

vector(ベクタ)

- C++でautoでvectorを宣言
- auto,ベクタの型vector< int > :: iterator it
- for(auto x: nums) xはnumsの要素(ベクタの要素cout << x)
- xはベクタの要素
- for(auto &x:nums) xはnumsの要素
- &xはxのアドレス
- ベクタの要素を表示(要素をconstで宣言)
- ベクタの要素を操作する関数
- ベクタvector deque string +,-,<=,>=
- ベクタset map list multiset multimap

string

コンストラクタ

- string(); 空のコンストラクタ string str;
- string(const char *s); sの文字列
- string(const string& str); strのstring
- string(int n, char c); n個のc

初期化

```
string str1,str2;
```

代入

- str1 = "hello";
- str2 = str1;

代入操作

- str1.assign("hello");
- str1.assign("hello",3); helloの最初の3文字str1

連結操作

- str1 += "世界"; str1 = str1 + "世界"
- str1.append("世界"); str1 = str1 + "世界"
- str2.append(str1,int pos,int n); str2 = str2 + str1.substr(pos,n)

検索操作

- s.find(string str,int pos); sのpos位置から-1まで検索
- s.rfind(); 検索
- s.replace(int pos,int n,string str); sのpos位置からn文字をstr

変換操作

- ASCIIコード

- = 0
- > 1
- < -1
- `s.compare(str)` **s**`[str]`

string

- `str[1]` 第二个字符
- `s.at(i)` 第 i 个字符

字符串操作

- `s.insert(pos,str);` **s**`[pos]`**pos**`[str]`
- `s.erase(pos,n);` **s**`[pos]`**n**

子串

- `s.substr(pos,n);` **s**`[pos]`**n** **string**

方法

```
s.size(); s.copy(); s.length(); s.assign(); s.append(); s.find(); s.rfind(); s.replace(); s.compare();
s.at(); s.insert(); s.erase();
```

vector(向量)

向量是线性容器，向量的每一个元素都是同类型的。向量的实现原理和数组类似，通过 **vector** 来实现向量，**v.begin()**,**v.end()**,**v.rbegin()**,**v.rend()**。**v.begin()****.**v.end()****.end()**.rend()**

构造

- `vector< int > v;`
- `vector(v.begin(),v.end());` **v**[begin(),end())] 构造向量

成员

- `v[i]=` 向量 **v** 的第 i 个元素
- `.assign()`
- `{}{}` 向量

成员函数

- `.empty();` false → true
- `.capacity()>= .size()`
- `.resize(int num,(int elem));` num 为向量的大小，elem 为新元素

成员方法

- `.push_back();`
- `.pop_back();`
- `.insert(pos,ele);` pos 为插入位置，ele 为插入元素
- `.erase(pos);` pos 为删除位置
- `.clear();`

成员常量

- `arr[1];`
- `.front();`

- .back(); リストの最後の要素

メソッド

- リストの構造を操作する
• .swap(vec);
• リストswapリストの構造を操作する **vector< int >(v).swap(v)** **vector< int > (v) リスト**

メソッド

- .reserve(int len); リストの構造を操作する(「push_back」) リストの構造を操作する

メソッド

```
.begin(),.end(),.rbegin(),.rend() .assign(); .empty(); .capacity(); .size(); .resize(); .push_back();  
.pop_back(); .insert(); .erase(); .front(); .back(); .reserve();
```

deque(ダブルエンドキュー)

リスト構造のリストvector

stack(スタック)

リスト構造のリストvector

メソッド

```
.push(); .pop(); .top(); .size(); .empty();
```

queue(キュー)

リスト構造のリストvector

メソッド

```
.push(); .pop(); .size(); .empty(); .front(); .back();
```

list(リスト)

リスト構造のリストvector **list**リスト構造のリストvector(リスト[])

メソッド

- .assign(); リスト構造のリストvector.**assign(1,10)**
- .swap();
- リスト=リスト

メソッド

- .insert(pos,elem); pos(位置) elem(要素)
- .erase(pos); pos(位置)
- .remove(elem); elem(要素)

メソッド

- .reverse(); リスト

- `.sort();` 亂序のvectorを昇順に並び替える

例題

```
.assign(); .swap(); .push_front(); .pop_front(); .push_back(); .pop_back(); .empty(); .size(); .resize();
.insert(); .erase(); .clear(); .remove(); .reverse(); .sort();
```

set/multiset

- `set`(集合)
- `multiset` setと似た構造
- `set`と`multiset`
- `multiset`と`set`

例題

- `set = multiset`

例題

`set.insert()`

- `.erase();`
 - `.erase(pos)` 要素削除
 - `.erase(elem)` 値削除

例題

- `.find(key);` key要素の位置を返す
- `.count(key);` key要素の個数

例題

- `resize`

pair

- `set< pair< type , type > >;`
- `pair< type , type >`

例題

- `pair< type , type > p (value1,value2);`
- `pair< type , type > p = make_pair(value1,value2);`

例題

- `.first();` pairの1つ目
- `.second();` pairの2つ目

例題

例題  alt text

例題  alt text

例題

```
.insert(); .size(); .empty(); .swap(); .clear(); .erase(); .find(); .count();
```

map/multimap

- map`pair<key,value>`
- pair`key(key) value(value)`
- key`value`
- map`key(value)`
- multimap`key()`
- map`key()`

map

- `map< int , int > m;`

map

- `m[key]=value`
- `m.insert(pair< int , int >)(key,value);`

map操作

- `.insert();`
 - `m.insert(pair< int ,int >)(key,value);`
 - `m.insert(make_pair(key,value));`
 - `m[key] = 20;` **key**는 `pair<key,value>`의 `key`로 `m[key]`는 `value`로
- `.erase();`
 - `.erase(key);` `key`는 `value`
 - `.erase(pos);` `value`
- `.erase();`

map操作

- `.find();` `map`을

map

- `set`

map

map`map<key,value>`

- `s.begin()->first(second);`
- `(*s.begin()).first(second);`
- `key m[key];`

map

`.size(); .empty(); .swap(); .clear(); .insert(); .erase(); .find(); .count();`

map`()`

- `operator<<(ostream& os, const pair<key,value> &p)`
- `operator=(const pair<key,value> &p)`

map

- `operator bool`
- `operator(operator())`

- ・`operator()`の重载
- ・`operator<</operator>>`

二項演算子

`<functional>`

- ・`plus`
- ・`minus`
- ・`multiplies`

一元演算子

- ・`plus< T >;`
- ・`minus< T >;`
- ・`multiplies< T >;`
- ・`divides< T >;`
- ・`modulus< T >;`
- ・`negate< T >;` **← negate**の重载

■ `plus< int > v; // 定義 v(50,10); // 計算`

比較演算子

- ・`bool equal_to< T >;`
- ・`bool not_equal_to< T >;`
- ・`bool greater< T >;`
- ・`bool greater_equal< T >;`
- ・`bool less< T >;`
- ・`bool less_equal< T >;` **← sort**の重载

論理演算子

- ・`bool logical_and< T >;`
- ・`bool logical_or< T >;`
- ・`bool logical_not< T >;`

STL

`<algorithm> <functional> <numeric>`

アルゴリズム

- ・`for_each` **← for**
 - `for_each(beg,end,_func);`
 - `beg:開始 end:終了 _func:関数(要素を操作する)`
 - `_func`の重载
- ・`transform` **← map**
 - `transform(beg1,end1,beg2,_func)`
 - `beg1:開始 end1:終了 beg2:結果 _func:関数(要素を操作する)`
 - `_transform`の重载

範囲

- ・`find` **← find_if**
 - `find(beg,end,value)`
 - `beg:開始 end:終了 value:検索値`

- `find_if(beg,end)`
- `find_if_not(beg,end)`
- `find_if(beg,end,_prec)`
 - `beg:vector<T> end:vector<T> _prec:bool(=,!=,==,!=,!=,!=)`
- `adjacent_find(beg,end)`
 - `adjacent_find(beg,end)`
 - `find_if(beg+1,end,_prec)`
 - `beg:vector<T> end:vector<T>`
- `binary_search(beg,end)`
 - `bool(bool)`
 - `binary_search(beg,end,value)`
 - `beg:vector<T> end:vector<T>`
 - `value:vector<T>`
- `count(beg,end)`
 - `count(beg,end,value)`
 - `beg:vector<T> end:vector<T> value:vector<T>`
 - `value:vector<T> const`
- `count_if(beg,end)`
 - `count_if(beg,end,_prec)`
 - `beg:vector<T> end:vector<T> _prec:bool(=,!=,==,!=,!=,!=)`

排序

- `sort; 亂序化`
 - `vector<T>`
- `random_shuffle; 亂序化`
 - `random_shuffle(lbegin, lend, value)`
 - `beg:vector<T> end:vector<T>`
- `merge; 合并`
 - `merge(beg1,end1,beg2,end2,dest)`
 - `beg1:vector<T> end1:vector<T> beg2:vector<T> end2:vector<T> dest:vector<T>`
 - `vector<T>`
- `reverse; 反转`
 - `reverse(beg,end)`
 - `beg:vector<T> end:vector<T>`

修改

- `copy; 复制`
 - `merge(beg,end,dest)`
 - `beg:vector<T> end:vector<T> dest:vector<T>`
- `replace; 替换`
 - `replace(beg,end,oldvalue,newvalue)`
 - `beg:vector<T> end:vector<T> oldvalue:vector<T> newvalue:vector<T>`
- `replace_if;`
 - `replace_if(beg,end,_pred,newvalue)`
 - `beg:vector<T> end:vector<T> _pred:bool(=,!=,==,!=,!=,!=) newvalue:vector<T>`

- swap;
 - swap(c1,c2)
 - c1: 1 c2: 2 swap

numeric

- < numeric >
- accmulate; **累加**
 - accmulate(beg,end,value)
 - beg: 起始 end: 结束 value: 值
- fill; **填充**
 - fill(beg,end,value)
 - beg: 起始 end: 结束 value: 值

set

set_intersection

- set_intersection; **交集**
 - set_intersection(beg1,end1,beg2,end2,dest)
 - beg1: v1 begin end1: v1 end beg2: v2 begin end2: v2 end dest: 交集
 - 交集大小(即size)
 - 交集元素(即元素满足两个集合的条件)
- set_union; **并集**
 - set_union(beg1,end1,beg2,end2,dest)
 - beg1: v1 begin end1: v1 end beg2: v2 begin end2: v2 end dest: 并集
 - 并集大小(即size)
 - 并集元素(即元素属于其中一个集合)
- set_difference; **差集**
 - set_difference(beg1,end1,beg2,end2,dest)
 - beg1: v1 begin end1: v1 end beg2: v2 begin end2: v2 end dest: 差集
 - 差集大小(即size)
 - 差集元素(即元素属于v1且不属于v2)
 - 差集元素(即v1 \ v2)