lab 4

SOURCE

ex 1

```
#include <iostream>
#include <vector>
using namespace std;

template <typename T>
void display_vec(const vector<T>& c){
  for (int i : c) {
    cout << i << " ";
  }
  cout << endl;
}

int main() {
  vector<int> v;
  for (size_t i=0; i<10; ++i){
    v.push_back(i);
  }
  std::cout<<"original vector\n";
  display_vec(v);
  v.push_back(10);
  std::cout<<"added element with .pushback(10)\n";
  display_vec(v);
  v.erase(v.begin()+3,v.begin()+8);
  std::cout<<"removed element 3-7\n";
  display_vec(v);
  v.pop_back();
  std::cout<<"poped last element\n";
  display_vec(v);
  v.insert(v.begin(),102);
  v.push_back(110011);
  std::cout<<"added 102 and 110011 to the vector\n";
  display_vec(v);
}
```

```
output:
original vector
0 1 2 3 4 5 6 7 8 9
added element with .pushback(10)
0 1 2 3 4 5 6 7 8 9 10
removed element 3-7
0 1 2 8 9 10
poped last element
```

```
0 1 2 8 9
added 102 and 110011 to the vector
102 0 1 2 8 9 110011
```

ex 2

```
#include <iostream>
#include <queue>

bool push_limited_queue(std::queue<int>& q, int limit, int value)
{
    if(q.size() >= limit)
    {
        return true;
    }
    q.push(value);
    return false;
}

void display(std::queue<int>& q)
{
    int size = q.size();
    for(int i = 0; i < size; i++)
    {
        std::cout<<q.front()<<" ";
        q.pop();
    }
    std::cout<<"\n";
}

void pop_n_elements(std::queue<int>& q, int n)
{
    if(q.size() < n) { std::cout<<"queue is too small"; return; }
    for(int i = 0; i < n; i++)
    {
        q.pop();
    }
    std::cout<<"ereased "<<n<<" elements\n";
}

void populate_queue(std::queue<int>& q,int first_element, int step, int
last_element)
{
    int i = first_element;
    for(; i <= last_element; i+=step)
    {
        q.push(i);
    }
    std::cout<<"populated queue with "<< (i-first_element)/step << " elements\n";
}
```

```
void populate_queue(std::queue<int>& q, int first_element, int step, int
last_element, int limit)
{
    bool limit_reached = false;
    int i = first_element;
    for(; i <= last_element; i+=step)
    {
        limit_reached = push_limited_queue(q, limit, i);
        if(limit_reached)
        {
            std::cout<<"queue limit exeeded by "<< (i - first_element)/step
                << " elements\n" << "all element over the limit will be erased\n";
            return;
        }
    }
    std::cout<<"populated queue with "<< (i-first_element)/step << " elements\n";
}

int main()
{
    std::queue<int> q;
    populate_queue(q, 1, 1, 12);
    display(q);
    std::string s = q.empty() ? "queue is empty!\n" : "queue isn't empty";
    std::cout<< s ;
    populate_queue(q, 1, 5, 55);
    pop_n_elements(q, 5);
    std::cout<<"length of the queue "<<q.size()<<" \n";
    display(q);
    populate_queue(q, 1, 1, 4, 3);
    display(q);
}
```

```
output:
populated queue with 12 elements
1 2 3 4 5 6 7 8 9 10 11 12
queue is empty!
populated queue with 11 elements
ereased 5 elements
length of the queue 6
26 31 36 41 46 51
queue limit exeeded by 3 elements
all element over the limit will be erased
1 2 3
```

ex 3

```
#include <iostream>
#include <queue>
```

```cpp
using namespace std;

class Applicant {
public:
    static int applicants_count;
    int applicant_num = 0;
    int time;
};

int Applicant::applicants_count = 0;

void new_applicant(std::queue<Applicant>& q, Applicant* a) //1
{
    q.push(*a);
    a->applicants_count++;
    a->applicant_num = a->applicants_count;
}

bool queue_limit(std::queue<Applicant>& q, int limit)//3
{
    if(q.size() >= limit+1) { return true; }
    else { return false; }
}

void new_applicant(std::queue<Applicant>& q, Applicant* a, int limit) //3
{
    if(!queue_limit(q, limit))
    {
        q.push(*a);
        a->applicants_count++;
        a->applicant_num = a->applicants_count;
    }
    else
    {
        std::cout<<"the queue exeeded size limit of "<<limit<<" applicants\n";
    }
}

void serve_applicant(std::queue<Applicant>& q) //2
{
    if(q.size() > 1)
    {
        q.pop();
        cout << "applicant served\n"
            << "applicants waiting for service:  " << q.size() - 1 << endl;
    }
    else { std::cout<< "all applicants are served!";}
}

int main() {
    queue<Applicant> eit;
    int limit = 3;
    Applicant p, p1, p2, p3, p4;
    new_applicant(eit, &p, limit);
    Applicant temp = eit.back();

    cout << "last applicant number: " << temp.applicant_num
        << ", queue size: " << eit.size() << endl;

    cout << "applicants waiting for service:  " << eit.size() - 1 << endl;

    new_applicant(eit, &p1, limit);

        cout << "last applicant number: " << eit.back().applicant_num
        << ", queue size: " << eit.size() << endl;

    cout << "applicants waiting for service:  " << eit.size() - 1 << endl;
    new_applicant(eit, &p2, limit);
    new_applicant(eit, &p3, limit);
    new_applicant(eit, &p4, limit);
    serve_applicant(eit);
    new_applicant(eit, &p4, limit);
}
```

```
output:
last applicant number: 0, queue size: 1
applicants waiting for service:  0
last applicant number: 0, queue size: 2
applicants waiting for service:  1
the queue exeeded size limit of 3 applicants
applicant served
applicants waiting for service:  2
```