Функциональные объекты

STL6Func1. Определить *функциональный объект* (*объект-функцию*, *функтор*) — структуру less_abs с операцией () — константной функцией-членом, имеющей два целочисленных параметра *a* и *b* и возвращающей результат сравнения $|a| < |b|$. Структура less_abs должна быть порождена от стандартной обобщенной структуры binary_function<int, int, bool>. Используя функциональный объект less_abs и алгоритм sort, отсортировать исходный вектор *V* целых чисел по возрастанию их абсолютных значений.

STL6Func2. Дан вектор *V* целых чисел. Используя два вызова алгоритма adjacent_find, найти две начальных пары элементов (*a*, *b*) вектора *V*, для которых выполняется неравенство $|a| \ge |b|$, и вывести найденные пары элементов в порядке возрастания их индексов. Если вектор содержит единственную пару требуемых элементов, то вывести только ее; если подходящих пар нет, то вывести единственное число 0. Для поиска требуемых пар использовать *функциональный адаптер* not2 (*инвертор*), применив его к функциональному объекту less_abs, описанному в STL6Func1.

Примечание. Для возможности применения функционального адаптера к объекту-функции необходимо, чтобы, во-первых, данный объект был порожден от соответствующего базового класса (в данном случае класса бинарных функций binary_function) и, во-вторых, имел реализацию операции () в виде *константной* функции-члена. Для непосредственного применения объекта-функции (без функциональных адаптеров) перечисленные условия не являются обязательными.

STL6Func3. Дано целое число *K* (> 0) и вектор *V*, содержащий целые числа. Используя функциональный объект less_abs, описанный в STL6Func1, совместно с функциональным адаптером bind (*связывателем*) в алгоритме remove_if, а также метод erase, удалить из вектора *V* все элементы, абсолютная величина которых меньше *K*. Вывести размер преобразованного вектора *V* и его элементы. Для возможности использования параметра _1 адаптера bind в программу необходимо добавить директиву using namespace std::placeholders.

Примечание. Если компилятор не поддерживает стандарт C++11, то вместо универсального связывателя bind можно использовать специализированный адаптер-связыватель для второго параметра bind2nd.

STL6Func4. Дано целое число *K* (> 0) и вектор *V*, содержащий целые числа. Используя функциональный объект less_abs, описанный в STL6Func1, совместно с функциональным адаптером bind в алгоритме find_if, найти и вывести последний элемент вектора, абсолютная величина которого больше *K*. Если вектор не содержит требуемых элементов, то вывести 0.

Примечание. Если компилятор не поддерживает стандарт C++11, то вместо универсального связывателя bind можно использовать специализированный адаптер-связыватель для первого параметра bind1st.

STL6Func5. Дано целое число *K* (> 0) и вектор *V*, содержащий целые числа. Используя функциональный объект less_abs, описанный в задаче STL6Func1, совместно с инвертором not2 и связывателем bind, построить унарный предикат для проверки условия $|x| \le K$ и использовать его в качестве последнего параметра алгоритма count_if для нахождения количества элементов вектора, абсолютная величина которых меньше или равна *K*.

Примечания. (1) Если компилятор не поддерживает стандарт C++11, то вместо связывателя bind можно использовать связыватель bind1st. (2) В данном случае нет необходимости в применении инвертора (обеспечивающего преобразование строгого неравенства в нестрогое), так как для решения задачи достаточно использовать предикат со строгим неравенством вида $|x| < K + 1$. Однако описанный прием может оказаться полезным в аналогичных ситуациях с более сложными функциональными объектами.

STL6Func6. Даны векторы V_1 и V_2 с одинаковым количеством элементов — целых чисел. Используя алгоритм transform и стандартный функциональный объект multiplies, преобразовать вектор V_2, умножив его элементы на соответствующие элементы вектора V_1.

STL6Func7. Дано целое число *K* и вектор *V*, содержащий целые числа. Используя алгоритм transform и стандартный объект-функцию minus со связывателем bind, преобразовать исходный вектор, уменьшив значения всех его элементов на величину *K*.

Примечание. Если компилятор не поддерживает стандарт C++11, то вместо связывателя bind можно использовать связыватель bind2nd.

STL6Func8. Даны целые числа *A*, *B* и вектор *V*, содержащий целые числа. Используя алгоритм transform и стандартные объекты-функции plus и multiplies с вложенными связывателями bind, преобразовать вектор *V*, заменив каждый его элемент *x* на значение $A \cdot x + B$.

Примечание. Специализированные связыватели bind1st и bind2nd не позволяют сконструировать требуемый объект-функцию на основе стандартных. Если компилятор не поддерживает стандарт C++11, то решить задачу, не создавая новые объекты-функции, можно с помощью *двукратного* применения алгоритма transform.

STL6Func9. Определить структуру point с целочисленными полями *x*, *y* и строковым полем *s*. Отношение порядка для данной структуры определяется следующим образом: $A < B$, если $A.x < B.x$ или ($A.x == B.x$ и $A.y < B.y$). Реализовать это отношение порядка в виде константной функции-члена `bool operator<(const point& b) const`. Кроме того, реализовать операцию `istream& operator>>(istream& is, point& p)`, которая последовательно считывает из входного потока *is* поля *x*, *y* и *s* структуры *p*, и операцию `ostream& operator<<(ostream& os, const point& p)`, которая записывает все поля структуры *p* в выходной поток *os* (поля записываются в том же порядке, в котором считываются, между полями вставляется один пробел, после вывода полей происходит переход на новую строку). Дан текстовый файл с именем *name*, содержащий текстовые представления элементов описанной выше структуры. Используя итератор чтения istream_iterator, заполнить этими данными вектор *V* с элементами типа point. Используя алгоритм stable_sort, отсортировать полученные данные с учетом заданного отношения порядка и с помощью алгоритма copy и итератора записи ostream_iterator записать отсортированный вектор в исходный файл, заменив его прежнее содержимое (при этом точки с одинаковыми координатами будут располагаться в том же порядке, что и в исходном файле, поскольку алгоритм stable_sort обеспечивает устойчивую сортировку).

Примечание. Благодаря явно определенной операции <, в алгоритме сортировки не требуется указывать объект-функцию для сравнения элементов вектора.

STL6Func10. Дан текстовый файл с именем *name*, содержащий текстовые представления элементов структуры point, реализованной в STL6Func9. Используя вспомогательный вектор *V* с элементами типа point и алгоритм count_if, определить и вывести количество элементов из исходного файла, которые меньше точки с координатами (0, 0). В качестве параметра — функционального объекта в алгоритме count_if использовать ссылку на функцию-член point::operator<, преобразовав ее в объект-функцию с помощью функции mem_fun_ref и применив к результату связыватель bind.

Примечания. (1) Если компилятор не поддерживает стандарт C++11, то вместо bind можно использовать bind2nd. (2) При использовании в качестве объектов-функций функций-членов необходимо применять к ним функцию mem_fun_ref (если же алгоритм применяется к контейнеру, содержащему указатели на объекты, то для преобразования функции-члена в объект-функцию требуется применить к ней функцию mem_fun). (3) Следует заметить, что для сортировки элементов вектора по убыванию нельзя использовать (синтаксически допустимый) объект-функцию `not2<bool>(mem_fun_ref(&point::operator<))`, так как в результате будет получен объект-функция для *нестрогого* сравнения ($a \geq b$), который не может использоваться в алгоритмах, связанных с сортировкой (для него нарушается условие строгого сравнения: если *a* «меньше» *b*, то *b* не может одновременно быть «меньше» *a*).

STL6Func11. Даны текстовые файлы с именами *name1* и *name2*, содержащие текстовые представления элементов структуры point, реализованной в STL6Func9. Файлы содержат одинаковое количество элементов. Реализовать операцию сложения для объектов point, которая складывает значения соответствующих полей (как числовых, так и строковых), в виде перегруженной функции `point operator+(const point& a, const point& b)`. Прочесть данные из файлов *name1* и *name2* в векторы V_1 и V_2 соответственно. Используя алгоритм transform с параметром plus<point>(), преобразовать элементы вектора V_1, прибавив к ним соответствующие элементы вектора V_2. Записать преобразованный вектор V_1 в файл *name1*, заменив его прежнее содержимое.

Примечание. Стандартный объект-функция plus является оберткой для операции +. Поскольку операция + реализована в виде обычной функции operator+ (не члена класса), ее обертку plus можно использовать в качестве параметра алгоритма.

STL6Func12. Дан текстовый файл с именем *name*, содержащий текстовые представления элементов структуры point, реализованной в STL6Func9. Прочесть данные из файла *name* в вектор *V*. Используя алгоритм transform с объектом-функцией plus<point>() (см. STL6Func11) и связывателем bind, преобразовать элементы вектора *V*, прибавив к ним точку с полями (10, 20, «Z»). Записать преобразованный вектор *V* в файл *name*, заменив его прежнее содержимое.

Примечание. Если компилятор не поддерживает стандарт C++11, то вместо связывателя bind можно использовать bind1st или bind2nd.

STL6Func13. Определить структуру range с целочисленным полем *n* и операцией (), которая возвращает выражение *n*++. Также определить конструктор range с одним параметром — начальным значением поля *n*. Даны целые числа *K* и *N* ($N > 0$). Используя алгоритм generate_n с объектом-функцией range и итератором вставки, заполнить вектор *V* набором из *N* последовательных значений, начиная со значения *K* (в результате вектор должен содержать значения *K*, $K + 1$, $K + 2$, …, $K + N - 1$), и вывести элементы полученного вектора.

Примечание. Если компилятор поддерживает стандарт C++11, то вместо объекта-функции range достаточно использовать лямбда-выражение с захваченной по ссылке внешней переменной `x` (которая инициализируется значением *K*): `[&x](){return x++;}`.

STL6Func14. Определить структуру arith_pr с вещественными полями *a* и *d* — текущим членом и разностью арифметической прогрессии — и операцией (), которая возвращает очередной член прогрессии, начиная с первого. Также определить конструктор arith_pr с двумя параметрами — первым членом прогрессии и разностью. Даны вещественные числа *A*, *D* и целое число *N* (> 0). Используя алгоритм generate_n с объектом-функцией arith_pr и итератором вставки, заполнить вектор *V* набором из *N* начальных членов арифметической прогрессии с первым членом *A* и разностью *D* (в результате вектор должен содержать значения *A*, $A + D$, $A + 2D$, …, $A + (N - 1)D$) и вывести элементы полученного вектора.

Примечание. Если компилятор поддерживает стандарт C++11, то вместо объекта-функции arith_pr достаточно использовать лямбда-выражение с двумя захваченными внешними переменными (см. примечание к STL6Func13).