Тема 8. Част II Стандартна темплейтна обектна библиотека *STL*

3. Компонентно пространство, декомпозиция.
4. Основни компонени
4.1. Контейнерни класове-Вектор (Vector)

3. Компонентно пространство

Дефиниция:

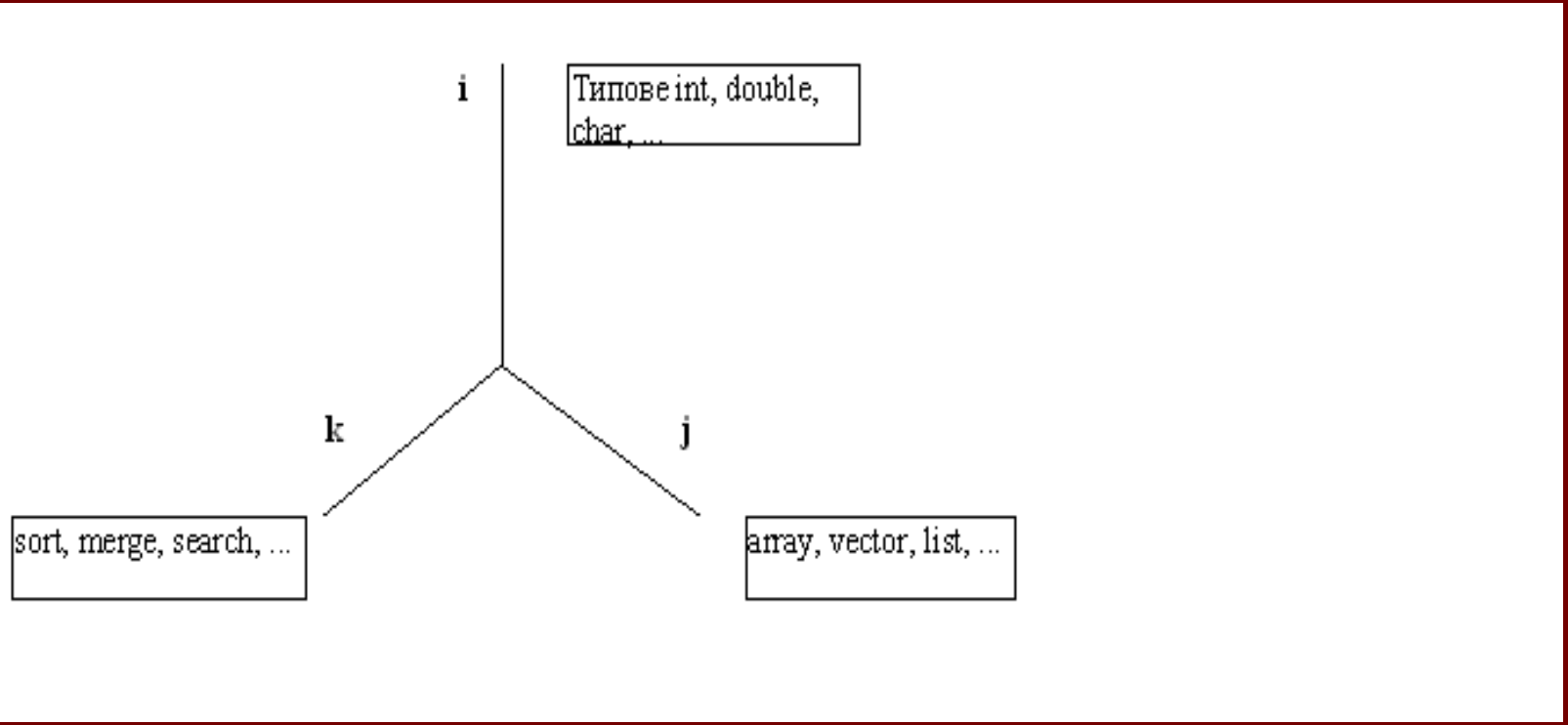
STL е компонентна библиотека.

Съставни части (компоненти):

- Контейнери - това са обекти, които съхраняват обекти от произволен тип;
- Итератори – обекти за достъп до елементите на даден контейнер;
- Алгоритми – стандартизирани функции с обща заглавна част. Могат да работят:
  - Върху STL контейнери;
  - С базови типове за съхранение - масиви;
  - Със създадени от потребителя контейнери;

Могат да се създават нови компоненти (да се разширяват).

3. Компонентно пространство



3. Компонентно пространство-декомпозиция

По този начин се различават, i*j*k различни версии на кода, който може да се програмира.

Пример: сортиращ алгоритъм за масив от int, double...

алгоритъм за търсене за списък от int, double ...

Намаляване на версиите на кода се постига по два начина:

- С използването на темплейтни функции, които са параметризирани по типовете данни, i-тата размерност (по тип на данните) се премахва, с което версиите на кода, който трябва да се проектира се намаляват на j*k, тъй като се създава само една имплементация, напр. на списък, която съхранава обекти от произволен тип;
- Със създаването на алгоритми, които да работят с различни контейнери-алгоритъм, който да работи с масив, списък, вектор и т.н.

Резултат: Остават само j+k версии на кода, който трябва да се проектира.

4. Основни компоненти

- Контейнер (Container):
- Итератор (Iterator):
- Алгоритъм (Algorithm):
  - Обект (клас) функция (Function Object):
- Адаптер (Adaptor):

4. Основни компоненти

- Контейнер (Container):

Дефиниция: **Обект, който е способен да съхранява и обработва обекти;**

4. Основни компоненти

- Итератор (Iterator):

Дефиниция: **Абстракция на алгоритмите, която позволява достъп до контейнерите по такъв начин, че алгоритмите да имат възможност да се прилагат върху различни контейнери;**

4. Основни компоненти

- Алгоритъм (Algorithm):

Дефиниция: **Изчислителна функция, която може да работи с различни контейнери;**

4. Основни компоненти

- Обект функция (Function Object):

Дефиниция: **Клас, който има дефиниран оператор за обръщение към функция - operator();**

4. Основни компоненти

- Адаптер (Adaptor):

Дефиниция: **Декапсулира компонент за да му предостави друг интефейс:**

- **Вследствие може да дефинира стек „адаптирайки" списък;**
- **Основа – наследяване.**

4. Основни компоненти-Организация на STL

Основни Microsoft include файлове:

<VALARRAY>	Вектори и операции
<algorithm>	Имплементация на общи алгоритми
<CSTDLIB>	bsearch(), bsort()
<ITERATOR>	Имплементация на итератори и итераторни адаптери
<functional>	Оператори, функционални обекти и адаптери
<CASSERT>	Макроси на асерции
<CCTYPE>	Класификация на символи
<CWCTYPE>	Класификация на разширени символи
<CERRNO>	Обработка на грешки в стил C
<exception>	Обработка на изключенията
<CFLOAT>	Макроси от стил C за граници на числа с плаваща запетая
<STACK>	Включва всички адаптери на контейнера
<NEW>	Управление на динамичната памет
<Streambuf>	Буфери на потоци
<IOMANIP>	Манипулатори
<IOS>	Базови вх/изх потоци
<IOSFWD>	Декларации на I/O средства
<IOSTREAM>	Стандартни вх/изх обекти и операции

4. Основни компоненти-Организация на STL

<ISTREAM>	Шаблон за входен поток
<MEMORY>	Подразбиращ се алокатор на контейнери
<BITSET>	Вектор от битове, последователен контейнер
<DEQUE>	Двусвързана опашка, последователен контейнер
<LIST>	Списък, последователен контейнер
<MAP>	Карта, мултикарта -асоциативен контейнер
<SSTREAM>	Потоци от/към низове
<SET>	Множество, мултимножество - асоциативен контейнер
<CSTRING>	Функции за низове от стил C
<VECTOR>	Едномерен масив-последователен контейнер
<STRING>	Стринг, базов тип за съхраняване на символни низове
<LIMITS> <CLIMITS>	Граници на числовите интервали
<QUEUE>	Опашка с два края - последователен контейнер
<OSTREAM>	Шаблон за изходен поток
<CTIME>	Дати и час в стил C
<FSTREAM>	Потоци от/към файлове
<XTREE>	Дървета

4. Основни компоненти-Организация на STL

<XTREE>	Дървета
<LOCALE>, <CLOCALE>	Местни формати за извеждане на данните (дата, валута и др)
<COMPLEX>	Комплексни числа и операции
<CSTDIO> <CWCHAR>	Стандартен вх/изх в стил C (за разширени символи)
<CMATH>	Стандартни математически функции
<NUMERIC>	Общи числови изчисления
<CSTDLIB>	Случайни числа в стил C
<UTILITY>	Оператори и двойки
<CSTDARG>	Списъци на аргументи на функции с променлива дължина
<CSTDDEF>	Библиотека от C за поддръжка на езика
<stdexcept>	Стандартни изключения

4.1. Контейнерни класове

Дефиниция (Бьорн Строуструп)

Контейнерни класове са класове, които съхраняват обекти от произволен тип (прости типове или обекти от класове):

- Едни от най-използваните класове при програмиране на приложения;
- Не зависят от езика на програмиране;
- Основен компонент на STL;
- Те съхранят елементите от определен тип в даннова структура.

Съхраняването на стойностите на елементите е основният етап на кодирането на всяка програма.

4.1. Контейнерни класове

Класификация

- Контейнерите се класифицират по начина на подреждане на елементите -

(структурната им организация).

4.1. Контейнерни класове

По този признак в STL се разграничават две **основни групи контейнери**:

- *Последователни контейнери* (*Sequence Containers*)
- *Асоциативни контейнери* (*Associative Containers*).

4.1. Контейнерни класове

Дефиниции (по стандарта **ANSI/ISO)**:

- **Последователен** е такъв вид контейнер, който организира крайно множество от обекти от еднакъв тип в строга линейна последователност (*Sequence Container)*
- **Асоциативен е** контейнер,които предоставя възможност за бърз достъп и получаване на данните, основан на ключове за търсене. *(Associative Container)*

4.1. Контейнерни класове

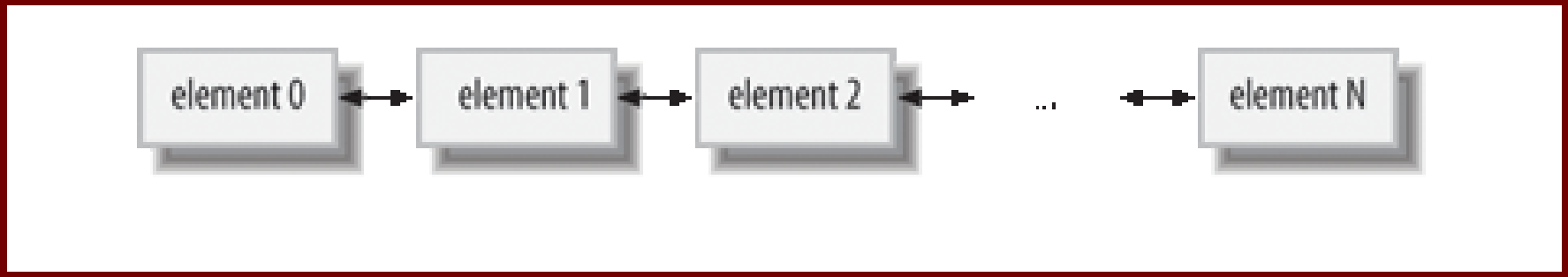
Видове последователни контейнери

STL предоставя **три основни вида (подгрупи)** *на последователни контейнери в зависимост от вътрешното представяне (имплементация):*

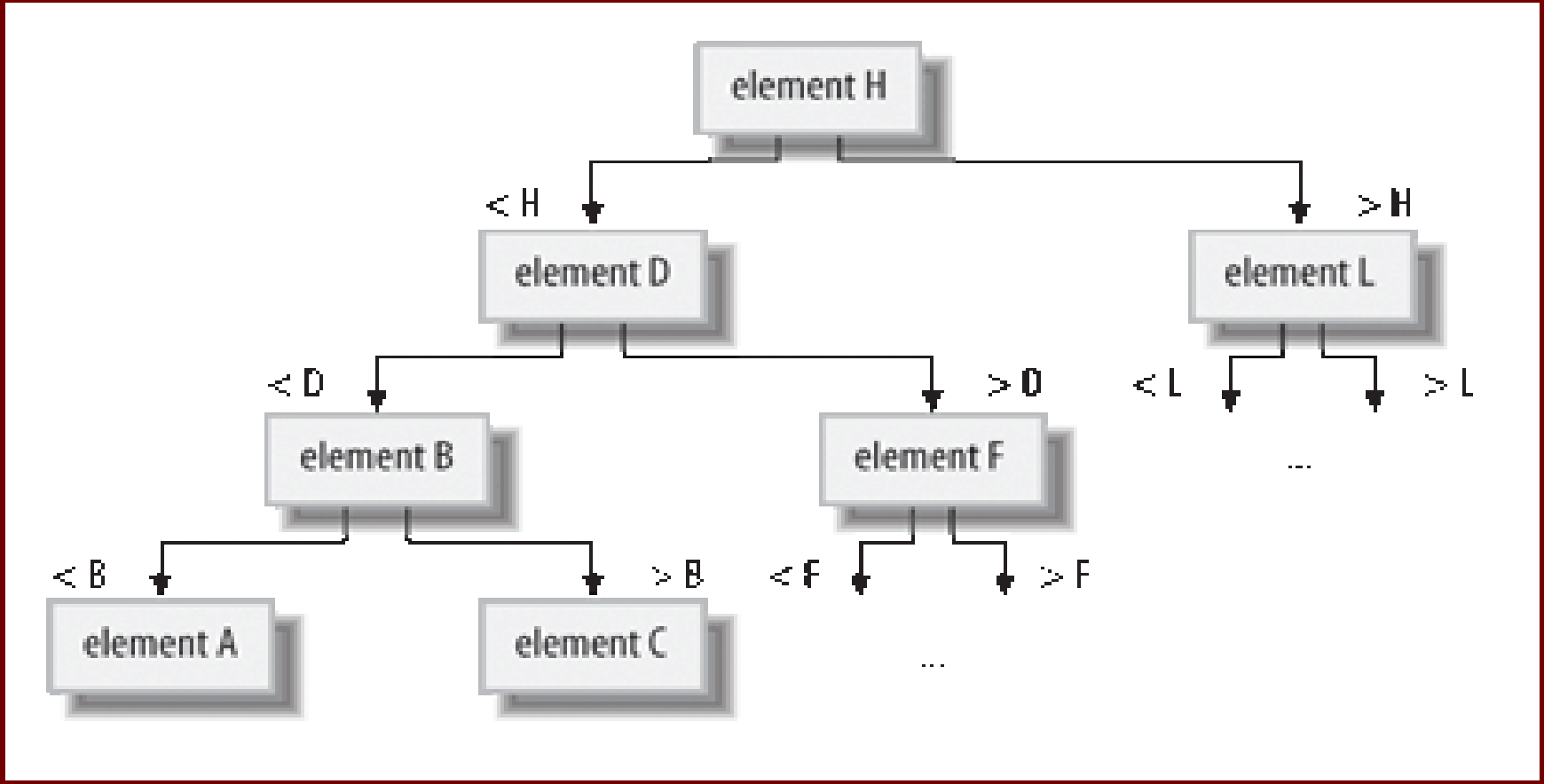
- *Вектор (Vector*);
- *Списък (List)* ;
- *Опашка (Deque).*

Deque *(съкратено от Double Ended Queue)* е обозначение на опашка, управлявана с два указателя - двойно свързан списък.

4.1. Контейнерни класове (последователни контейнери)



4.1. Контейнерни класове
(Асоциативни контейнери)



4.1. Контейнерни класове

При ***асоциативните контейнери*** елементите са структурирани и за получаването им се използва **двоично търсене**.

В зависимост от структурата на всеки от възлите:

- ключовата стойност уникална ли е?

- Ако ключовата стойност е уникална- всяка стойност на ключа може да съхранява **само един елемент**:
  - Множество (set)
  - Карта (map)

4.1. Контейнерни класове
(Асоциативни контейнери)

В зависимост от структурата на всеки от възлите

Видове асоциативни контейнери:

- Ако за всеки ключ могат да се съхраняват **повече от една стойност**, т.е ключовата стойност **не е уникална:**
  - Мултимножество (*multiset*)
  - Мултикарта (*multimap*)

4.1. ***Контейнерни класове:*** *Вектор (vector)*

Дефиницията на класа в стандартната библиотека е следната:

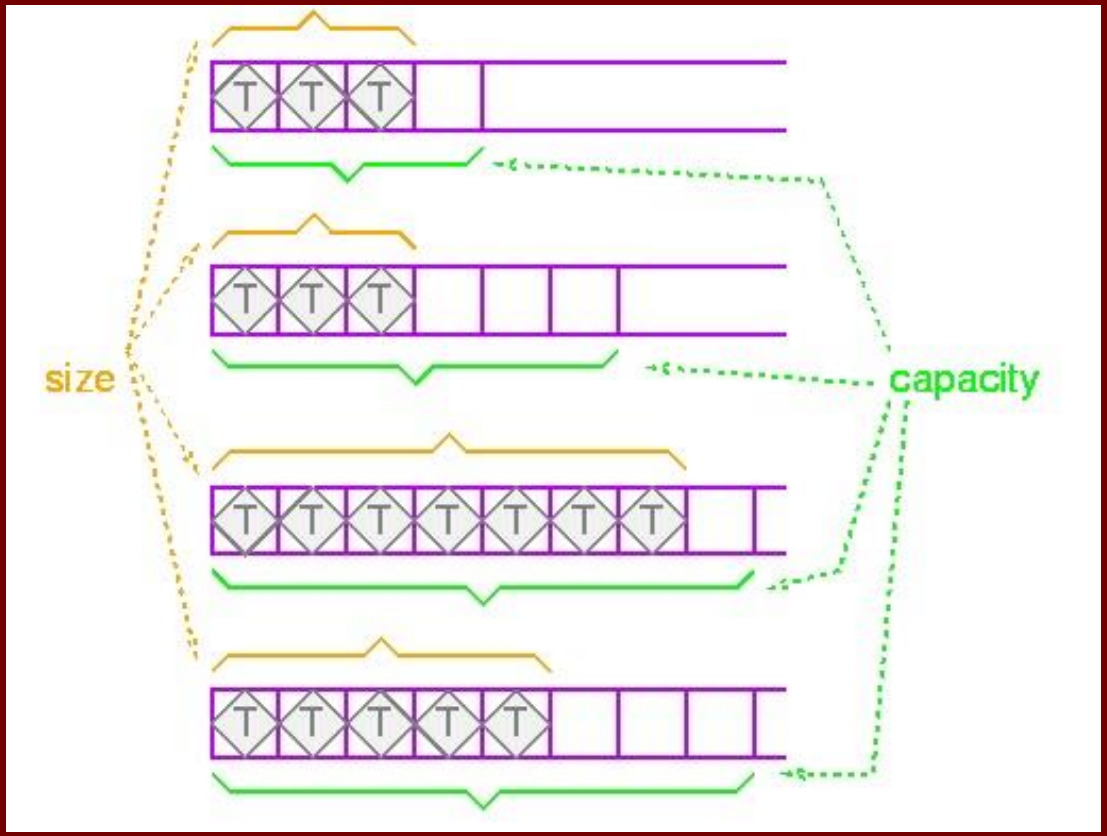
```
template<class T, class _A = allocator<T> >
class vector {
    //дефиниция на типове:
typedef typename _Mybase::value_type value_type;
typedef typename _Mybase::size_type size_type;
typedef typename _Mybase::iterator iterator;
typedef typename _Mybase::const_iterator const_iterator;

    //дефиниция на член променливи
    //дефиниция на член функции
};
```

Дефинирани са като статични членове типовете:

- На размера;
- На стойността;
- Итератора …

4.1. ***Контейнерни класове:*** *Вектор (vector)*

Организация и основни параметри:



4.1. ***Контейнерни класове:*** *Вектор (vector)*

Синтаксис на създаване на обекти:

vector<конкретен_тип> <име на обекта> [варианти на конструктора];

където конкретен_тип е избрания от потребителя тип.

Пример:

string - вектор от символите, съхранени в контейнера;

Особености:

- Не се изисква шаблонния спецификатор (подразбира се);
- Съхранява базовия тип (char*) в клас;
- Поддържа векторните операции;
- Поддържа операциите на базовия тип;
- Динамичен...

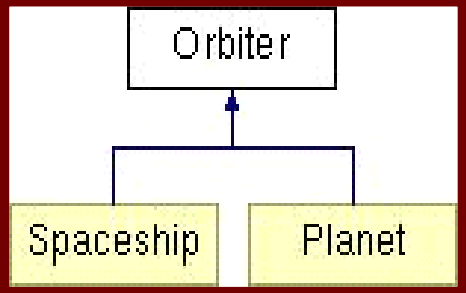
4.1. ***Контейнерни класове:*** *Вектор (vector)*

Примери за създаване и използване:

```
vector<int> int_vector (10);
vector<char>    char_vector (10);
vector<CXY>     XY_vector (10);

vector<Planet> sun_planets_vector;
int current_size = sun_planets_vector.size();
cout << "current vector size: " << current_size<<endl;
```

Изход: current vector size: 0

```
vector<Spaceship> ships_vector;
int current_size = ships_vector.size();
cout << "current vector size: " << current_size<<endl;
```



4.1. ***Контейнерни класове:*** *Вектор (vector)*

- За получаване на големината (от тип value_type) на контейнера се използва и тип int.
- Определяне на името на типа на обектите, които могат да се записват в контейнера се използва typeid и функцията му name().

Пример:

```
vector<Orbiter> v = new vector<Spaceship>();
cout << "value type: " << typeid (v).name();
```

4.1. ***Контейнерни класове:*** *Вектор (vector)*

Основни операции:

Контейнерът има основни операции:

- Добавяне;
- Изтриване;
- Промяна.

на обекти в него.

Векторът предоставя член функции (оператори) за тази цел.

4.1. ***Контейнерни класове:*** *Вектор (vector)*

Свойства и ефективност на основните операции:

Функционалността на вектора гарантира:

- Добавянето и изтриването на края на вектора да се извършва за амортизирано постоянно време O(1);
- Добавянето и изтриването в средата да се извършва за линейно време O(n).

4.1. ***Контейнерни класове:*** *Вектор (vector)*

В съответствие с дефиницията на стандарта, съществуват следните понятия:

- амортизирано време;
- амортизирано постоянно време;
- линейно време;
- константно време

4.1. ***Контейнерни класове:*** *Вектор (vector)*

Дефиниции:

- амортизирано (линейно) време.

Амортизираното време може да бъде определено като времето, за изпълнение на операцията върху някакъв контейнер, при което същото може да се изменя в големи граници, когато се изпълняват последователните операции, но общото време за последователност от N операции има по-добра граница, отколкото времето за N кратно изпълнение на най-лошия по време случай.

4.1. ***Контейнерни класове:*** *Вектор (vector)*

- Амортизирано константно време Приблизително равно на времето, използвано при работа със C/C++ масиви, но може да бъде най-много равно, но не и по-добро.

Векторът е динамичен тип:

- Той има качеството да увеличава автоматично големината си при добавяне/изтриване;
- Преоразмеряване се извършва, когато се изпълнява операцията insert, но няма алокирано място за съхраняване на новодобавения обект;
- Алокира се място за n+1 елемента (където n е текущата големина на контейнера);
- Копират се n-те съществуващи елемента в новоалокираното пространство в паметта;
- Алокацията и копирането се изпълнява за линейно време. След това новият елемент се добавя.

4.1. ***Контейнерни класове:*** *Вектор (vector)*

Ефективност:

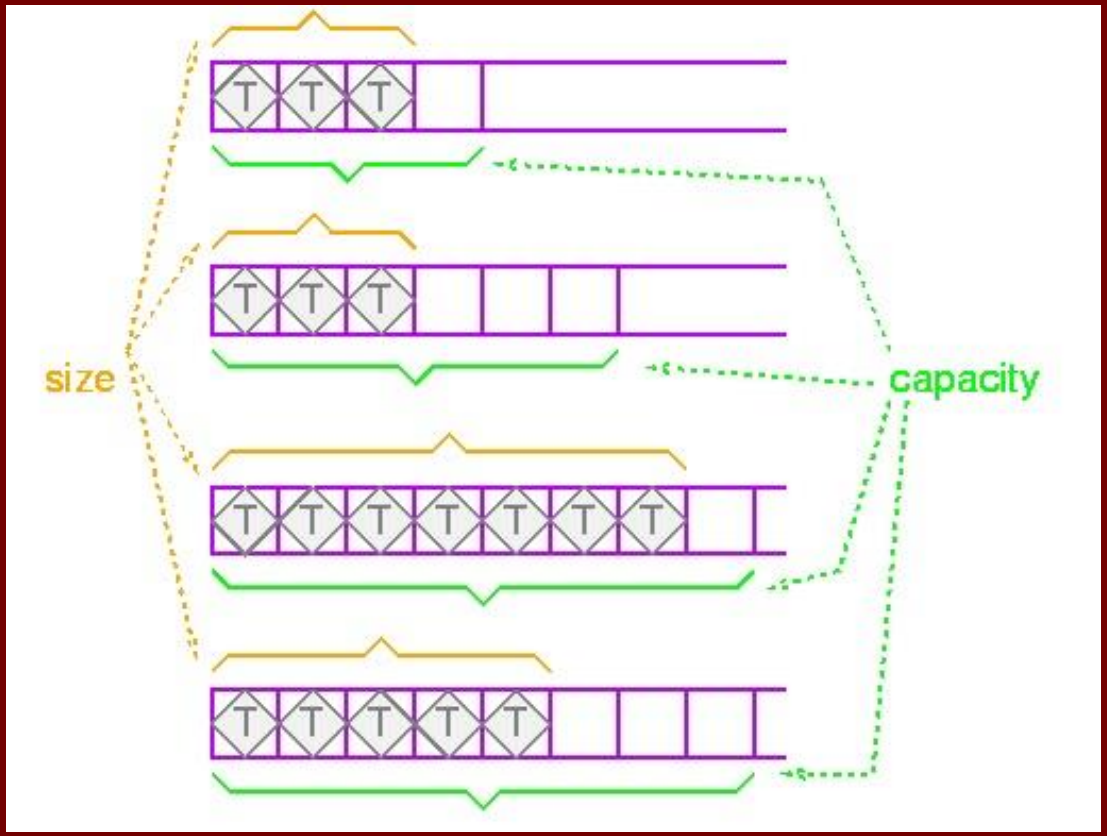
Порядък O(n) за n вмъквания:

Изпълняват приблизително около n операции за добавяне:

- За всяко добавяне се изисква O(1) време;
- Това определя общо време за добавяне, което се оценява в най-лошия случай като време O(n) за всичките n операции добавяне.

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

Капацитет и големина и член функции за имплементацията им:



4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

- Капацитет и големина - O(1):

vector<int> iv;

Член функцията **capacity()** връща броя на елементите, за които е алокирана памет:

vector<int>::size_type capacity = iv.capacity();

cout << "capacity: " << capacity <<endl;

Изход:capacity: 0

Брой на заетите с обекти елементи - **size()** <= капацитета

int current_size = iv.size();

cout << "current vector size: " << current_size<<endl;

Изход:current vector size: 0

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

- Добавяне на елементи в края на vector **push_back()**

- O(1) амортизирано

Пример:

```
vector<int> iv;
iv.push_back (3);
cout << iv.capacity() << endl;
cout << iv[0];
```

Изход:1
3

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

- Функции за вмъкване на елементи: **insert()**.

Пример за използване :

```
vector<int> iv;
iv.insert (iv.end(), 3);
cout << iv.capacity() << endl;
cout << iv[0];
```

Изход:1
3

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

Добавяне:

- Член функцията **insert()** изисква два аргумента:
  1. Итератор (iterator), указващ позицията на добавянето
  2. Елементът, който се вмъква.
- Действие:
  - Елементът втори аргумент се вмъква преди специфицираната чрез двойката позиция;
  - Новият елемент е преди елемента, който специфицираният итератор сочи.

Понятието итератор е тема на следващата лекция!

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

- **pop_back** & **back** : Премахване (получаване) на последния елемент от последователността - O(1)

Пример за използване :

```
vector<int> iv;
iv.insert (iv.end(), 3);
cout << iv.capacity() << endl;
cout << iv[0];
int vLastElement=iv.back()
iv.pop_back();
cout << iv.size() << endl; // 0
```

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

- Промяна на капацитета **reserve**. Използва се при наличие на информация за необходима памет за елементи - O(1) до O(n) :

Пример:

```
vector<int> iv;
iv.reserve(500);
cout << iv.capacity() << endl; //500
```

- Не променя размера на вектора

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

- Промяна на размера **resize**. Използва се при наличие на информация за броя необходими елементи - O(1) до O(n) :

Пример:

```
vector<int> iv;
iv.resize(10);
cout << iv.size() << endl; //10
```

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

- Проверка за празен вектор **empty** - O(1)

Връща true ако е празен.

Пример:

```
vector<int> iv;
bool bEmpty = iv.empty(); //true
```

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

- Изтриване на елемент(и) от вектора: **erase**

Параметър итератор(и) – O(n)

Пример за използване:

```
iv.erase(iv.begin(), iv.end()); // всички

iv.erase(iv.begin()+3, iv.begin()+6); // обхват

iv.erase(iv.begin()+2); // трети елемент
```

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

- Получаване на референция към елемент, по подадения параметър : **at()** – O(1). Ако индексът е невалиден - exception от класа out_of_range.

```
vector<int> iv(20);
try {
  int iValue20=iv.at(19)
  int iValBug=iv.at(iv.size()+1);  // грешен достъп
}
catch (out_of_range obj) {
  cout << obj.what() << endl;
}
```

4.1. ***Контейнерни класове:*** *Вектор (vector)* - функции

- Размяна на елементите на вектори – **swap** – O(1) (размяна на указатели)
- Пример:

```
v1.swap(v2);
```

4.1. ***Контейнерни класове:*** *Вектор (vector)* - оператори

Оператор за присвояване O(n):

operator=

Пример:

```
vector<int> v1 (3,6);
vector<int> v2 (5,0);

v2=v1; // v1,v2 size() ->3
v1=vector<int>(); // v1.size() ->0
// v2.size()  -> 3
```

- !!!Копира се съдържанието на вектора

4.1. ***Контейнерни класове:*** *Вектор (vector)* - оператори

- Оператор за индексация - O(1):

operator[]. Достъп до елемента по указан индекс. Извън размера-незащитен достъп и непредвидим резултат. Разлика с at()?

int iVal = v2[0]; // at(0) => front() Резултат?

Въпроси?