Обзор библиотеки STL

Ортогональное пространство STL

В точке начала отсчёта данные не определены, хотя существует тип *void.*

Каждая точка на плоскости *XY* – это совокупность данных любых типов, заключённых в контейнеры.

Каждая точка на плоскости *XZ* – работа какого-то алгоритма с данными простого типа.

Точки на плоскости *YZ* смысла не имеют.

Каждая точка, не лежащая на плоскостях *XY* и *YZ* – это работа алгоритма с контейнером, заполненным данными какого-то типа.

Состав STL:

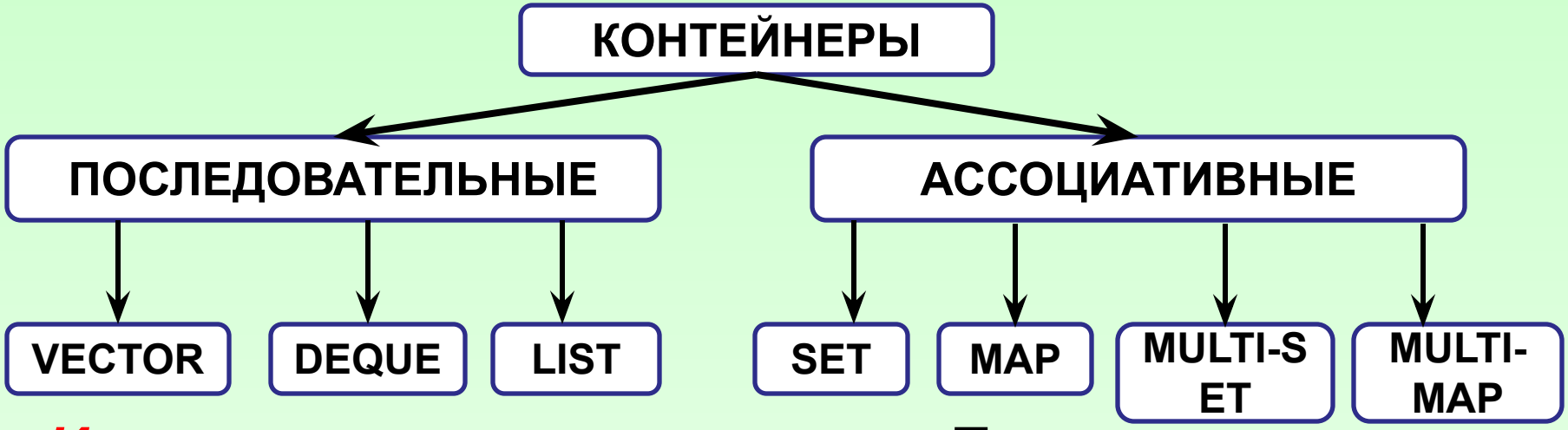
- **контейнеры,**
- **итераторы,**
- **алгоритмы,**
- **аллокаторы,**
- **адаптеры.**

Контейнер **– хранилище, единицами хранения (элементами) являются другие объекты + набор методов, предназначенных для манипулирования элементами контейнера. В контейнере может быть только один тип данных.**

Последовательный контейнер **представляет множество объектов одного типа в строго линейной последовательности.**

Ассоциативный контейнер **обеспечивает быструю выборку данных, соответствующих указанному ключу. Для реализации ассоциативных контейнеров используются двоичные деревья.**

Состав STL:



Итератор **– аналог указателя. Получив итератор какого-то элемента контейнера, при помощи оператора инкремента ++ можно перейти к следующему элементу контейнера, а при помощи -- - к предыдущему элементу.**

Разыменованный итератор – это данные элемента контейнера, => их можно получить или изменить.

Для каждого класса контейнеров определяется свой класс итераторов, однако интерфейсы итераторов разных контейнеров полностью совпадают.

Итераторы

Итераторы **можно назвать посредниками между алгоритмами и контейнерами, потому что многие алгоритмы используют итераторы для того, чтобы перебрать элементы в контейнере.**

Какие функции должен выполнять итератор?

1. **Возможность разыменования того объекта, на который указывает итератор.**
2. **Возможность изменить объект, на который указывает итератор.**
3. **Возможность сравнения итераторов (==), например, определить, дошли ли до конечного элемента контейнера.**

Итераторы, обладающие такими свойствами называются базовыми (*Trivial Iterator*) и практического применения такие итераторы не имеют. К итераторам добавляются новые свойства и получаем новые виды контейнеров.

Итераторы

Input Iterator – **получаем доступ только для чтения данных (необходим инкремент).**

Output Iterator – **нужно уметь записывать данные в контейнер.**

Forward Iterator – **чтение и запись данных (перемещение в одном направлении).**

Bidirectional Iterator – **перемещение не только в прямом, но и в обратном направлениях.**

Random Access Iterator – **произвольный доступ к любому элементу контейнера таким образом, что время доступа к любому элементу не зависит от размера контейнера.**

Итераторы

Кроме этих типов существуют (адаптеры итераторов):

- ***реверсивный*** **или обратный итератор,- перебор элементов осуществляется в обратном порядке (применяются методы *rbegin()* и *rend()*),**
- ***итератор вставки*****, например insert_iterator,**
- ***потоковые итераторы –*** **для использование потоков в/в, для входного потока – istream_iterator, для выходного – ostream_iterator.**

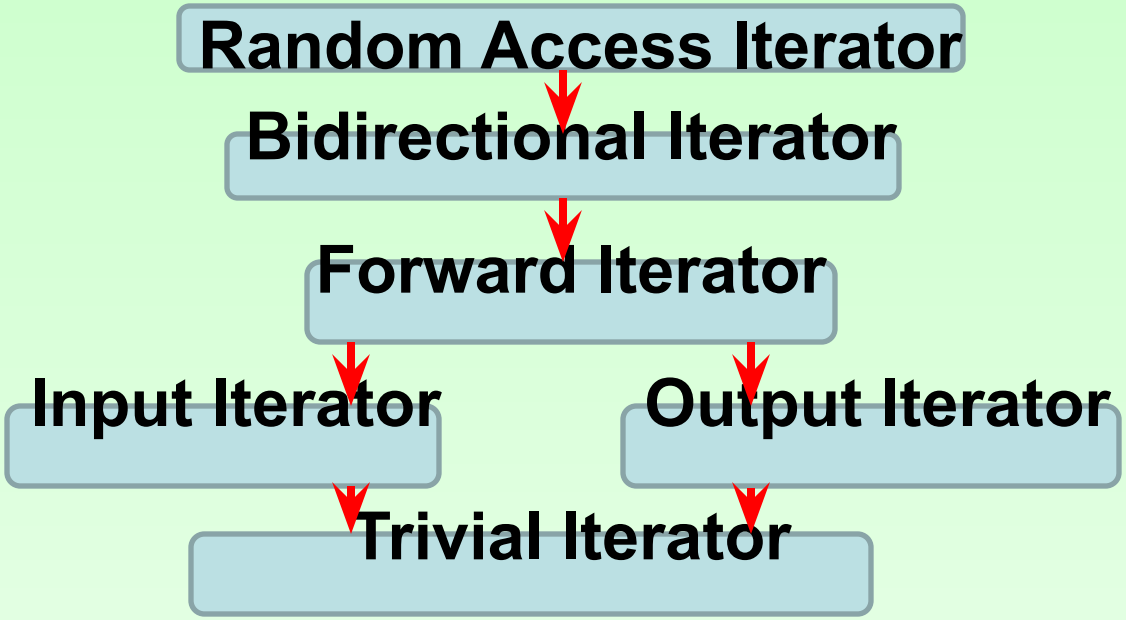
Например:

```
ostream_iterator<int> i(cout,  " ") ;
copy (a, a+N,  i) ; // потоковый итератор – 3-й
                       аргумент
```

или

```
istream_iterator<int> is(cin);
```

Иерархия итераторов



Все итераторы, которые находятся выше, включают в себя функциональность операторов, находящихся ниже.

Во всех контейнерных классов для того, чтобы получить итератор начального элемента, необходимо использовать метод *begin(),* для получения итератора конечного элемента – метод *end().*

Алгоритмы

Алгориты **делятся на несколько категорий:**

- ***немодифицирующие*** **алгоритмы (не изменяющие порядок следования элементов в контейнере) – count, count_if, find, find_if и др.;**
- ***модифицирующие*** **алгоритмы (изменяющие порядок следования элементов в контейнере) - copy, replace, reverse, swap (обмен местами двух элементов) и др.;**
- ***алгоритмы сортировки и поиска*** **(упорядочивание, поиск, слияние и т.д.) – merge, sort, stable_sort (сохраняет порядок для одинаковых элементов), partial_sort (частичная сортировка), max_element и др.;**
- ***алгоритмы работы с множествами*** **– accumulate, includes set_intersection set_difference и др.**
- ***Численные алгоритмы*** **(подключаем заголовочный фай)**

Аллокаторы

Аллокаторы **предназначены для выделения и освобождения памяти, – низкоуровневый интерфейс. Если контейнер выделяет память при помощи аллокатора, то при удалении контейнера можно не заботиться об освобождении памяти, всё делается автоматически.**

Адаптеры **– это классы, которые упрощают интерфейс для доступа к объектам другого класса.**

Одним из недостатков STL является не очень удобная работа с итераторами, т.е. после добавления или удаления элемента в/из контейнера итератор может стать недействительным.

Множества и словари

set (multiset) **– ассоциативный контейнер, который содержит элементы, отсортированные в соответствии с уникальным (неуникальным) ключом. Сортировка производится с использованием функции Compare. Операции поиска, удаления и включения имеют логарифмическую сложность.**

std::set

```
template<
    class Key,
    class Compare = std::less <Key>,
    class Allocator = std::allocftor <Key>
> class set;
```

Множества и словари

map (multimap) **– ассоциативный контейнер, который отсортированные список пар в соответствии с уникальным (неуникальным) ключом. Сортировка производится согласно функции Compare. Операции поиска, удаления и включения имеют логарифмическую сложность.**

std::multimap

```
template<
    class Key,
    class T,
    class Compare =std::less<Key>,
class Allocator = std::allocator<std::pair<const Key, T> >
> class multimap;
```

Пример использования словаря

Составить программу учета заявок на авиабилеты.

Каждая заявка содержит: пункт назначения, номер рейса, фамилию и инициалы пассажира, желаемую дату вылета.

Программа должна обеспечивать выбор с помощью меню и выполнение одной из следующих функций:

- добавление заявок в список;
- удаление заявок;
- вывод заявок по заданному номеру рейса и дате вылета;
- вывод всех заявок, упорядоченных по пунктам назначения;
- вывод всех заявок, упорядоченных по датам вылета.

Хранение данных организовать с применением контейнерного класса `multimap`, в качестве ключа использовать «пункт назначения».

Создаём класс *CTicketInfo,* который будет содержать все необходимые поля для заявки, а также этот класс будет использоваться при описании словаря *multimap.*

Описание класса

```
class CTicketInfo
{
public:
    long ticket_id;  // идентификатор билета
    long n_reis;     // номер рейса
    string dest;     // пункт назначения
    string fio;      // ФИО пассажира
    string date;     // дата вылета
};
#include <map>
multimap<long, CTicketInfo> m_Tickets;
```

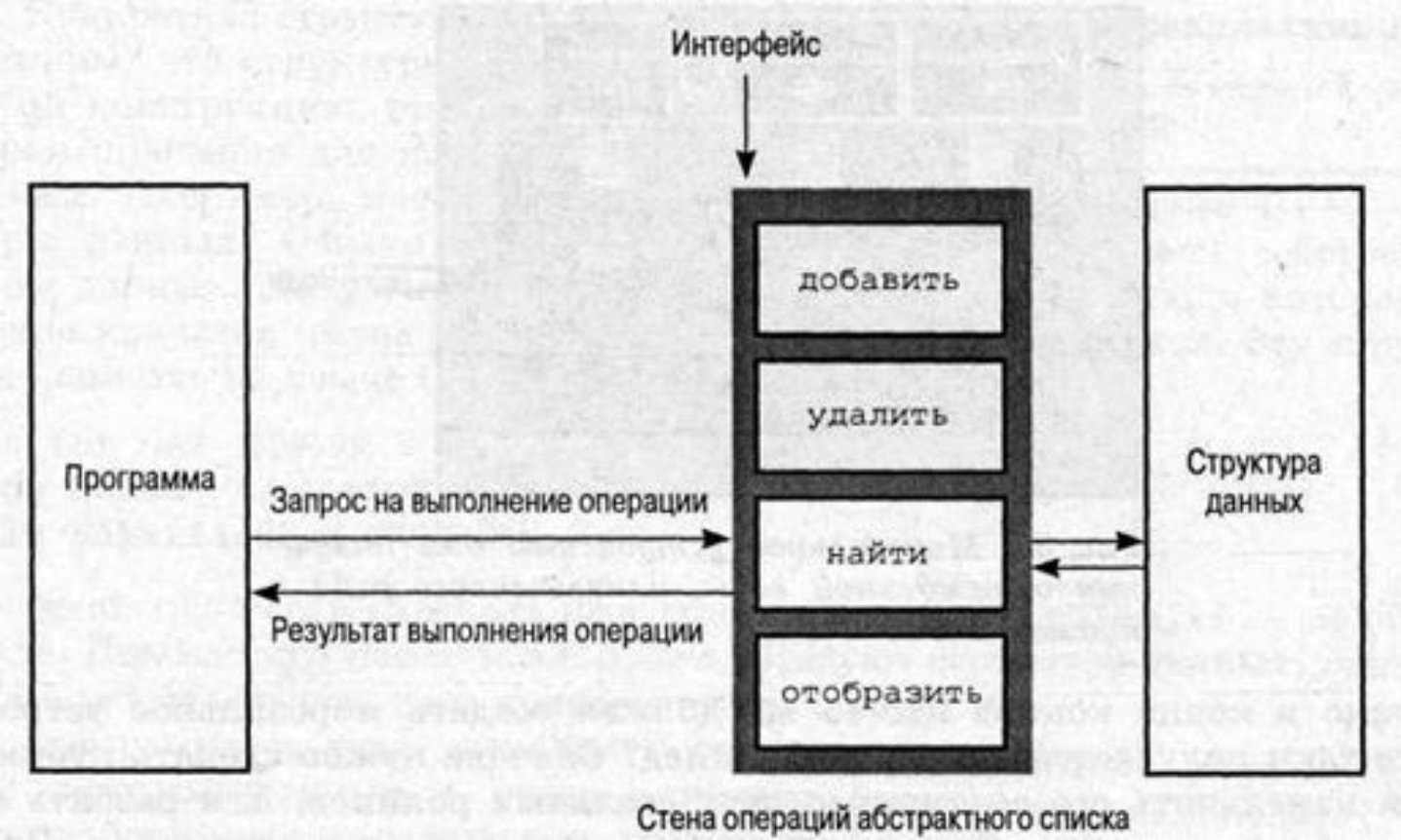
int add_ticket()

```
int add_ticket()
{   setlocale(LC_ALL,"Russian");
    long reis_n, id_ticket;
    char ch[30]; string str;
    CTicketInfo tickets;
    multimap<long, CTicketInfo>::iterator it;
    bool yes(false), no(false), item_not_found(false);
    char user_respond[2], respond_no[ ] = "n",
respond_yes[ ] = "y";
    while (strcmp(respond_no, user_respond) != 0)
    {   system("cls");
    cout << "Добавление заявки: (en words only)\n";
        cout << "Введите номер заявки: ";
        cin >> id_ticket;
        it = m_Tickets.find(id_ticket);
        if (it != m_Tickets.end())  и т.д.
```

Абстракция данных

Абстрактный тип данных **– это совокупность данных и операций над ними.**

Структура данных **– это конструкция, определённая в языке программирования для хранения набора данных.**



Абстрактный список

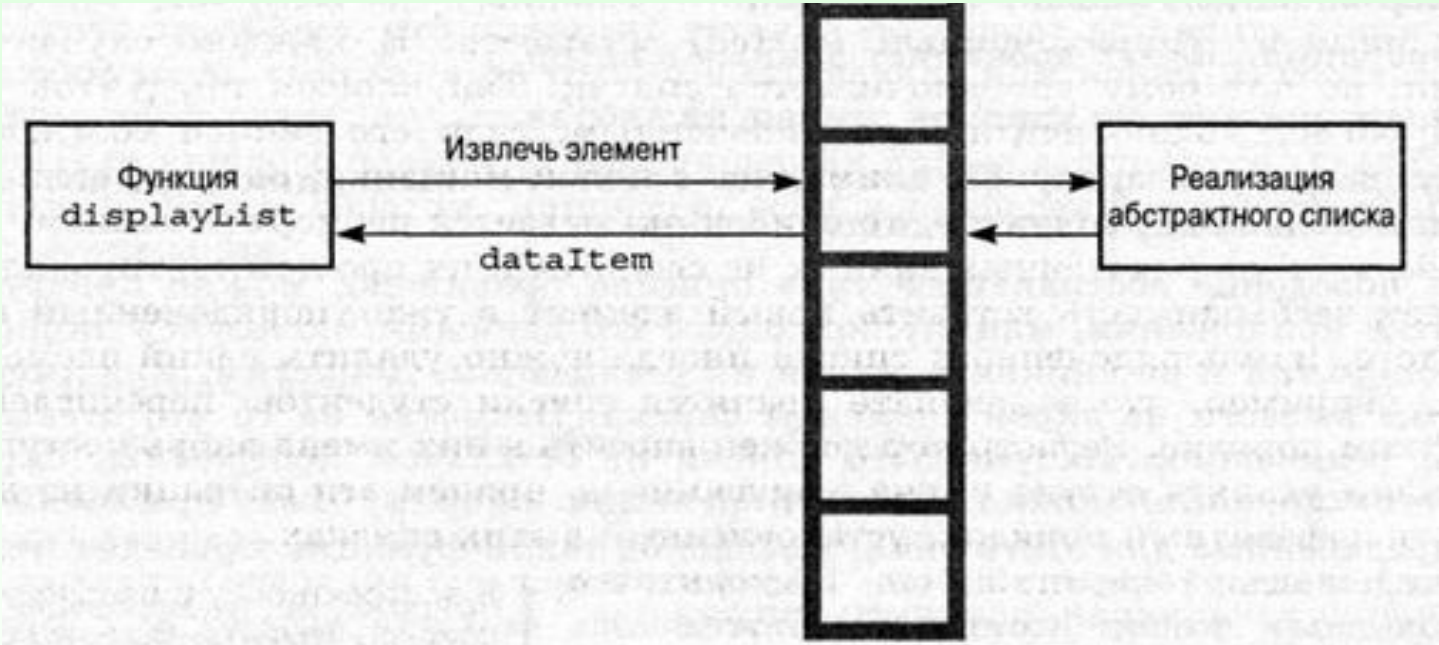
Нам может понадобиться доступ к любому элементу списка. Это значит, что мы можем просматривать элемент, находящийся на позиции *i*, удалять его или вставлять новый элемент в эту позицию. Эти операции являются частью абстрактного типа данных под названием список *(list).*

Операции над абстрактным списком:

1. **Создать пустой список.**
2. **Уничтожить список.**
3. **Определить, пуст ли список.**
4. **Определить кол-во элементов в списке.**
5. **Вставить элемент в указанную позицию списка.**
6. **Удалить элемент, находящийся в указанной позиции списка.**
7. **Просмотреть (извлечь) элемент, находящийся в указанной позиции списка.**

Абстрактный список

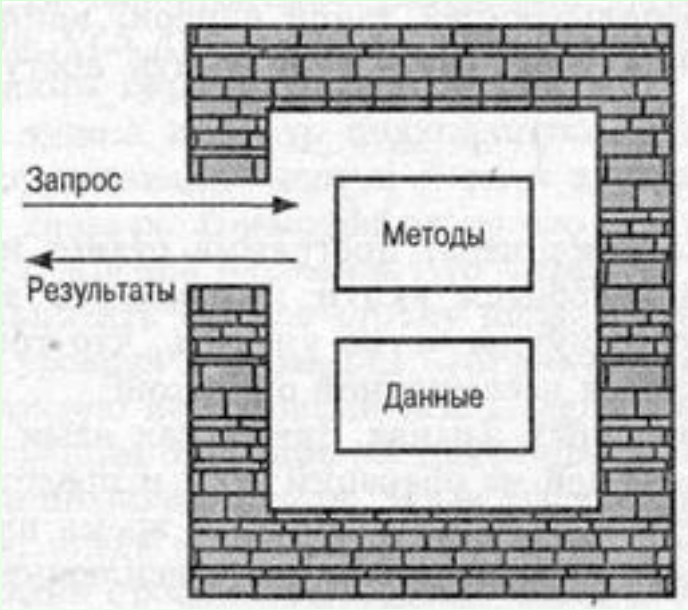
Принципиально важно, что *спецификация абстрактного типа не затрагивает вопросов его реализации.* Именно это ограничение позволяет воздвигать стены между реализацией АТД и программой, использующей его. Такая программа называется клиентом (client). Единственным обстоятельством, влияющим на выполнение программы, является содержание самой операции.



Стена операций абстрактного списка

Данные и методы, инкапсулированные в объекте

Инкапсуляция объединяет данные АТД с его операциями, называемыми *(methods),* образуя объекты *(objects).* Инкапсуляция скрывает детали реализации.



Основные положения:

- **Класс языка C++ определяет новый тип данных.**
- **Объект – это экземпляр класса.**
- **Конструктор создаёт и инициализирует объект.**
- **Данные-члены должны быть закрытыми.**
- **Константные функции не могут изменять данные-члены класса.**
- **Конструктор по умолчанию не имеет аргументов.**
- **Файл реализации содержит определения всех функций-членов класса и др.**

Абстрактный стек

Операции над абстрактным стеком:

1. Создать пустой стек.
2. Уничтожить стек.
3. Определить, пуст ли стек.
4. Добавить в стек новый элемент.
5. Удалить из стека элемент, добавленный последним.
6. Извлечь из стека элемент, добавленный последним.

Абстрактная очередь

Операции над абстрактной очередью:

1. Создать пустую очередь.
2. Уничтожить очередь.
3. Определить, пуста ли очередь.
4. Добавить в очередь новый элемент.
5. Удалить из очереди элемент, поставленный туда раньше всех.
6. Извлечь из очереди элемент, поставленный туда раньше всех.

Бинарные деревья

Операции над абстрактной бинарным деревом:

1. **Создать пустое бинарное дерево.**
2. **Создать бинарное дерево, содержащее один узел, по заданному элементу.**
3. **Создать бинарное дерево по заданному корню и двум бинарным поддеревьям этого корня.**
4. **Уничтожить бинарное дерево.**
5. **Определить, пусто ли бинарное дерево.**
6. **Определить или изменить данные, записанные в корне бинарного дерева.**
7. **Присоединить к корню бинарного дерева левый или правый дочерний узел.**

Бинарные деревья (продолжение)

Операции над абстрактной бинарным деревом:

8. Присоединить к корню бинарного дерева левое или правое поддерево.
9. Отсоединить от корня бинарного дерева левое или правое поддерево.
10. Вернуть копию левого или правого поддерева корня бинарного дерева.
11. Обойти узлы бинарного дерева в прямом, симметричном или обратном порядке.

Графы как абстрактные типы данных

Операции над абстрактным графом:

1. Создать пустой граф.
2. Уничтожить граф.
3. Определить, пуст ли граф.
4. Определить количество вершин в графе.
5. Определить количество рёбер в графе.
6. Определить, существует ли ребро, соединяющее два заданных ребра.
7. Вставить вершину в граф. Поисковые ключи, хранящиеся в вершинах, должны отличаться от ключа новой вершины.
8. Вставить ребро, соединяющее две заданные вершины графа.
9. Удалить из графа указанную вершину, а также все рёбра, соединяющие её с другими вершинами.