Шаблоны

Лекция 9

Понятие шаблона

- ***Шаблоном*** называется конструкция языка программирования, позволяющая определять набор родственных функций или классов
- Отдельные функции (классы) таких наборов различаются только типами обрабатываемых данных и/или типами возвращаемых результатов
- Такие функции (классы) называются ***специализациями шаблона***

Объявление шаблона

- Объявление шаблона начинается с ключевого слова **template**
- Синтаксис объявления:

template <class T>

<объявление/описание функции/класса>

или

template <typename T>

<объявление/описание функции/класса>

Объявление шаблона

- При этом ключевые слова **class** и **typename** взаимозаменяемы, т.е. допускается использование любого из них
- Напомним, что объявлением функции является ее прототип, а объявлением класса – перечисление его полей и прототипов методов
- Описание функции – это заголовок со следующим за ним телом функции
- Описание класса – это перечисление его полей и описание методов

Пример шаблона функции

- Шаблон функции printArray позволяет создавать функции-специализации, выводящие на экран значения элементов массива соответствующих встроенных типов:

```
template <typename T>
    void printArray(T a[ ], int n)
    {   for (int i = 0; i < n; i++)
         cout << a[i] << "  ";
        cout << endl;  }
```

Пример шаблона класса

- Шаблон _Stack< typename T> позволяет создавать классы-специализации, объекты которых являются стековыми структурами:

```
template <typename T> class _Stack
{  int top, n;
   T *a;
 public: _Stack(int);
   T pop();
   void push(T);  };
```

Описание методов

- Описание методов этого класса можно дать вне его объявления, например:

```
template <typename T>
void _Stack<T>::push(T x)
{
    // добавление элемента
    if (top < n)
        a[++top] = x;
}
```

Специализации шаблонов

- После объявления и определения шаблона можно обратиться к его специализациям
- Вызов функций-специализаций выполняется с указанием типа-аргумента, например:

  printArray<**int**>(**int**_arr, 10);
- Объявление объектов классов-специализаций осуществляется с указанием типа-аргумента, например:

  _Stack<**double**> dbl_stack(10);

Специализации шаблонов

- Вызов нестатических методов классов-специализаций производится так же, как и вызов методов обычных классов, например:

```
dbl_stack.push((double)rand() / RAND_MAX * 100);
```

- Для вызова статических методов требуется дополнять имя метода именем типа-специализации с использованием операции разрешения, например:

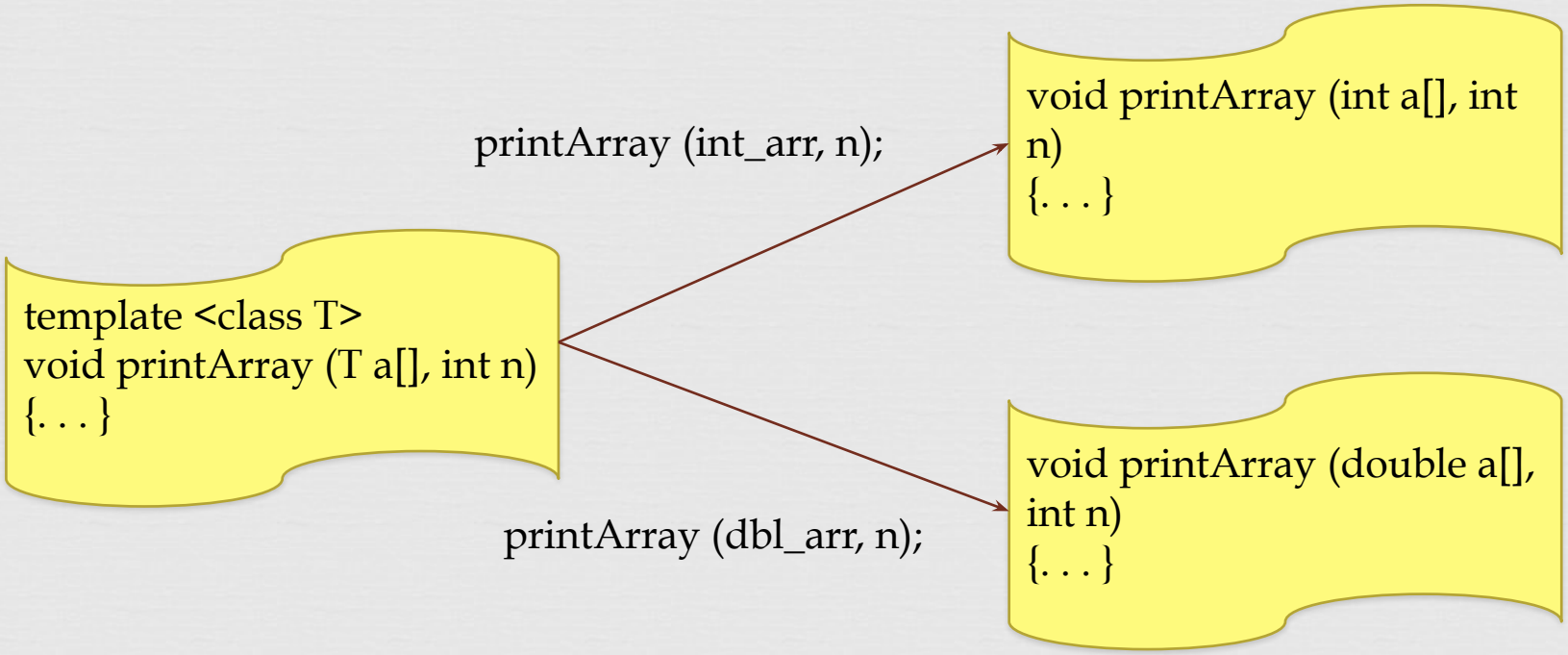
```
_Stack<double>::GetType()
```

вызов метода, возвращающего имя типа-аргумента

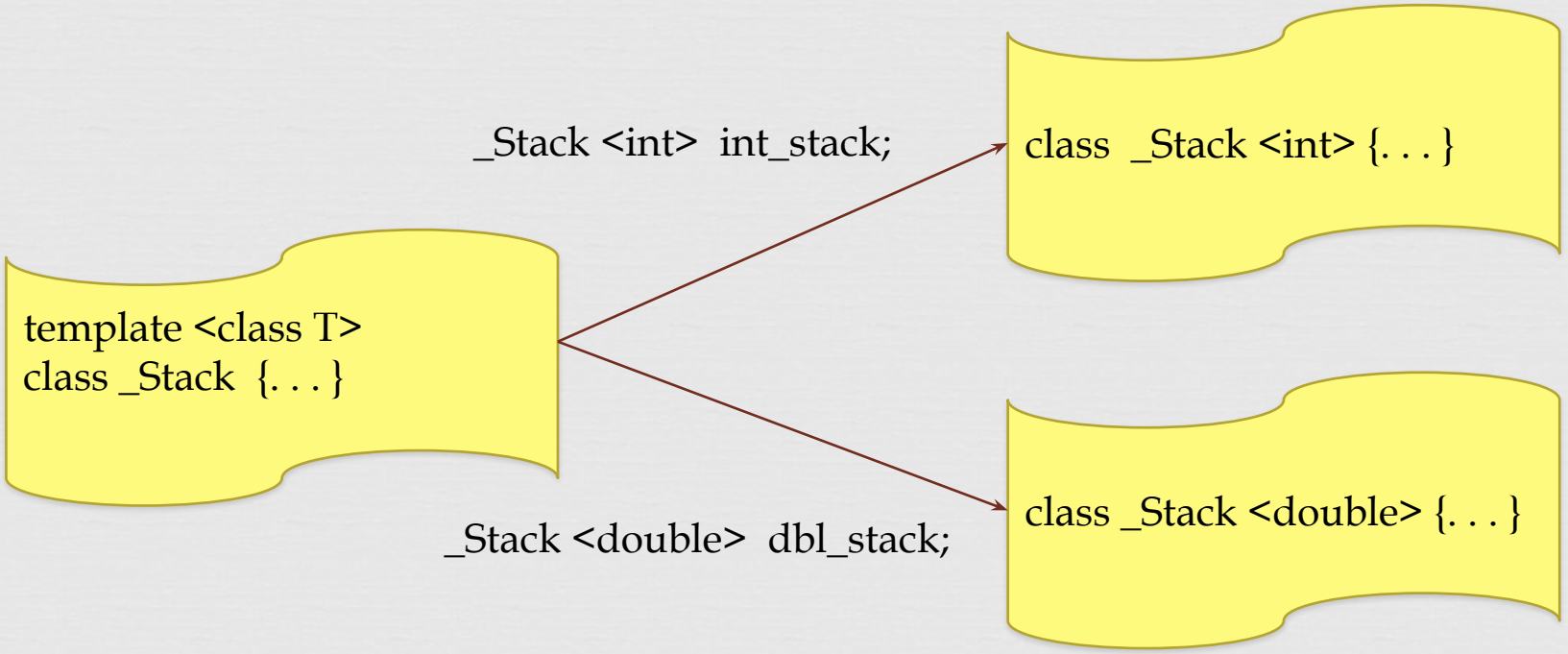
Создание специализаций

- Специализации шаблонов создаются компилятором путем добавления соответствующего кода в формируемый объектный файл
- Функции-специализации создаются при наличии в исходном коде их вызовов
- Классы-специализации создаются при наличии объявлений их объектов, либо вызовов статических методов

Шаблон функции и его специализации



Шаблон класса и его специализации

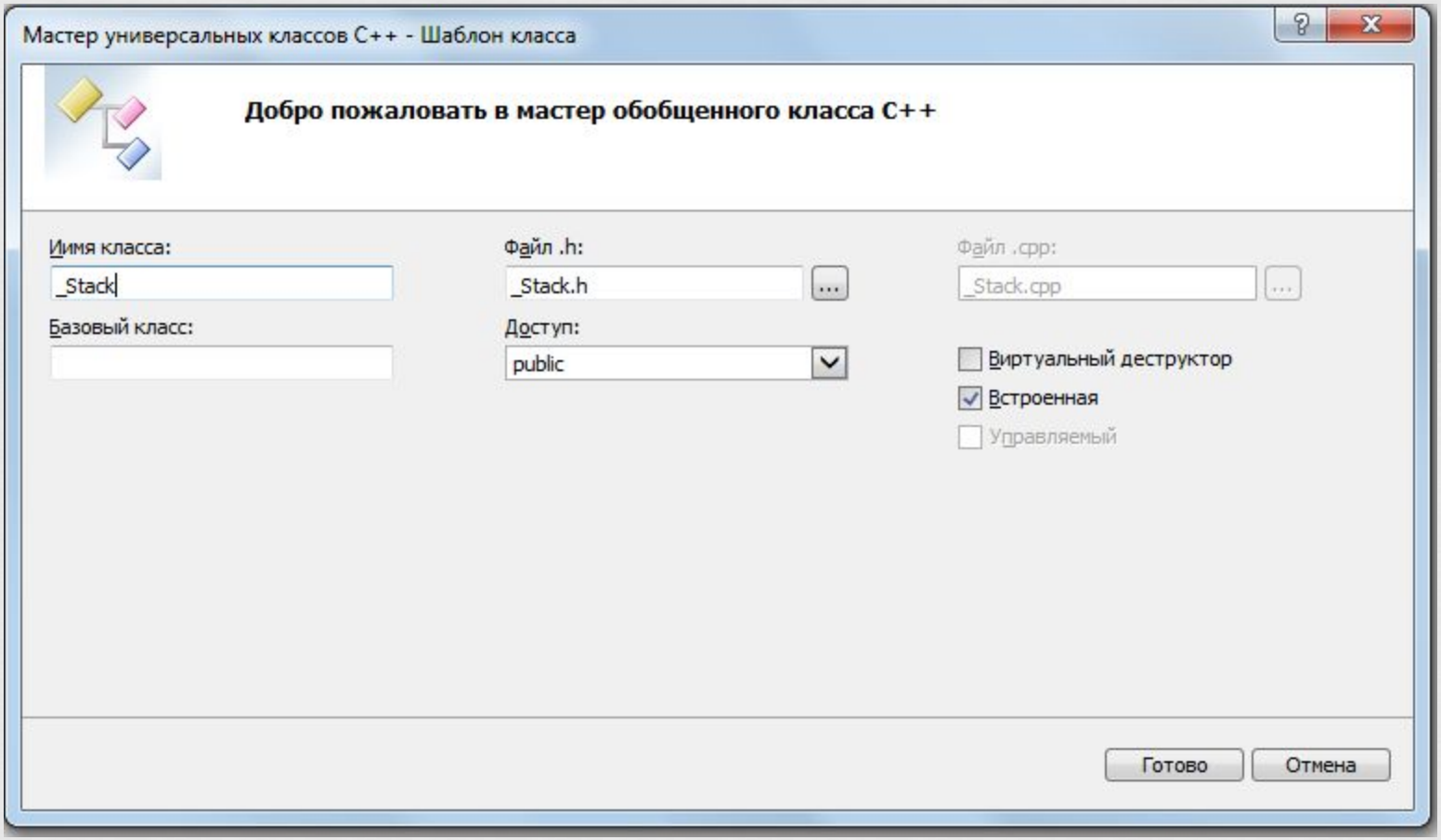


Создание класса-шаблона

- Последовательность действий для создания класса-шаблона в проекте Visual C++

1. Выполнить команду Проект Добавить класс ... Добавить
2. На форме Мастер универсальных классов C++ указать имя класса и установить флажок Встроенная

- Это означает, что объявление и описание нового класса будут находиться в заголовочном файле, а файл реализации типа .cpp создаваться не будет


Мастер универсальных классов C++ - Шаблон класса
Добро пожаловать в мастер обобщенного класса C++
Имя класса:
_Stack
Файл .h:
_Stack.h
...
Файл .cpp:
_Stack.cpp
...
Базовый класс:
Доступ:
public
Виртуальный деструктор
Встроенная
Управляемый
Готово
Отмена

Создание класса-шаблона

3. В созданном заголовочном файле объявить и описать класс-шаблон с использованием типа-параметра

- Описание всех или части методов класса-шаблона может быть вынесено за пределы объявления класса

Библиотека стандартных шаблонов (STL)

STL (Standard Template Library) определяет мощные, организованные в виде шаблонов компоненты, которые реализуют многие распространенные структуры данных и алгоритмы, используемые при их обработке

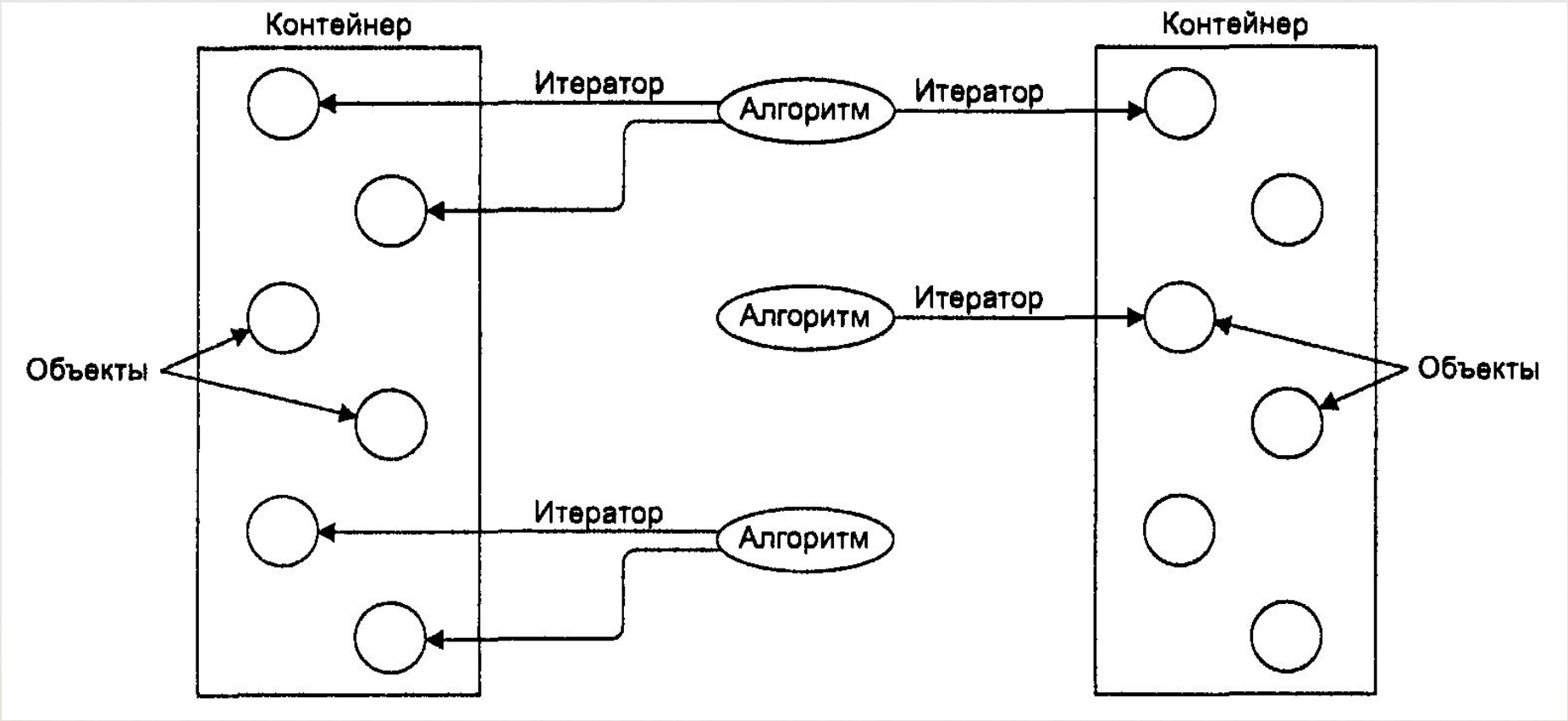
История разработки

- Стандартная библиотека шаблонов разработана в период с 1979 по 1994 год
- Основным разработчиком стал российский программист Александр Александрович Степанов с 1977 года работающий в США
- Другими разработчиками были Мень Ли и Дэвид Мюссер
- В 1994 году STL стала частью официального стандарта языка C++

Основные компоненты STL

- Библиотека STL содержит три основных компонента: контейнеры, итераторы, алгоритмы
- *Контейнеры* представляют собой распространенные структуры данных (массивы, списки, стеки, очереди и др.), реализованные в виде классов-шаблонов
- *Итераторы* являются аналогами указателей и используются для перебора элементов STL-контейнеров
- *Алгоритмы* STL являются функциями, выполняющими различные действия над данными

Контейнеры, итераторы, алгоритмы



Контейнеры

- Контейнеры делятся на три категории – контейнеры последовательностей, ассоциативные контейнеры и адаптеры контейнеров
- *Контейнеры последовательностей* представляют линейные структуры данных (векторы, списки)
- *Ассоциативные контейнеры* являются нелинейными структурами, которые позволяют быстро отыскивать хранящиеся в них элементы (множества, словари)
- *Адаптеры контейнеров* представляют собой ограниченные варианты контейнеров последовательностей (стеки, очереди)

Класс контейнера	Описание контейнера
Контейнеры последовательностей	
vector<T>	Вектор: быстрая вставка и удаление в конце; прямой доступ к любому элементу
list<T>	Список: двусвязный список, быстрая вставка и удаление в любом месте
deque<T>	Кортеж: быстрая вставка и удаление в начале или конце; прямой доступ к любому элементу
Ассоциативные контейнеры	
set<T>	Множество: быстрый поиск, не допускает дубликатов
map<T>	Словарь: отображение «один к одному», дубликаты не допускаются, быстрый поиск по ключу
Адаптеры контейнеров	
stack<T>	Стек: «последним вошел, первым вышел» (LIFO)
queue<T>	Очередь: «первым вошел, первым вышел» (FIFO)

Заголовочные файлы

- Для работы с контейнером необходимо подключить заголовочный файл с соответствующим именем

Контейнер	Заголовочный файл
vector<T>	<vector>
list<T>	<list>
deque<T>	<deque>
set<T>	<set>
map<T>	<map>
stack<T>	<stack>
queue<T>	<queue>

Общие методы контейнеров

Метод	Описание метода
empty()	Возвращает true, если в контейнере нет элементов
size()	Возвращает текущее число элементов в контейнере
operator=	Присваивает один контейнер другому
operator<	Возвращает true, если первый контейнер меньше второго
operator<=	Возвращает true, если первый контейнер меньше или равен второму
operator>	Возвращает true, если первый контейнер больше второго
operator>=	Возвращает true, если первый контейнер больше или равен второму
operator==	Возвращает true, если первый контейнер равен второму
operator !=	Возвращает true, если первый контейнер не равен второму
swap()	Обменивает элементы двух контейнеров

Конструкторы и деструкторы

- Все контейнеры имеют конструкторы по умолчанию и конструктор, инициализирующий контейнер копией существующего контейнера того же типа
- Кроме того, каждый контейнер имеет несколько конструкторов, предлагающих различные способы его инициализации
- Все контейнеры имеют деструкторы для очистки контейнера после того, как он станет больше не нужен

Дополнительные методы

- Контейнеры последовательностей и ассоциативные контейнеры имеют ряд дополнительных общих методов

Метод	Описание метода
max.size()	Максимальное число элементов для контейнера
begin()	Ссылка (итератор) на первый элемент контейнера
end()	Ссылка (итератор) на позицию, за последним элементом контейнера
rbegin()	Ссылка (итератор) на первый элемент обращенного контейнера
rend()	Ссылка (итератор) на позицию, за последним элементом обращ. конт.
erase()	Стирает один или несколько элементов контейнера
clear()	Стирает все элементы контейнера

Итераторы

- В отличие от указателей, являющихся простыми переменными, итераторы – это объекты специальных классов, вложенных в классы контейнеров
- С каждым контейнерным классов связан свой тип итераторов, поэтому внутреннее поведение итераторов зависит от структуры данных (контейнеров), в которых выполняется перебор
- Тем не менее, итераторы различных контейнеров имеют унифицированный интерфейс

Категории итераторов

- В библиотеке STL определены три основных категории итераторов:
  - поступательные,
  - двунаправленные,
  - произвольного доступа
- Кроме того, имеются две категории специализированных итераторов:
  - входные,
  - выходные

Специализированные итераторы

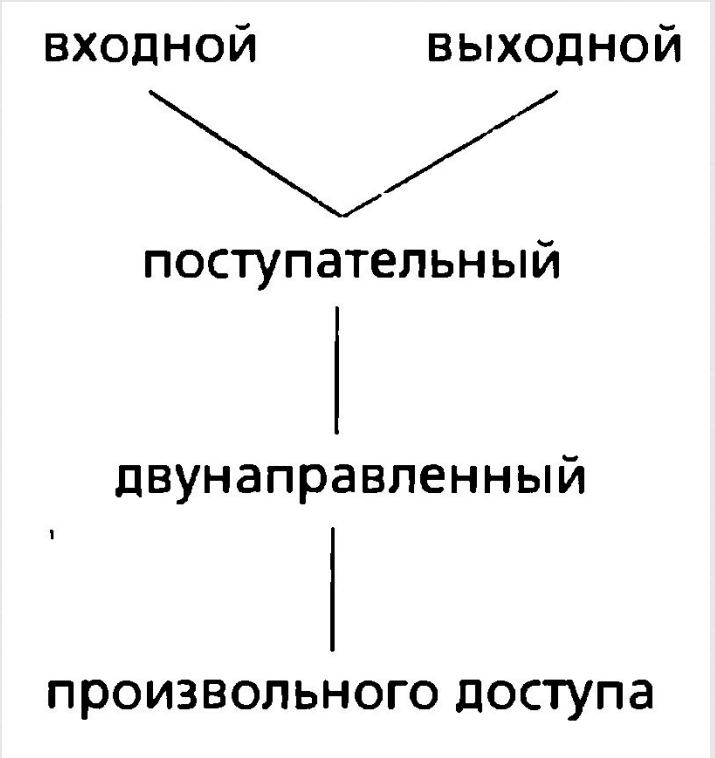
- Специализированные итераторы используются в операциях ввода/вывода
- Входной итератор «указывает» на входной поток (объект cin или файл, открытый в режиме чтения) и обеспечивает последовательное считывание данных в контейнер
- Выходной итератор «указывает» на выходной поток (объект cout или файл, открытый в режиме записи) и обеспечивает последовательный вывод данных из контейнера

Категории итераторов

Категория итератора	Описание возможностей
Входной	Используется для ввода данных в контейнер; продвигается только в прямом направлении на один элемент за шаг
Выходной	Используется для вывода данных из контейнера; продвигается только в прямом направлении на один элемент за шаг
Поступательный	Комбинирует возможности входного и выходного итераторов и хранит их позицию в контейнере
Двунаправленный	Комбинирует возможности поступательного итератора со способностью двигаться в обратном направлении
Произвольного доступа	Комбинирует возможности двунаправленного итератора с возможностью прямого доступа к любому элементу контейнера

Иерархия итераторов

- Видно, что каждая категория итераторов поддерживает все возможности категорий, расположенных выше
- Таким образом, «слабейшие» типы итераторов расположены наверху, а самая мощная – в самом низу
- Таким образом, имеет место иерархия возможностей итераторов



Объявление итераторов

- Объявление итераторов осуществляется способом, универсальным для всех контейнеров
- Прежде всего, нужно отметить, что итераторы делятся на изменяемые и константные
- Изменяемые итераторы используются при переборе элементов контейнеров в режиме чтения/записи, константные – в режиме только чтения
- Перебор элементов контейнеров последовательностей может вестись в прямом, либо обратном направлении

Объявление итераторов

- Соответственно, для объявления итератора может быть использовано одно из следующих слов:
  - iterator
  - const_iterator
  - reverse_iterator
  - coinst_reverse_iterator
- В объявлении итератора необходимо указать тип контейнера с использованием операции разрешения

Объявление итераторов

- Примеры объявления итераторов:

  vector **<int>**::iterator ind;

  list **<double>**::const_iterator pos;

  dequeue**<long>**::reverse_iterator rpos;

Позиционирование элементов

- Итераторы – это «интеллектуальные» указатели и поэтому их можно использовать для доступа к элементам контейнера
- Позицию элемента контейнера сохраняют только итераторы основных категорий и в дальнейшем речь будет идти только о них

Операции над итераторами

- Подобно иерархии возможностей для итераторов существует аналогичная иерархия операций
- Поэтому для итераторов каждой из основных категорий применимы все операции итераторов предшествующих категорий

Поступательные итераторы

Операция	Описание
*iter	Значение элемента в позиции итератора
++ iter	Смещение вперед (возвращает новое значение)
iter ++	Смещение вперед (возвращает старое значение)
iter1 == iter2	Сравнение итераторов на равенство
iter1 != iter2	Сравнение итераторов на неравенство
iter1 = iter2	Присваивание для итераторов

Двунаправленные итераторы

- Для двунаправленных итераторов, помимо вышеперечисленных определены следующие операции

Операция	Описание
-- iter	Смещение назад (возвращает новое значение)
iter --	Смещение назад (возвращает старое значение)

Произвольного доступа

Операция	Описание
iter[n]	
iter += n	
iter -= n	
iter + n	
n + iter	
iter – n	
iter1 – iter2	
iter1 < iter2	
iter1 > iter2	
iter1 <= iter2	
iter1 >= iter2	