

სააუდიტორიო სამუშაო:

- ამოცანა 1 `/[first, last)` დიაპაზონის ობიექტების ბეჭდვა ეკრანზე/
- ამოცანა 2 `/[first, last)` დიაპაზონში მოცემული ობიექტის ძებნა/ [>>>](#)
- ამოცანა 3 `/[first, last)` დიაპაზონში გარკვეული თვისების მქონე ობიექტის ძებნა/ [>>>](#)
- სავარჯიშოები [>>>](#)

ამოცანა 1 `/[first, last)` დიაპაზონის ობიექტების ბეჭდვა ეკრანზე/. მოიყვანეთ ფუნქციის კოდი, რომელიც იტერატორების `[first, last)` ფრაგმენტში (დიაპაზონში, `range`) მოთავსებულ ობიექტებს დაბეჭდავს ეკრანზე. მოიყვანეთ ფუნქციის გამოძახების რამდენიმე ვარიანტი სხვადასხვა დიაპაზონისთვის, რომლებიც სხვადასხვა ტიპის ობიექტებითაა შევსებული.

ამოხსნა: იტერატორი, ისევე როგორც პოინტერი, გვიჩვენებს ობიექტის მდებარეობას კონტეინერში. ამიტომ, პოინტერის მსგავსად იტერატორსაც აქვს ირიბი მნიშვნელობა, რომლისთვისაც იგივე (განმისამართების) ოპერატორი - "*" გამოიყენება. განსხვავება არის ++ ოპერატორის მოქმედების პრინციპში, რომელიც სხვადასხვა კლასის იტერატორისთვის სხვადასხვანაირად არის იმპლემენტირებული.

ჩვენს ფუნქციას და პროგრამა-დრაივერს შესაძლოა ჰქონდეს სახე:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <list>
#include <iterator>
#include <string>
using namespace std;

template<typename InputIterator>
void printRange(InputIterator first, InputIterator last)
{
    while (first != last)
    {
        std::cout << *first << endl;
        ++first;
    }
}

int main()
{
    vector<int> v = {11, 24, 541};
    printRange(v.begin(), v.end());
    cout << endl;
    list<double> lst = {11.88, 21.4, 541.9};
    printRange(lst.begin(), lst.end());
    cout << endl;
    ifstream ifs{ "strings.txt" };
    istream_iterator<string> i(ifs), e;
    printRange(i, e);
}
```

11
24
541
11.88
21.4
541.9
erti
ori
sami

თუ ფაილში წერია „erti ori sami“, მაშინ პროგრამის შესრულების შედეგი ჩანს მარჯვენა მხარეს.

აქ, პროგრამისთვის მნიშვნელობა არა აქვს თუ `template<typename InputIterator>` განაცხადში რას დავარქმევთ ტიპს. მნიშვნელობა აქვს პროგრამისთვის რომელიც ამ ფუნქციის გამოყენებას დააპირებს, რომ იცოდეს რომელი იტერატორი გადააწოდოს ფუნქციას და რომელი არა. ჩვენ ავირჩიეთ `InputIterator`, რადგან ამას გვკარნახობს ფუნქციის ტანი:

```

while(first != last)
{
    std::cout << *first << endl;
    ++first;
}

```

როგორც ვხედავთ, იტერატორს მოეთხოვება რომ განსაზღვრული იყოს != და * ოპერატორები (++ არაფერს გვეუბნება, იგი ნებისმიერ იტერატორს მოეთხოვება). ეს ორი ოპერატორი InputIterator-ებს განსაზღვრავს.

მთავარი ფუნქციის

```

istream_iterator<string> i(ifs), e;
printRange(i, e);

```

სტრიქონები გვიჩვენებს, რომ ეს ფუნქცია მუშაობს არა მხოლოდ კონტეინრებთან, არამედ შემავალ ნაკადთანაც.

<<< ამოცანა 2 /`[first, last)` დიაპაზონში მოცემული ობიექტის ძებნა/. მოიყვანეთ ფუნქციის კოდი, რომელიც დაადგენს არის თუ არა იტერატორების `[first, last)` დიაპაზონში მოცემული ობიექტი. დადებითი პასუხის შემთხვევაში ალგორითმმა უნდა დააბრუნოს პირველივე (დასაწყისიდან) ასეთი ელემენტის იტერატორი, წინააღმდეგ შემთხვევაში კი `last`. მოიყვანეთ ფუნქციის გამოძახების რამდენიმე განსხვავებული დიაპაზონისთვის.

ამოხსნა: ფუნქციას და პროგრამა-დრაივერს შესაძლოა ჰქონდეს სახე:

```

#include <iostream>
#include <fstream>
#include <vector>
#include <list>
using namespace std;
template<typename InputIterator, typename T>
InputIterator myFind(InputIterator first, InputIterator last, T b)
{
    while (first != last && *first != b)
        ++first;
    return first;
}
int main()
{
    vector<int> v = { 11, 24, 541 };
    vector<int>::iterator it;
    it = myFind(v.begin(), v.end(), 5421);
    if (it != v.end())
        cout << "vector - number found! " << *it << endl;
    else
        cout << "vector - number not found!" << endl;

    list<double> lst = { 11.88, 21.4, 541.9 };
    auto itLst = myFind(lst.begin(), lst.end(), 541.9);
    if (itLst != lst.end())
        cout << "list - number found! " << *itLst << endl;
    else
        cout << "list - not found!" << endl;
}

```

<<< ამოცანა 3 /`[first, last)` დიაპაზონში გარკვეული თვისების მქონე ობიექტის ძებნა/. მოიყვანეთ ფუნქციის კოდი, რომელიც დაადგენს არის თუ არა კონტეინერის `[first, last)` დიაპაზონში (range) მოცემული თვისების მქონე ელემენტი. დადებითი პასუხის შემთხვევაში ალგორითმმა უნდა დააბრუნოს პირველივე (დასაწყისიდან) ასეთი ელემენტის იტერატორი, წინააღმდეგ შემთხვევაში კი `last`. მოიყვანეთ ფუნქციის გამოძახების რამდენიმე ვარიანტი სხვადასხვა ტიპის დიაპაზონისთვის. `p` თვისება შესაძლოა იყოს მომხმარებლის მიერ შექმნილი ბულის ტიპის უნარული (ანუ ერთადგილიანი) ფუნქცია, ფუნქციის ადაპტერი ან ლამბდა.

ამოხსნა: ერთი შესაძლო ვარიანტი ასეთია:

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

template<typename InputIterator, typename Predicate>
InputIterator my_find_if(InputIterator first, InputIterator last, Predicate pred)
{
    while (first != last && !pred(*first))
        ++first;
    return first;
}

int main()
{
    vector<int> v = { 11, 24, 541 };
    auto it = my_find_if(v.begin(), v.end(), not1(bind2nd(greater<int>(), 11)));
    if (it != v.end())
        cout << "vector - needed number found! " << *it << endl;
    else
        cout << "not found!" << endl;
}
```

<<< სავარჯიშოები

1. შექმენით ფუნქცია, რომელიც დაბეჭდავს $[first, last)$ დიაპაზონის ელემენტებს m სვეტად. იტერატორის ტიპის შერჩევითვის ისარგებლეთ ლექციაში მოყვანილი მასალით.
2. შექმენით ფუნქცია, რომელიც m სვეტად დაბეჭდავს $[first, last)$ დიაპაზონიდან ისეთ ელემენტებს, რომლებიც აკმაყოფილებენ გარკვეულ თვისებას.
3. დაწერეთ ფუნქცია, რომელიც მოძებნის $[first, last)$ დიაპაზონის მქონე ელემენტებს შორის მაქსიმალურს და დააბრუნებს მის იტერატორს.
4. დაწერეთ ფუნქცია, რომელიც მოძებნის $[first, last)$ დიაპაზონის გარკვეული თვისების მქონე ელემენტებს შორის მაქსიმალურს და დააბრუნებს მის იტერატორს. თუ ასეთი არ არის, ფუნქციამ უნდა დააბრუნოს იტერატორი $last$.
5. (*) დაწერეთ ფუნქცია სახელით `extremal`, რომელიც პირველ ორ არგუმენტად მიიღებს $first, last$ იტერატორებს, $[first, last)$ დიაპაზონის ელემენტებს შორის მოძებნის ექსტრემალურ ელემენტს და დააბრუნებს მის ინდექსს. ექსტრემალური ელემენტი უნდა მოიძებნოს მესამე არგუმენტად გადაწოდებული ბინარული პრედიკატის აზრით.
6. გაარჩიეთ შემდეგი კოდი. რას აკეთებს ფუნქცია f და რა შემთხვევა რჩება მას დაუმუშავებელი:

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

template<typename InIt>
auto f
(
    InIt first,
    InIt last
)
{
    auto v = *first;
    ++first;
```

```

while (first != last)
{
    v += *first;
    ++first;
}
return v;
}
int main()
{
    vector<int> a{ 1,2,3,4,5 };
    auto sum = f(a.begin(), a.end());
    cout << sum << endl;

    vector<string> b{"We ", "are ", "from ", "TSU " };
    auto conc = f(b.begin(), b.end());
    cout << conc << endl;
}

```

7. დაწერეთ ფუნქცია, რომელიც რომელიც არგუმენტად გადაცემულ ერთ-ერთ პარამეტრში დააგროვებს [first, last) დიაპაზონის
- ელემენტების ჯამს;
 - კვადრატების ჯამს;
 - შებრუნებული ელემენტების ჯამს (ვიგულისხმობთ რომ არანულოვანებია).