Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Национальный исследовательский университет «Высшая школа экономики»

Московский институт электроники и математики Национального исследовательского университета «Высшая школа экономики»

Департамент прикладной математики

Отчет

По лабораторной работе №1

По курсу «Алгоритмизация и программирование»

ФИО	Номер Группы	Дата
Цыплаков Александр Александрович	БПМ214	16.10.22

Москва - 2021 г.

Вариант №1

Задание №1

1) Сгенерировать файлы (.txt, .csv, .xml, .json и т.д. - любой формат на ваше усмотрение) с наборами случайных данных, каждый из которых состоит из элементов следующих типов:
   a. Целочисленные (любой тип)
   b. Строки от 20 до 40 символов (std::string)

   Наборы данных содержат 1000000 элементов.
   Для генерации можно использовать любое из: Excel, Python, C++.

Код программы:

```
import random
file1 = open('INTdata.txt', 'w', encoding= 'utf8')
file1.write('\n'.join(map(lambda _:
''.join(__import__('random').choice('123456789') for _  in range(5)),
range(int(input('n: '))))))
file2 = open('STRdata.txt', 'w', encoding='utf8')
file2.write('\n'.join(map(lambda _:
''.join(__import__('random').choice('qwertyuiopasdfghjklzxcvbnm') for _ in
range(25)), range(int(input('n: '))))))
```

Первым делом мы должны импортировать библиотеку для генерации различных вариантов строк. После чего открываем ранее созданный текстовый файл с нужным разрешением и кодировкой. Следующим шагом генерируем рандомные числа от 1 до 9 по 5 штук в одной строке, таких строк по заданию должно быть 10000000. Проделываем такую же процедуру для букв, только вместо цифр указываем буквы.

Задание №2

2) На языке C++ реализовать функционал для считывания наборов входных данных, сгенерированных в 1-ом пункте, в последовательные контейнеры: std::vector и std::list

Например, вид функции может быть такой:

тип_контейнера<тип считываемого значения> название_функции (const std::string& path_file, const int len);

Код программы:

```
std::vector<int> integer_vec_reader(const string& PATH, const int len)
{
    std::ifstream instream(PATH);
    vector<int>result;
    if (instream.is_open())
    {
        for (int i = 0; i < len; i++)
        {
            string current;
            getline(instream, current);
            result.push_back(std::stoi(current));
        }
    }
    instream.close();
    return result;
}
```

Выше представлена часть кода, программы, в которой создается вектор интовых значений из файла с цифрами, в котором мы также должны сделать проверку открытия файла, после чего запустить цикл для считывания данных. По образу и подобию создается лист интовых значений, где почти нет отличий с вектором. Для файла с символами делаем аналогичную процедуру, с незначительными изменениями:

```
std::vector<string>string_vec_reader(const string& PATH, const int len)
{
    std::ifstream instream(PATH);
    vector<string> result;
    if (instream.is_open())
    {
        string current;
        for (int i = 0; i < len; i++)
        {
            getline(instream, current);
            result.push_back(current);
        }
    }
    instream.close();
    return result;
}
```

Задание №3

3) Реализовать класс TimeLogger для работы с замерами времени. Обеспечить интерфейс класса, позволяющий использовать его по следующему сценарию:
   a. Создать объект logger, связать его с выходным файлом (куда будет записана статистика)
   b. Обнулить счетчик времени (например, logger.reset())
   c. *Произвести какие-то действия, время которых мы хотим замерить*
   d. Записать длительность (например, logger.log(…))

В результате должен получаться файл, содержащий данные о типе контейнера, размере данных, операции, времени выполнения этой операции.

Используя данные, сгенерированные в пункте 1, последовательно загружать их в контейнеры с помощью интерфейсов из пункта 2, **изменяя количество элементов от 100 000 до 1 000 000 с шагом 100 000.**
Далее провести эксперименты, описанные в пунктах 4-7, замеряя время с помощью TimeLogger, разработанного в пункте 3. Для корректности замеров времени отключить оптимизации компилятора и выбрать конфигурацию **Release**.

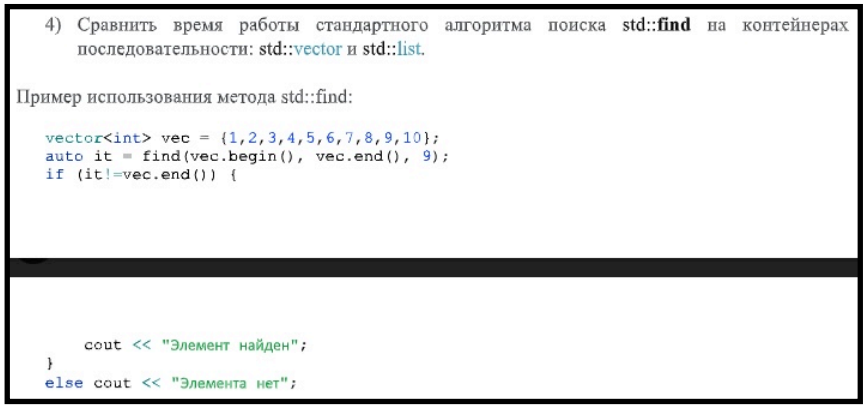
Должно получиться по 40 замеров для каждой операции 4-7 (2 типа контейнеров × 10 длин контейнеров × 2 типа данных = 40).

Код программы:

(Представлен ниже)
В этом задании мы должны провести замеры времени для определенных функций и действий в ходе работы нашей программы.
Первым делом мы должны открыть поток и задать начальные значения времени, после чего закрыть поток и установить текущее время, а далее использовать геттер для получения frozen и current time, аналогично используем сеттер для установления текущего времени. Public завершается функцией, которая отвечает за ввод в файл, а также в ней мы считаем разницу зафиксированного и текущего времени, а затем прописываем строки, которые будут выводится в итоговый файл

```cpp
class TimeLogger //реализация работы со временем и запись отдельный файл
{
public:
    std::ofstream outstream;

    TimeLogger(const string& PATH) //конструктор принимает путь до файла куда кидаем
    {
        outstream.open(PATH); //открываем поток и задаем начальные значения
        frozen_time = std::chrono::steady_clock::now();
        current_time = std::chrono::steady_clock::now();
    }

    ~TimeLogger()//деструктор
    {
        outstream.close();//закрытие потока
    }

    void reset()//утанавливаем текущее время
    {
        set_ctime(std::chrono::steady_clock::now());
        set_ftime(std::chrono::steady_clock::now());
    }

    std::chrono::steady_clock::time_point get_ftime()//getr для получения frozen time
    {
        return frozen_time;
    }

    std::chrono::steady_clock::time_point get_ctime()//getr для получения frozen time
    {
        return current_time;
    }

    void set_ftime(std::chrono::steady_clock::time_point time)//setr для установления текущего времени
    {
        frozen_time = time;
    }

    void set_ctime(std::chrono::steady_clock::time_point time)
    {
        current_time = time;
    }

    void log(const string container_type, const int len, const string operation) //ввод в файл,
        считаем разницу зафиксированного времени и текущего
    {
        if (outstream.is_open()) //проверка на открытие
        {
            set_ctime(std::chrono::steady_clock::now());
            auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(get_ctime() -
                get_ftime());
            outstream << "container: " << container_type << "; size: " << len
                << "; operation: " << operation << "; " << "time:"
                << elapsed_ms.count() << ";\n";
        }
    }
private:
    std::chrono::steady_clock::time_point frozen_time;
    std::chrono::steady_clock::time_point current_time;
};
```

Задание №4



4) Сравнить время работы стандартного алгоритма поиска **std::find** на контейнерах последовательности: std::vector и std::list.

Пример использования метода std::find:

```
vector<int> vec = {1,2,3,4,5,6,7,8,9,10};
auto it = find(vec.begin(), vec.end(), 9);
if (it!=vec.end()) {

    cout << "Элемент найден";
}
else cout << "Элемента нет";
```

Код программы:

```
//OPERATION : FIND
for (auto elem : sizes)
{
    logger.reset();
    vector<int>nums = integer_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/INTdata.txt", elem);
    std::find(nums.begin(), nums.end(), nums[elem-1]); //ищем последний элемент из заполненного
        массива
    logger.log("vector<int>", elem, "std::find");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    list<int>nums = integer_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/INTdata.txt", elem);
    std::find(nums.begin(), nums.end(), *next(nums.begin(), elem-1));
    logger.log("list<int>", elem, "std::find");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    vector<string>nums = string_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/STRdata.txt", elem);
    std::find(nums.begin(), nums.end(), nums[elem - 1]);
    logger.log("vector<string>", elem, "std::find");
}
logger.outstream << endl;


for (auto elem : sizes)
{
    logger.reset();
    list<string>nums = string_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/STRdata.txt", elem);
    std::find(nums.begin(), nums.end(), *next(nums.begin(), elem - 1));
    logger.log("list<string>", elem, "std::find");
}
```

std::vector значительно быстрее осуществляет обход элементов, чем std::list, так как в нем данные лежат последовательно

Задание №5

5) Сравнить время работы сортировки на контейнерах последовательности: std::vector и std::list.

Примечание: стандартный алгоритм сортировки **std::sort** удастся применить к контейнерам std::vector и std::deque, но он не сработает с std::list… Зато у std::list, в отличие от std::vector и std::deque, есть свой собственный метод **.sort()** – зачем он нужен, т.е. почему нельзя было просто обойтись std::**sort**?

Пример использования:

```
vector<int> vec = {1,2,3,4,5,6,7,8,9,10};
sort(vec.begin(), vec.end());
```

Код программы:

```
//OPERATION: SORT
for (auto elem : sizes)
{
    vector<int>nums = integer_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/INTdata.txt", elem);
    logger.reset();
    std::sort(nums.begin(), nums.end());
    logger.log("vector<int>", elem, "std::sort");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    list<int>nums = integer_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/INTdata.txt", elem);
    nums.sort();
    logger.log("list<int>", elem, "std::sort");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    vector<string>nums = string_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/STRdata.txt", elem);
    std::sort(nums.begin(), nums.end());
    logger.log("vector<string>", elem, "std::sort");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    list<string>nums = string_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/STRdata.txt", elem);
    nums.sort();
    logger.log("list<string>", elem, "std::sort");
}
```

Задание №6

6) Сравнить время работы удаления элемента из начала и конца контейнеров последовательности: std::vector, std::list и std::deque.

Примечание: удаление элемента работает слишком быстро, чтобы замеры по времени отразили качественный результат, поэтому необходимо загрузить данные со сгенерированного файла в контейнер, после чего поэлементно полностью удалить его. При выполнении этого пункта рекомендуется пользоваться методами **.pop_front()** и **.pop_back()**. Однако у std::vector нет метода **.pop_front()** – почему? Как тогда удалить элемент из начала std::vector?

```cpp
vector<int> vec = {1,2,3,4,5,6,7,8,9,10};

vec.push_back(15); //  вставили 15 в конец
vec.insert(vec.begin(), 15); // вставили 15 в начало

for (auto x: vec) {
    cout << x << endl;
}
```

Код программы:

```cpp
// POP_BACK
for (auto elem : sizes)
{
    logger.reset();
    vector<int>nums = integer_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы /№1/№1/INTdata.txt", elem);
    for (int i = 0; i < elem; i++)
    {
        nums.pop_back();
    }
    logger.log("vector<int>", elem, "pop_back()");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    list<int>nums = integer_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы /№1/№1/INTdata.txt", elem);
    for (int i = 0; i < elem; i++)
    {
        nums.pop_back();
    }
    logger.log("list<int>", el
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    vector<string>nums = string_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы /№1/№1/STRdata.txt", elem);
    for (int i = 0; i < elem; i++)
    {
        nums.pop_back();
    }
    logger.log("vector<string>", elem, "pop_back()");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    list<string>nums = string_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы /№1/№1/STRdata.txt", elem);
    for (int i = 0; i < elem; i++)
    {
        nums.pop_back();
    }
    logger.log("list<string>", elem, "pop_back()");
}
```

```cpp
// POP_FRONT
for (auto elem : sizes)
{
    logger.reset();
    vector<int>nums = integer_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/INTdata.txt", elem);
    for (int i = 0; i < elem; i++)
    {
        nums.erase(nums.begin());
    }
    logger.log("vector<int>", elem, "pop_front()");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    list<int>nums = integer_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/INTdata.txt", elem);
    for (int i = 0; i < elem; i++)
    {
        nums.pop_front();
    }
    logger.log("list<int>", elem, "pop_front()");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    vector<string>nums = string_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/STRdata.txt", elem);
    for (int i = 0; i < elem; i++)
    {
        nums.erase(nums.begin());
    }
    logger.log("vector<string>", elem, "pop_front()");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    list<string>nums = string_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные работы
        /№1/№1/STRdata.txt", elem);
    for (int i = 0; i < elem; i++)
    {
        nums.pop_front();
    }
    logger.log("list<string>", elem, "pop_front()");
}
```

При использовании вставки в начало std::list работает эффективнее, а при вставке в конец std::vector эффективнее

Задание №7

7) Сравнить время работы вставки элемента в начало и конец контейнеров последовательности: std::vector, std::list и std::deque.

Примечание: вставка элемента работает слишком быстро, чтобы замеры по времени отразили качественный результат, поэтому необходимо полностью загрузить данные со сгенерированного файла в контейнер, после чего создать пустой контейнер и с помощью вставки в начало/конец полностью заполнить его элементами из ранее загруженного контейнера. Также проверить при таком подходе на сколько быстрее будет заполняться std::vector, если предварительно вызвать метод .**reserve**(…), в отличие от случая, когда это не сделано. При выполнении этого пункта рекомендуется пользоваться методами .**push_front**(el) и .**push_back**(el). Однако у std::vector нет метода .**push_front**(el) – почему? Как тогда вставить элемент в начало std::vector?

Код программы:

```cpp
//PUSH_FRONT
for (auto elem : sizes)
{
    logger.reset();
    vector<int>nums_storage = integer_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные
        работы /№1/№1/INTdata.txt", elem);
    vector<int>nums;
    for (int i = 0; i < elem; i++)
    {
        nums.insert(nums.begin(), nums_storage[i]);
    }
    logger.log("vector<int>", elem, "push_front()");
}
logger.outstream << endl;

//PUSH_FRONT IN RESERVE USING CASE сначала выделяем память, а потом добавляем
for (auto elem : sizes)
{
    logger.reset();
    vector<int>nums_storage = integer_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные
        работы /№1/№1/INTdata.txt", elem);
    vector<int>nums;
    nums.reserve(elem);
    for (int i = 0; i < elem; i++)
    {
        nums.insert(nums.begin(), nums_storage[i]);
    }
    logger.log("vector<int>", elem, "push_front()");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    list<int>nums_storage = integer_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные
        работы /№1/№1/INTdata.txt", elem);
    list<int>nums;
    for (int i = 0; i < elem; i++)
    {
        nums.push_front(*next(nums_storage.begin(), i));
    }
    logger.log("list<int>", elem, "push_front()");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    vector<string>nums_storage = string_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторны
        работы /№1/№1/STRdata.txt", elem);
    vector<string>nums;
    for (int i = 0; i < elem; i++)
    {
        nums.insert(nums.begin(), nums_storage[i]);
    }
    logger.log("vector<string>", elem, "push_front()");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    list<string>nums_storage = string_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные
        работы /№1/№1/STRdata.txt", elem);
    list<string>nums;
    for (int i = 0; i < elem; i++)
    {
        nums.push_front(*next(nums_storage.begin(), i));
    }
    logger.log("list<string>", elem, "push_front()");
}
logger.outstream << endl;

//PUSH_BACK
for (auto elem : sizes)
{
    logger.reset();
    vector<int>nums_storage = integer_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные
        работы /№1/№1/INTdata.txt", elem);
    vector<int>nums;
    for (int i = 0; i < elem; i++)
    {
        nums.push_back(nums_storage[i]);
    }
    logger.log("vector<int>", elem, "push_back()");
}
logger.outstream << endl;
```

```cpp
//PUSH_BACK IN RESERVE USING CASE
for (auto elem : sizes)
{
    logger.reset();
    vector<int>nums_storage = integer_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные
        работы /№1/№1/INTdata.txt", elem);
    vector<int>nums;
    nums.reserve(elem);
    for (int i = 0; i < elem; i++)
    {
        nums.push_back(nums_storage[i]);
    }
    logger.log("vector<int>", elem, "push_back()");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    list<int>nums_storage = integer_list_reader("/Users/sashats/Documents/АиП/C++/Лабораторные
        работы /№1/№1/INTdata.txt", elem);
    list<int>nums;
    for (int i = 0; i < elem; i++)
    {
        nums.push_back(*next(nums_storage.begin(), i));
    }
    logger.log("list<int>", elem, "push_back()");
}
logger.outstream << endl;

for (auto elem : sizes)
{
    logger.reset();
    vector<string>nums_storage = string_vec_reader("/Users/sashats/Documents/АиП/C++/Лабораторные
        работы /№1/№1/STRdata.txt", elem);
    vector<string>nums;
    for (int i = 0; i < elem; i++)
    {
        nums.push_back(nums_storage[i]);
    }
    logger.log("vector<string>", elem, "push_back()");
}
logger.outstream << endl;
```

Подведем итоги:

Из проделанной работы мы можем заметить, что сортировку лучше использовать на векторе, как и поиск элементов, чем на списке, в связи со временем работы, но std::list значительно удобнее при работе с большими данными.