Лабораторная работа № 1

# **Формирование массивов экспериментальных данных**

Цель работы:

* освоить принципы формирования монотонных (упорядоченных и упорядоченных в обратном порядке), частично упорядоченных и случайных последовательностей данных (приложение А);
* изучить функции, позволяющие производить оценку длительности времени выполнения алгоритмов;
* изучить и использовать правила оформления функций с использованием указателей на функции в качестве формальных параметров (или правила формирования массивов указателей на функции).

**Общие сведения**

Изучение и исследование разнообразных алгоритмов сортировки и поиска, структур данных невозможно без использования либо *реальных* массивов данных, либо формируемых *искусственно*, но обладающих свойствами схожими со свойствами реальных массивов данных.

Так как у исследователя не всегда имеется возможность использования реальных данных, то возникает необходимость в формировании псевдо-реальных массивов данных, обладающих теми или иными свойствами.

При исследовании алгоритмов сортировки, как правило, оговаривается характер упорядочивания – по возрастанию или убыванию. При этом под монотонной последовательностью (массивом) данных понимается последовательность, для *всех* элементов которой выполняется отношение *Di* ≤ *Di*+1 (либо *Di* ≥ *Di*+1).

Если для всех элементов выполняется отношение *Di* < *Di*+1 (либо *Di* > *Di*+1), то последовательность называется **строго монотонной**.

Если последовательность данных является монотонной и характер отношений ее элементов соответствует цели алгоритма сортировки, то последовательность называется **упорядоченной**, в противном случае – упорядоченной в обратном порядке.

**Случайной последовательностью** называется последовательность, элементы которой являются случайными величинами (как правило, псевдослучайными), распределенными по некоторому закону распределения.

Наиболее часто для формирования псевдослучайных величин используется равномерный закон, но на его основе можно сформировать псевдослучайные величины, распределенные и по другим законам распределения – экспоненциальному, нормальному и др.

Помимо упорядоченных и случайных последовательностей существуют также последовательности, элементы которых обладают свойством упорядоченности в каком-либо смысле.

Для определенности, будем называть такие последовательности **частично упорядоченными**.

Можно выделить следующие, наиболее характерные случаи частично упорядоченных последовательностей:

* «пилообразные»,
* «синусоидальные»,
* «ступенчатые»,
* «квази-упорядоченные».

В первом случае всю последовательность можно разбить на интервалы (длины интервалов, для простоты, можно считать одинаковыми), в пределах которых отношение порядка выполняется, но оно, может быть, не справедливо для соседних элементов, принадлежащих двум соседним интервалам.

Во втором случае отношение порядка на соседних интервалах чередуется, например, для *k*-го интервала справедливо *Di* ≤ *Di*+1, для *k*+1-го – *Di* ≥ *Di*+1.

В третьем случае в некотором смысле упорядочены сами интервалы – максимальное значение на *k*-м интервале меньше (не больше), чем минимальное значение на *k*+1-м интервале.

Квази-упорядоченная последовательность – это последовательность, в которой имеется ограниченное количество инверсий (количество элементов, для которых не выполняется отношение порядка) и оно много меньше размера последовательности.

В качестве исходных последовательностей для исследования алгоритмов сортировки и поиска наиболее часто используют последовательности целых чисел и чисел с плавающей запятой.

В этом случае для формирования псевдослучайной величины можно использовать функции стандартной библиотеки **srand(), rand ()**:

void **srand**( unsigned int seed );

// инициализация генератора случайных чисел **rand**.

Вариант вызова этой функции: **srand** ( time(NULL));

Чтобы генерировать случайные числа, функция **srand()** обычно инициализируется некоторыми различными значениями, например, такие значения генерируются функцией time.

Значение, возвращенное функцией time (объявлена в заголовке <ctime>) отличается каждую секунду, что дает возможность получать различные случайные последовательности чисел при каждом новом вызове функции **rand**.

int **rand**(void);

Возвращает псевдослучайную величину в диапазоне от 0 до RAND\_MAX;

При каждом обращении к функции возвращается целое в интервале между нулем и значением **RAND\_MAX**, которое в любой реализации должно быть не меньше числа 32 767.

Используя функцию rand() можно получить и псевдослучайные числа с плавающей запятой, принадлежащие некоторому заданному диапазону.

**Оценка временных характеристик алгоритмов**:

В ряде случаев приемлемо использование функции прикладного программного интерфейса Windows (для этого необходимо подключить заголовочный файл windows.h):

void **GetLocalTime**(SYSTEMTIME \* st);

В качестве параметра эта функция получает указатель на структуру SYSTEMTIME, поля которой содержат текущие значения (на момент вызова функции) миллисекунд, секунд, минут, …, года, что позволяет оценивать временные характеристики с точностью до миллисекунд.

Можно использовать функцию **GetSystemTime**(&time);

Еще одной функцией, которая позволяет оценить время выполнения, является функция

DWORD **GetTickCount**(void);

или DWORD **GetTickCount64**(void);

Эта функция возвращает количество «тиков» (миллисекунд), прошедших с момента запуска операционной системы.

В зависимости от того в какой программной среде выполняется работа и на каком компьютере, иногда необходимо использовать для подсчета времени (**в микросекундах**) специальную библиотеку:

#include<chrono>

// подключение библиотеки для вычисления времени работы алгоритмов

. . .

auto begin = std::chrono::steady\_clock::now();

// получаем время перед началом формирования последовательности

auto end = std::chrono::steady\_clock::now();

// получаем время по окончанию формирования последовательности

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::microseconds>(end - begin); //получаем время работы в микросекундах

std::cout << "Время работы алгоритма по формированию последовательности: " << elapsed\_ms.count() << " (мкС)" << std::endl; // вывод времени работы

. . .

**Задание**

**1 этап работы**

Разработать функции, формирующие последовательности чисел, для целых чисел и чисел с плавающей запятой (должно быть два комплекта функций и соответственно результатов):

* упорядоченные по возрастанию,
* упорядоченные по убыванию,
* случайную последовательность,
* частично упорядоченные: «пилообразные», «синусоидальные», «ступенчатые».

.

При реализации функций считать, что выделение памяти под последовательности происходит вне этих функций, функции в качестве формальных параметров получают указатель на массив, его размер и, возможно, **диапазон изменения величин и длину интервалов** (для частично упорядоченных последовательностей).

Рекомендуется использовать указатели на функцию в качестве формального параметра или массив указателей на функции для автоматизации сбора статистической информации.

Для массивов размерностью **от 150 до 200 элементов** (размерность выбирается самостоятельно) предусмотреть вывод значений в файл.

Используя данные из файлов, **построить графики** полученных зависимостей (возможно с привлечением других программных средств) для доказательства правильности полученных функций.

**2 этап работы**

1. Оценить длительность формирования последовательностей всех типов для различных значений размеров последовательностей (5⋅105, 10⋅105, …, 50⋅105), и на основе полученных значений построить графики зависимостей длительностей(времени) формирования массивов от их размера (лучше в виде столбчатых диаграмм, можно в Microsoft Exel).
2. Составить отчет, в котором привести структурные схемы алгоритмов главной функции и какой-то одной из функций (например, реализующей частично упорядоченную последовательность), текст программы, результаты по временным интервалам для различных размерностей массивов, графики зависимостей (времени от размерностей) и выводы по полученным результатам.

**Приложение А. Виды последовательностей**