | 1  1[0] |
| class Animal  int lengthOfName  int weight;   double quotient;  string name; | 20  4[0]  4[4]  8[8]  4[12] |
| class Container  int currentLen  Animal \*cont | 80008  4[0]  8\*10000 = 80000[4] |

**Память программ**

Справа: Память, (байты) и стартовые точки индексов

|  |  |
| --- | --- |
| main()  int args | 4[0] |
| Тут возможен вызов errMessage1() | |
| char\*[5] argv  Container container | 20[4] (5\*4 = 20)  80008[24] |
| Если читаем файл | |
| FILE \*input | 8[80008] |
| Container in(FILE \*input)  Animal::staticIn(input)  int type;  char charName [100];  int weight;  Animal \*result; | 4[0]  100[4]  4[104]  8[108] |
| Bird.in()  char flag [100];  Fish.in()  int a  Beast.in()  int choose; | 100  100[0]  4  4[0]  4  4[0] |
| Если данные генерируются | |
| int size  inRandom(size)  Animal::staticInRandom()  int type;  int weight;  Animal \*result;  Bird.inRandom()  char flag [100];  Fish.inRandom()  int a  Beast.inRandom()  int choose; | 80032[24]  0  16  4[0]  4[4]  8[8]  100  100[0]  4  4[0]  4  4[0] |
| Если не файл и не генерация – вызов errMessage1() 0 | |
| FILE \*output | 8[80016] или [80056] |
| Container out() 0 | |
| Container ShakerSorting()  int left;  int right;  int flag;  Animal \*temp | 20  4[0]  4[4]  4[8]  8[12] |
| FILE \*output2 | 8[80024] или [80064] |
| Container out() 0 | |

В таблице памяти программ я показал методы и их варианты вызова. Обратите внимание, у некоторых элементов стартовые точки индексов считаются от main, поэтому у FILE \*output2, например, индекс [80024] или [80064], а те методы, которые вызываются в середине main имеют свою индексацию.

В стеке будут находиться вызовы функций, представленных в проекте, а также примитивные типы. Тем временем в кучу лягут все enum, строки и ссылки на объекты.

**Вот что показали замеры времени при генерации животных на входе.**

|  |  |
| --- | --- |
| **Количество животных** | **Время, сек** |
| 10 | 0.000004 |
| 100 | 0.011000 |
| 1000 | 0.015000 |
| 10000 | 0.389000 |

**Характеристики программы**

Размер проекта вместе с тестами: 5,80 МБ (6 091 850 байт)

Размер без тестов: 6 075 433 байт

Число программных файлов проекта: 12

Все файлы .cpp лежат в папке Sources, .h в Headers.

CMakeLists.txt

**# Headers**

* **Fish.h**
* **Bird.h**
* **Beast.h**
* **Animal.h**
* **Container.h**

**# Sources**

* **Fish.cpp**
* **Bird.cpp**
* **Beast.cpp**
* **Animal.cpp**
* **Container.cpp**
* **main.cpp)**

**Особенности реализации проекта**

**Проект написан в объектно-ориентированной парадигме. Есть базовый класс Animal с виртуальными или статическими методами. От него наследуются классы Fish, Bird и Beast. Внутри этих классов переопределяются некоторые методы базового класса для работы со специфическими полями потомка.**

**Сравнение с предыдущим проектом**

1. **Скорость**

|  |  |  |
| --- | --- | --- |
| **Количество животных** | **Время, сек ООП** | **Время, сек Процедурный подход** |
| 10 | 0.000004 | 0,00026 |
| 100 | 0.011000 | 0.000646 |
| 1000 | 0.015000 | 0.006414 |
| 10000 | 0.389000 | 0.394828 |

Показатели обоих подходов сравнимы. Часто скорость либо приблизительно одинакова, либо у процедурной реализации она больше. Предполагаю, это связано с тем, что в ооп-реализации задействуется большее количество памяти, и обёртки объектов замедляют работу.

1. **Размер проектов**

ООП-реализация заняла 5,80 МB против 240,1 KB у процедурной.