**SEZNAM**

Deklaracija:

* list <pod\_tip> ime\_seznama; //kliče se default konstruktor; ustvari prazen seznam
* list <pod\_tip> ime\_seznama (st\_elementov); //določeno število elementov, neznana vrednost elementov
* list <pod\_tip> ime\_seznama (st\_elem, vrednost); //podana velikost; vsi elementi imajo isto vrednost

**primer:**

**list<int> seznam;**

**seznam.push\_back(75);**

**seznam.push\_back(11);**

**seznam.push\_front(3);**

**list<int>::iterator it;**

**cout << "seznam: ";**

**for ( it=seznam.begin(); it != seznam.end(); it++)**

**cout << \*it<<", ";**

**it--;**

**cout<<endl;**

**seznam.insert(it,10);**

**for (list<int>::iterator it=seznam.begin(); it != seznam.end(); it++)**

**cout << \*it<<", ";**

**seznam.sort(); //sortira elemente seznama**

**seznam.pop\_back(); //briše zadnji element**

**seznam.remove(3); //erase briše spozicije in uporablja iterator**

**VEKTOR**

* **razredni template,** deduje lastnosti dinamične tabele
* ima možnost spreminjanja velikosti
* ima možnost preverjanja indeksa ob naslavljanju posameznega elementa
* funkcije za kopiranje, brisanje
* **samodejna sprostitev pomnilnika ob končanju življenja**
* v osnovi je uporaba enostavna – podobna statični tabeli

**Odvisno od deklaracije, se kličejo različni konstruktorji (enako kot pri list!):**

**vector <pod\_tip> ime\_vektorja; //prazen vektor**

**vector<int> x(100); //**pazi! niso []

**vector<float> x (5, 3.2);** //inicializira se vseh 5 elementov z vrednostjo 3.2

x[10]=1; //brez preverjanja indeksa

x.at(10)=1; //s preverjanjem indeksa; če je indeks prevelik – vrže izjemo (exception)

x.resize(200); //poveča vektor na 200 (doda še 100); prvih 100 elementov ohrani vrednosti

!! če uporabimo **resize() –** alocirapomnilnik in inicializira z default vrednostjo:

1. **ALOCIRA SE NOV DEL POMNILNIKA (**za200elementov**);**
2. **SKOPIRA SE PREJŠNJI VEKTOR (TABELA)**
3. **BRIŠE SE STARA TABELA**

**Če so velike tabele in kompleksni elementi – vse skupaj je zelo počasi**

**x.reserve(1000); //**rezervira 1000 elementov; vse do 1000 se za večanje ne bo kopiralo, če bo //večje pa se kopira kot pri resize()

//zelo hitro

Metoda reserve():

1. samo alocira, NE INICIALIZIRA
2. ne spreminja velikosti vektorja – samo kapaciteto

x.push\_back(5); //doda 1 element (vrednost je 5) na konec vektorja

x.clear(); //briše in postavi size=0

empty() //**NE BRIŠE VSEBINE**; preverja, ali je vektor prazen

**while (!x.empty())**

erase() //briše z določene pozicije (uporablja iterator):

x.erase(x.begin()+5);

x.erase(x.begin(), x.begin()+5); //briše od začetka do 5. elementa

**insert()** //vstaqvi nov element pred pozicijo; obvezna uporaba iteratorja!!

**ITERATOR je objekt, ki omogoča sprehajanje poelementih kolekcije (zbirke?, kontejnerja?) in vsebuje konstruktor:**

**Iterator() //vrne nov element**

Vektorji so najboljši za vstavljanje na konec vektorja(boljši kot list (seznami) in deques(vrste))

Za vstavljanje ali brisanje drugam oz.na druga mesta kot na konec vektorji so slabši kot seznami

primer:

vector<int> x (3, 100);

vector<int> :: iterator it;

it = x.begin();

it = x.insert (it, 200); //vstavi 200 na začetek

x.insert (it,2,300);

primer2:

vector<int> x (3,100);

x.insert (x.begin(),2,200); // 2 je št.vstavljanj; 200 je vrednost

**size() //število elementov**

**capacity() //vrne velikost alociranega prostora; je >= size()**

**Primer:**

**int n=5;**

**vector<int> x;**

**for (int i=0; i<=n; i++)**

**x**.**push**\_**back**(**i);**

**cout<<x.capacity()<<endl;**

**MAP**

* je kontejner, ki vsebuje pare objektov
* vsak objekt je lahko kateregakoli tipa
* elementi NISO indeksirani od 0 naprej, indeksirani so tako kot uporabnik želi (s čimerkoli)
* to je možno zaradi preobložitve operatorja []

primer:

map<string, int> dijak;

dijak["Zan Bizjak"]=3;

dijak["amadej Groselj"]=1;

dijak["se en tam"]=1;

cout << "Zan Bizjak ima oceno " << dijak["Zan Bizjak"] << endl;

cout << "Amadej Groselj ima oceno " << dijak.at("amadej Groselj") << endl;

cout << "se en tam pa ima oceno " << dijak["se en tam"] << endl;