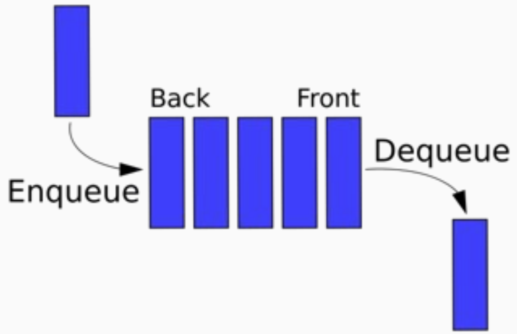
Napredno programiranje i programski jezici

06 C++ (lista, STL, izuzeci)

Fakultet tehničkih nauka, Novi Sad
23-24/Z
Dunja Vrbaški

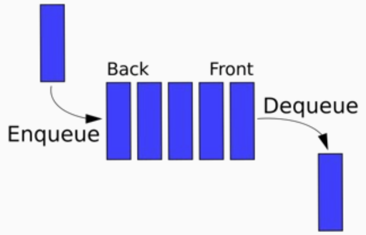
ZADATAK

Realizovati ATP queue kao generičku podklasu klase List<T>.



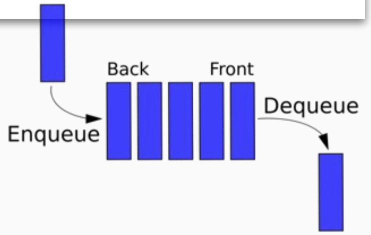
```
size()
empty()

add(int, const T&)
remove(int)
read(int, T&)
clear();
```

```
size()
empty()

addToQueue(~~int, ~~const T&)
removeFromQueue(~~int~~)
readFromQueue(~~int,~~ T&)
clear()
```



```
template<T>


size()
empty()

add(int, const T&)
remove(int)
read(int, T&)
clear()

operator<<(ostream&, const List<T>&)
```

```
template<T>
: List<T>

size()
empty()

addToQueue(~~int, ~~const T&)
removeFromQueue(~~int~~)
readFromQueue(~~int,~~ T&)
clear()

printOut(const LinkedQueue<T> &)
```



```
template <class T>
class LinkedQueue;

template <class T>
void printOut(const LinkedQueue<T> &);

template <class T>
class LinkedQueue : private List <T>
{
    ...
}
```

```
template <class T>
class LinkedQueue : private List <T>
{
public:
    LinkedQueue() {};
    bool readFromQueue(const T &retVal) const {
        return List<T>::read(1,retVal);
    }
    void removeFromQueue()
    {
        this -> remove(1);
    }
    void addToQueue(const T &El)
    {
        this -> add(size() + 1,El);
    }
    bool empty() const
    {
        return List<T>::empty();
    }
    int size() const
    {
        return List<T>::size();
    }
    friend void printOut<>(const LinkedQueue<T>&);
    virtual ~LinkedQueue() {}
};
```

```
template <class T>
class LinkedQueue : private List <T>
{
public:
    LinkedQueue() {};

    bool readFromQueue(const T &retVal) const {
        return List<T>::read(1,retVal);
    }
    void removeFromQueue()
    {
        this -> remove(1);
    }
    void addToQueue(const T &El)
    {
        this -> add(size() + 1,El);
    }
    bool empty() const
    {
        return List<T>::empty();
    }
    int size() const
    {
        return List<T>::size();
    }
    friend void printOut<>(const LinkedQueue<T>&);
    virtual ~LinkedQueue() {}
};
```

```
template <class T>
void printOut(const LinkedQueue<T> &rlq)
{
    cout << endl;
    cout << "Velicina reda: " << rlq.size() << endl;
    cout << "Sadrzaj reda je: ";
    T retVal;
    for(int i = 1; i <= rlq.size(); i++)
    {
        if(i > 1) cout << ", ";
        rlq.read(i, retVal);
        cout << retVal;
    }
    cout << endl << endl;
}
```

```
template <class T>
class LinkedQueue : private List <T>
{
    ...
};
```

```
template <class T>
void printOut(const LinkedQueue<T> &rlq)
{
    ...
}
```

```
typedef LinkedQueue<int> IntQueue;

int main(){
   IntQueue a;

   a.addToQueue(1);
   a.addToQueue(2);
   ...
   printOut(a);
   ...
   a.removeFromQueue();

   ...
   int ret;
   a.readFromQueue(ret);
   ...
```

```cpp
template <class T>
class LinkedQueue : private List <T>
{
public:
    LinkedQueue() {};

    bool readFromQueue(const T &retVal) const {
        return List<T>::read(1,retVal);
    }
    void removeFromQueue()
    {
        this -> remove(1);
    }
    void addToQueue(const T &El)
    {
        this -> add(size() + 1,El);
    }
    bool empty() const
    {
        return List<T>::empty();
    }
    int size() const
    {
        return List<T>::size();
    }
    friend void printOut<>(const LinkedQueue<T>&);
    virtual ~LinkedQueue() {}
};
```

```cpp
ostream& operator<<(ostream& out, const List<T>& rl) {
        out << endl;
        out << "--------" << endl;
        for(int i = 1; i <= rl.size(); i++){
                if(i != 1) out << ", ";
                T res;
                rl.read(i, res);
                out << res;
        }
        out << endl << "--------" << endl;
        return out;
}
```

Da li će za objekte neke konkretizacije klase LinkedQueue raditi operator <<?

```cpp
cout << queue;
```

```
template <class T>
class LinkedQueue : private List <T>
{
public:
    LinkedQueue() {};

    bool readFromQueue(const T &retVal) const {
        return List<T>::read(1,retVal);
    }
    void removeFromQueue()
    {
        this -> remove(1);
    }
    void addToQueue(const T &El)
    {
        this -> add(size() + 1,El);
    }
    bool empty() const
    {
        return List<T>::empty();
    }
    int size() const
    {
        return List<T>::size();
    }
    friend void printOut<>(const LinkedQueue<T>&);
    virtual ~LinkedQueue() {}
};
```

```
ostream& operator<<(ostream& out, const List<T>& rl) {
        out << endl;
        out << "--------" << endl;
        for(int i = 1; i <= rl.size(); i++){
                if(i != 1) out << ", ";
                T res;
                rl.read(i, res);
                out << res;
        }
        out << endl << "--------" << endl;
        return out;
}
```

Da li će za objekte neke konkretizacije klase LinkedQueue raditi operator <<?

```
cout << queue;
```

```
error: 'List<int>' is an inaccessible base of 'LinkedQueue<int>'
```

ZADATAK

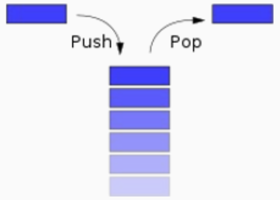
(neobavezno)

Pokušati sa implementacijom još nekih operatora za listu. Na primer:

- [ ] (čitanje i pisanje)
- += (konkatenacija)

U klasi List pokušati sa implementacijom metode `sort` (bilo koji sort algoritam).
Testirati konkretizaciju `List<int>`, `List<double>`.
Šta je problem ako se koristi konkretizacija sa klasama `Student` i `Complex`. Pokušati rešavanje.
Primeniti u zadatku sa Studentom.

Realizovati generičku klasu Stack.



C++ Standard Library

Kolekcija gotovih klasa i funkcija
Ako postoji - koristi!

RAZVOJ:

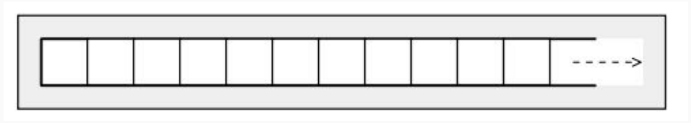
→ C++ jezik + C++ STL

naš kod, third-party biblioteke/kod

<iostream>	standardni ulazi/izlaz
<cmath>	matematika
<ctime>	date, time
<string>	stringovi
<fstream>	fajlovi
<vector>	kolekcije
<list>	
<queue>	
<stack>	
<map>	
<set>	

....

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> v1;
    vector<int> v2(10);
    ...
}
```



- dinamički niz
- kapacitet (capacity())
- realokacija + gube se reference i pokazivači
- različiti konstruktori

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    for (unsigned i = 0; i < v.size(); i++) {
        cout << v[i] << " ";
    }
    cout << endl;
}
```

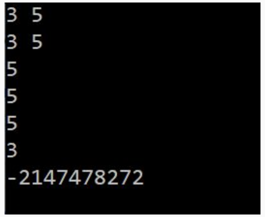
```
int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    for (unsigned i = 0; i < v.size(); i++)
        cout << v[i] << " ";
    cout << endl;

    for (int n : v)
        cout << n << " ";
    cout << endl;
}
```

```
int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    cout << v[1] << endl;
    cout << v.at(1) << endl;
    cout << v.back() << endl;
    cout << v.front() << endl;
}
```

```
int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    cout << v[1] << endl;
    cout << v.at(1) << endl;
    cout << v.back() << endl;
    cout << v.front() << endl;

    cout << v[5] << endl;
}
```

?

```
int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    cout << v[1] << endl;
    cout << v.at(1) << endl;
    cout << v.back() << endl;
    cout << v.front() << endl;

    cout << v[7] << endl;
}
```

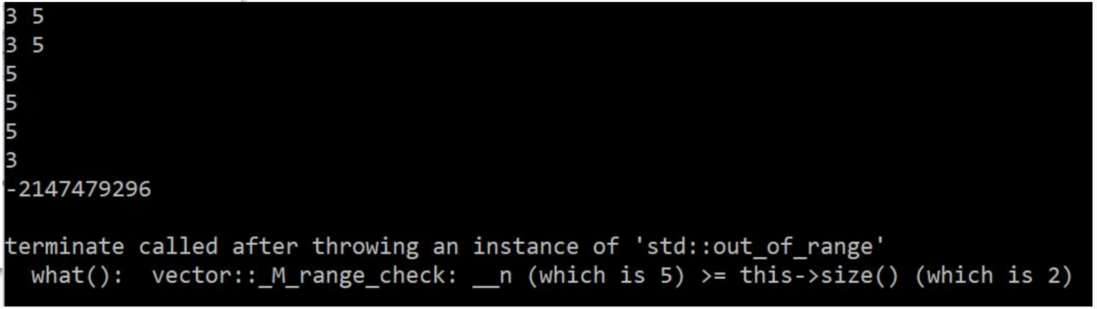


```
3 5
3 5
5
5
5
3
-2147478272
```

```
int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    cout << v[1] << endl;
    cout << v.at(1) << endl;
    cout << v.back() << endl;
    cout << v.front() << endl;

    cout << v[5] << endl;
    cout << v.at(5) << endl;
}
```

?

```
int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    cout << v[1] << endl;
    cout << v.at(1) << endl;
    cout << v.back() << endl;
    cout << v.front() << endl;

    cout << v[5] << endl;
    cout << v.at(5) << endl;
}
```



```
3 5
3 5
5
5
5
3
-2147479296

terminate called after throwing an instance of 'std::out_of_range'
  what():  vector::_M_range_check: __n (which is 5) >= this->size() (which is 2)
```

generisan je izuzetak

Izuzeci

Generišu se na određenim mestima, kad dođe do greške.
<exception>

Hijerarhija klasa, base: exception

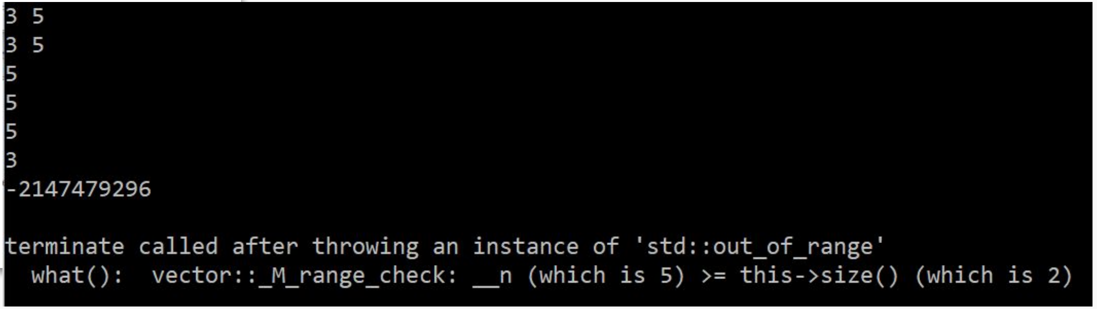
Compile-time greška vs Run-time greška

Da li mi nekako možemo da prepoznamo kad se desila run-time greška u programu?
Da prepoznamo kad je generisan izuzetak?
Da obradimo grešku i nastavimo sa izvršavanjem?

```
int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    try {
        cout << v.at(5) << endl;
    }
    catch (const exception& e) {
        cout << "Greska!" << endl;
    }
    cout << "Nastavili smo sa radom!";

}
```

```
int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    try {
        cout << v.at(5) << endl;
    }
    catch (const exception& e) {
        cout << "Greska!" << endl;
        cout << e.what();
        cout << endl;
    }
    cout << "Nastavili smo sa radom!" <<
endl;

}
```

```
int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    cout << v[1] << endl;
    cout << v.at(1) << endl;
    cout << v.back() << endl;
    cout << v.front() << endl;

    cout << v[5] << endl;
    cout << v.at(5) << endl;
}
```



```
3 5
3 5
5
5
5
3
-2147479296

terminate called after throwing an instance of 'std::out_of_range'
  what():  vector::_M_range_check: __n (which is 5) >= this->size() (which is 2)
```

Usput:
Obratiti pažnju: [] - ne generiše izuzetak, ne proverava opseg

```
int fun(const vector<int>& v) {
    return v.at(0);
}

int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    cout << fun(v) << endl;
}
```

?

```
int fun(const vector<int>& v) {
    return v.at(0);
}

int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    cout << fun(v) << endl;
    v.clear();
    cout << fun(v) << endl;
}
```

?

```
int fun(const vector<int>& v) {
    if (v.size() > 0)
        return v.at(0);
    throw runtime_error("Moj izuzetak - vektor je prazan");
}

int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    cout << fun(v) << endl;
    v.clear();

    try{
       cout << fun(v) << endl;
    }
    catch (runtime_error e){
        cout << e.what() << endl;
    }
}
```

?

```
int fun(const vector<int>& v) {
    if (v.size() > 0)
        return v.at(0);
    throw 101;
}

int main()
{
    vector<int> v1;
    vector<int> v2(10);

    v.push_back(3);
    v.push_back(5);

    cout << fun(v) << endl;
    v.clear();

    try{
       cout << fun(v) << endl;
    }
    catch (int e){
        cout << e << endl;
    }
}
```

Konstruktori, seteri

Greška je možda nastala duboko u steku poziva funkcija ili metoda.
Sa izuzecima smo u stanju da tu informaciju prosledimo.
U nekoj tački onda možemo obraditi izuzetak.

Ne treba izbegavati, ne treba previše koristiti.
Praksa, iskustvo, preporuke.

C++ & DS

biblioteke za druge PJ
ekstenzije
big data obrada
delovi pipeline

implementacija ML algoritama

ZADATAK

(neobavezno)

Pogledati SL, posebno STL

Pogledati razliku između lvalue, rvalue (c++ rvalue lvalue, c++ &&)
Pogledati move ctor (c++ move ctor)
Pogledati moderan način za rad sa pokazivačima (SL) (modern c++ pointers, c++ smart pointers)

Pogledati šta su: iterators, assertions, lambda izrazi….i sve ostalo na šta naiđete

Pogledati: https://isocpp.org/ (core guidelines, super faq).

Pogledati neko predavanje sa cppcon (YT).

Pogledati na kraju udzbenika zadatak za pripremu.