Лабораторная работа №10
«Разработка шаблона класса»

Скоробогатов С.Ю.

12 мая 2016 г.

1 Цель работы

Целью данной работы является изучение шаблонов классов языка C++.

2 Исходные данные

Информация об объявлении и специализации шаблонов дана в лекционном курсе. Здесь мы рассмотрим только один вопрос, касающийся специализации шаблонов в зависимости от значений их целочисленных параметров.

Пусть шаблон некоторого класса Abs имеет целочисленный параметр N:

```
template <int N>
class Abs
{
public:
    void print();
};

template <int N>
void Abs<N, isPositive>::print()
{
    cout << N << endl;
}
```

Допустим, мы хотим добиться, чтобы в этом совершенно искусственном примере метод print всегда печатал абсолютное значение числа N. При этом мы собираемся достигнуть этого результата через специализацию шаблона класса.

Добавим в шаблон дополнительный булевский параметр isNegative со значением по умолчанию, вычисляемым на основе значения параметра N:

```
template <int N, bool isNegative = (N > 0)>
class Abs
{
public:
    void print();
};

template <int N, bool isNegative>
```

```
9  void Abs<N, isPositive>::print()
10 {
11     cout << N << endl;
12 }
```

Очевидно, что при отрицательных значениях параметра N значение параметра isNegative будет истинным. Поэтому мы можем добавить специализированную версию шаблона для отрицательных N:

```
14 template <int N>
15 class Abs<N, true>
16 {
17 public:
18     void print();
19 };
20
21 template <int N>
22 void Abs<N, true>::print()
23 {
24     cout << -N << endl;
25 }
```

Теперь при инстанциации шаблона класса Abs с отрицательным N компилятор будет задействовать специализированную версию, в которой N печатается с изменением знака. В этом можно убедиться, добавив в нашу программу следующий код:

```
27 int main()
28 {
29     Abs<-5> t;
30     t.print();
31     return 0;
32 }
```

3 Задание

Согласно выбранному из таблиц 1–6 описанию требуется составить шаблон класса, разместив его в отдельном заголовочном файле. Проверку работоспособности класса требуется организовать в функции `main`, размещённой в файле «main.cpp».

Таблица 1: Варианты заданий

1	Vector<T, N> – вектор размера N с элементами типа T, имеющий операции сложения, скалярного умножения и умножения на число. Vector<T, 3> дополнительно имеет операцию векторного умножения.
2	Polynom<T, N> – полином порядка N с коэффициентами типа T, имеющий операции вычисления значения, сложения и дифференцирования. Polynom<T, 2> и Polynom<T, 1> дополнительно имеют операцию нахождения корней.
3	Stack<T> – стек с элементами типа T, имеющий в дополнение к обычным стековым операциям операцию вычисления максимального элемента, работающую за константное время. Stack<string> должен дополнительно иметь операцию переворачивания всех строк, находящихся в стеке, работающую за константное время.
4	PQueue<T, N> – очередь с приоритетом максимального размера N с элементами типа T, имеющая одновременно операцию удаления максимального элемента и операцию удаления минимального элемента. Указанные операции должны работать за логарифмическое время. В PQueue<**bool**, N> все операции должны работать за константное время.
5	Stack<T> – стек с элементами типа T, имеющий в дополнение к обычным стековым операциям операцию переворачивания стека. Все операции должны работать за константное время за исключением случаев, когда требуется увеличить размер памяти, выделенной для хранения элементов. Для представления стека нужно использовать **модифицированный кольцевой буфер**. В Stack<string> дополнительно должна быть реализована операция, сообщающая о наличии в стеке пустой строки. Эта операция должна работать за константное время.
6	Queue<T, N> – очередь с элементами типа T и максимальным размером N, имеющая обычные для очереди операции. В Queue<**bool**, N> каждый элемент должен быть представлен одним битом.
7	Polynom<T, N> – полином порядка N с коэффициентами типа T, имеющий операцию вычисления значения. В Polynom<**bool**, N> i-ый коэффициент означает наличие или отсутствие в полиноме члена x^i. При этом для хранения каждого коэффициента должен использоваться один бит.
8	PQueue<T, N> – очередь с приоритетом максимального размера N с элементами типа T, имеющая обычные для очереди с приоритетом операции. В PQueue<**bool**, N> каждый элемент должен быть представлен одним битом.
9	Stack<T, N> – стек с элементами типа T и максимальным размером N, имеющий обычные стековые операции. В Stack<**bool**, N> каждый элемент должен быть представлен одним битом.
10	Queue<T, N> – очередь с элементами типа T и максимальным размером N, имеющая в дополнение к обычным для очереди операциям операцию переворачивания очереди, работающую за константное время. В Queue<**int**, N> должна быть дополнительно реализована операция вычисления суммы элементов, работающая за константное время.

Таблица 2: Варианты заданий

11	Vector<T, N> – вектор размера N с элементами типа T, имеющий операции сложения и скалярного умножения. В Vector<**bool**, N> каждый компонент должен быть представлен одним битом.
12	Stack<T> – стек с элементами типа T, имеющий в дополнение к обычным стековым операциям операцию переворачивания стека. Все операции должны работать за константное время. Для представления стека нужно использовать **двунаправленный список**. В Stack<**int**> дополнительно должна быть реализована операция, сообщающая о наличии в стеке нулевого элемента. Эта операция должна работать за константное время.
13	Seq<T, N> – неизменяемая упорядоченная последовательность длины N элементов типа T, имеющая операцию поиска первого вхождения в неё подпоследовательности длины M. В операции поиска должен использоваться алгоритм Кнута–Морриса–Пратта. Операция поиска должна создаваться шаблоном, имеющим параметр M. Этот шаблон должен быть специализирован для случая $M = N$, при котором алгоритм Кнута–Морриса–Пратта не нужен, а достаточно просто сравнить две последовательности.
14	Seq<T, N> – неизменяемая упорядоченная последовательность длины N элементов типа T, имеющая операцию, возвращающую значение i-того элемента, и операцию поиска элемента делением пополам. В классе Seq<T, N> должен быть реализован статический метод merge, выполняющий слияние двух упорядоченных последовательностей, имеющих длины N и M, в упорядоченную последовательность длины $N + M$, ссылка на которую передаётся ему в качестве параметра. Конструктор класса Seq<T, N> должен принимать в качестве параметра массив, имеющий тип T[N], и выполнять сортировку этого массива слиянием с использованием метода merge. Конструктор специализированной версии класса Seq<T, 1> прости копирует внутрь объекта передаваемый ему массив единичной длины.
15	SegmentTree<T, N> – дерево отрезков для последовательности длины N с элементами типа T, имеющее операцию вычисления суммы элементов на заданном отрезке и операцию изменения указанного элемента. SegmentTree<string, N> должно быть реализовано через массив строк размера N, причём операция вычисления суммы (конкатенации) элементов на отрезке должна работать за время, пропорциональное длине результирующей строки.
16	FenwickTree<T, N> – дерево Фенвика для последовательности длины N с элементами типа T, имеющее операцию вычисления исключающего ИЛИ элементов на заданном отрезке и операцию изменения указанного элемента. В FenwickTree<**bool**, N> каждому элементу должен соответствовать 1 бит.
17	SparseTable<T, N> – разреженная таблица для последовательности длины N с элементами типа T, имеющая операцию вычисления максимума на заданном отрезке. В SparseTable<**bool**, N> для хранения каждого промежуточного максимума должен использоваться 1 бит.

Таблица 3: Варианты заданий

18	FixNum<T, N, M> – число с фиксированной точкой, реализованное на базе массива размера N целочисленного типа T и имеющее M бит, отведённых для хранения дробной части. Число должно поддерживать следующие операции: 1. сложение с числом, имеющим $N2$ бит в добной части, в результате которого получается число, имеющее в дробной части $\max(N, N2)$ бит (эта операция должна создаваться шаблоном, имеющим праметр $N2$); 2. перевод в число с плавающей точкой. Специализированная версия FixNum<T, 1, M> должна использовать для хранения числа не массив, а единственное значение типа T.
19	BitmapSet<T, N> – множество натуральных чисел из интервала $[0, N)$, реализованное как битовая маска на базе массива целочисленного типа T. Множество должно поддерживать операцию проверки принадлежности числа, а также операции добавления и удаления числа. Специализированная версия BitmapSet<**long**, 64> должна использовать для хранения множества не массив, а единственное значение типа **long**.
20	Queue<T, N> – очередь с элементами типа T и максимальным размером N, реализованная через двойной стек и имеющая обычные для очереди операции. Queue<**int**, N> дополнительно имеет операцию нахождения максимального числа в очереди.
21	Queue<T, N> – очередь с элементами типа T и максимальным размером N, реализованная через двойной стек и имеющая в дополнение к обычным для очереди операциям операцию переворачивания очереди, работающую за константное время. В Queue<**bool**, N> каждый элемент должен быть представлен одним битом.
22	Queue<T, N> – «неизменяемая» очередь с элементами типа T и максимальным размером N, имеющая обычные для очереди операции. «Неизменяемость» очереди заключается в том, что операции Enqueue и Dequeue вместо изменения очереди, для которой они вызваны, создают и возвращают новую очередь, отличающуюся от исходной на один элемент. При этом исходная очередь полностью сохраняет своё состояние и работоспособность. Операции Enqueue и Dequeue нужно реализовать так, чтобы они работали за амортзированное константное время. Для этого нужно использовать два стека размера N. Операция Enqueue записывает новый элемент в первый стек, а операция Dequeue забирает элемент из второго стека. Фокус заключается в том, что стеки выделяются в динамической памяти и являются общими для очереди, на которой сработала операция, и для очереди, которая была создана в результате работы операции. Отметим, что новая пара стеков создаётся в момент создания новой пустой очереди, а также тогда, когда операция Dequeue обнаруживает, что второй стек пуст и нужно копировать содержимое первого стека во второй. Для освобождения памяти, занимаемой парой стеков, которая потенциально может разделяться сразу несколькими очередями, требуется воспользоваться подсчётом ссылок на эту пару стеков (можно воспользоваться шаблоном shared_ptr).

Таблица 4: Варианты заданий

23	Graph<V, E> – простой граф, вершины которого имеют атрибуты типа V, а рёбра – атрибуты типа E. Подразумевается, что вершины графа пронумерованы и обращение к ним осуществляется по их номерам. Граф должен быть реализован через матрицу смежности. Граф должен иметь следующие операции: 1. определение, смежны ли две вершины; 2. добавление вершины; 3. добавление ребра; 4. получение атрибута вершины с указанным номером; 5. получение атрибута ребра, соединяющего две вершины. Graph<V, **int**> должен дополнительно иметь операцию построения минимального остовного дерева.
24	Digraph<V, E> – простой орграф, вершины которого имеют атрибуты типа V, а дуги – атрибуты типа E. Подразумевается, что вершины графа пронумерованы и обращение к ним осуществляется по их номерам. Граф должен быть реализован через матрицу смежности. Граф должен иметь следующие операции: 1. определение, существует ли дуга, соединяющая указанные вершины; 2. добавление вершины; 3. добавление ребра; 4. получение атрибута вершины с указанным номером; 5. получение атрибута дуги, соединяющей две вершины. Digraph<V, **int**> должен дополнительно иметь операцию вычисления кратчайшего расстояния между двумя вершинами.
25	Matrix<T,N> – квадратная матрица размера N, элементы которой имеют тип T. Матрица должна иметь следующие операции: 1. запись значения в элемент с индексами (i, j); 2. чтение значения из элемента с индексами (i, j); 3. построение новой матрицы путём удаления i-той строки и j-того столбца. В Matrix<**bool**,N> при $N \leq 8$ матрица должна быть представлена 64-разрядным целым числом, в котором каждому элементу соответствует один бит.

Таблица 5: Варианты заданий

26	Matrix<T,M,N> – матрица размера $M \times N$, элементы которой имеют тип T. Матрица должна иметь следующие операции: 1. запись значения в элемент с индексами (i, j); 2. чтение значения из элемента с индексами (i, j); 3. транспонирование. В Matrix<**bool**,M,N> при $M \leq 8,\ N \leq 8$ матрица должна быть представлена 64-разрядным целым числом, в котором каждому элементу соответствует один бит.
27	IntStack<L, H> – стек целых чисел из диапазона от L до H, имеющий стандартный для стека набор операций. Если размер диапазона не превышает 256, для представления стека должен использоваться массив **char**'ов. В случае диапазона, имеющего размер, не превышающий 65536, должен использоваться массив **short**'ов.
28	IntQueue<L, H> – очередь целых чисел из диапазона от L до H, имеющая стандартный для очереди набор операций и реализованная через кольцевой буфер. Если размер диапазона не превышает 256, для представления очереди должен использоваться массив **char**'ов. В случае диапазона, имеющего размер, не превышающий 65536, должен использоваться массив **short**'ов.
29	IntSet<L, H> – множество целых чисел из диапазона от L до H, реализованное через отсортированный массив и имеющее следующие операции: 1. определение принадлежности числа множеству (реализуется путём поиска числа в массиве делением пополам); 2. добавление числа в множество; 3. удаление числа из множества. В случае, если размер диапазона не превышает 64, для представления множества должно использоваться значение типа **unsigned long**, в котором каждому числу соответствует один бит.
30	IntPQueue<L, H> – очередь с приоритетом, предназначенная для целых чисел из диапазона от L до H и имеющая стандартный для очереди с приоритетом набор операций. Если размер диапазона не превышает 256, для представления очереди должен использоваться массив **char**'ов. В случае диапазона, имеющего размер, не превышающий 65536, должен использоваться массив **short**'ов.
31	IntVector<L, H, N> – целочисленный вектор размера N с элементами из диапазона от L до H, имеющий следующие операции: 1. сложение с другим вектором типа IntVector<L2, H2, N>, в результате которого формируется новый вектор типа IntVector<L+L2, H+H2, N>; 2. скалярное умножение на вектор типа IntVector<L2, H2, N>. Если размер диапазона не превышает 256, для представления вектора должен использоваться массив **char**'ов. В случае диапазона, имеющего размер, не превышающий 65536, должен использоваться массив **short**'ов.

Таблица 6: Варианты заданий

32	Matrix<L, H, M, N> – целочисленная матрица размера $M \times N$, элементы которой принадлежат диапазону от L до H. Матрица должна иметь следующие операции: 1. запись значения в элемент с индексами (i, j); 2. чтение значения из элемента с индексами (i, j); 3. транспонирование. Если размер диапазона не превышает 256, для представления матрицы должен использоваться массив **char**'ов. В случае диапазона, имеющего размер, не превышающий 65536, должен использоваться массив **short**'ов.
33	Polynom<L, H, N> – полином степени N с элементами из диапазона от L до H, имеющий следующие операции: 1. дифференцирование полинома, в результате которого должен формироваться новый полином типа Polynom<L2, H2, N2>, где L2, H2 и N2 вычисляются на базе значений L, H и N; 2. вычисление значения полинома в точке. Если размер диапазона не превышает 256, для представления полинома должен использоваться массив **char**'ов. В случае диапазона, имеющего размер, не превышающий 65536, должен использоваться массив **short**'ов.
34	
35	
36	