Politechnika Świętokrzyska

Laboratorium

Programowanie w języku C++ 2

Ćwiczenie 2

Kontenery deque i list
Iteratory

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie studentów z biblioteką STL języka C++.

dr inż Robert Kazała

Kontener deque

Metody kontenera deque

Modifiers:

assign	Assign container content (public member function)
push_back	Add element at the end (public member function)
push_front	Insert element at beginning (public member function)
pop_back	Delete last element (public member function)
pop_front	Delete first element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_front C++11	Construct and insert element at beginning (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)

Allocator:

get_allocator	Get allocator (public member function)

fx Non-member functions overloads

relational operators	Relational operators for deque (function)
swap	Exchanges the contents of two deque containers (function template)

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)

Przykład 1

deque constructors – size constructors

```
#include <deque>
#include <iostream>

using namespace std;

int main()
{
    deque<int> d1(10, 0);
    cout<<"Size: "<<d1.size()<<endl;
    for(unsigned i = 0; i < d1.size(); ++i)
    {
        cout<< d1[i]<<" ";
    }
    cout<<endl;
    return 0;
```

```
}
```

Przykład 2

deque – iterator constructors

```
#include <deque>
#include <iostream>

using namespace std;

int main()
{
    int a1[]={1,2,3,4,5,6,7,8,9,10};
    //first one
    deque <int>d1(a1, a1+10);
    cout<<"Size (d1):  "<<d1.size()<<endl;
    for(unsigned i = 0; i < d1.size(); ++i)
    {
        cout<< d1[i]<<" ";
    }
    cout<<endl;
    //second one;
    deque <int>d2(a1+5,a1+10);
    cout<<"Size (d2):  "<<d2.size()<<endl;
    for(unsigned i = 0; i < d2.size(); ++i)
    {
        cout<< d2[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

Przykład 3

Konstruktor iteracyjny – inicjowanie inną kolekcją

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

using namespace std;

int main()
{
    //vector
    vector <int>v(10, 0);
    for(unsigned i = 0; i < v.size(); ++i)
    {
        v[i]=i+1;
    }
```

```
    cout<<"Size (v):  "<<v.size()<<endl;
    for(unsigned i = 0; i < v.size(); ++i)
    {
        cout<< v[i]<<" ";
    }
    cout<<endl;
    //deque
    deque <int>d(v.begin(), v.begin()+5);
    cout<<"Size (d):  "<<d.size()<<endl;
    for(unsigned i = 0; i < d.size(); ++i)
    {
        cout<< d[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

Przykład 4

deque – konstruktor kopiujący

```
#include <deque>
#include <iostream>

using namespace std;

int main()
{
    int a1[]={1,2,3,4,5,6,7,8,9,10};
    //first one
    deque <int> d1(a1, a1+10);
    cout<<"Size (d1):  "<<d1.size()<<endl;
    for(unsigned i = 0; i < d1.size(); ++i)
    {
        cout<< d1[i]<<" ";
    }
    cout<<endl;
    //second one;
    deque <int> d2(d1);
    cout<<"Size (d2):  "<<d2.size()<<endl;
    for(unsigned i = 0; i < d2.size(); ++i)
    {
        cout<< d2[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

Kontener list

Metody kontenera list

(constructor)	Construct list (public member function)
(destructor)	List destructor (public member function)
operator=	Assign content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

empty	Test whether container is empty (public member function)
size	Return size (public member function)
max_size	Return maximum size (public member function)

Element access:

front	Access first element (public member function)
back	Access last element (public member function)

Modifiers:

assign	Assign new content to container (public member function)
emplace_front C++11	Construct and insert element at beginning (public member function)
push_front	Insert element at beginning (public member function)
pop_front	Delete first element (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
emplace C++11	Construct and insert element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
resize	Change size (public member function)
clear	Clear content (public member function)

Operations:

splice	Transfer elements from list to list (public member function)
remove	Remove elements with specific value (public member function)
remove_if	Remove elements fulfilling condition (public member function template)
unique	Remove duplicate values (public member function)
merge	Merge sorted lists (public member function)
sort	Sort elements in container (public member function)
reverse	Reverse the order of elements (public member function)

Observers:

get_allocator	Get allocator (public member function)

fx Non-member function overloads

relational operators (list)	Relational operators for list (function)
swap (list)	Exchanges the contents of two lists (function template)

Przykład 5

Tworzenie i wykorzystanie list

```
#include <algorithm>
#include <iostream>
#include <list>

int main()
{
    // Create a list containing integers
    std::list<int> l = { 7, 5, 16, 8 };

    // Add an integer to the front of the list
    l.push_front(25);
    // Add an integer to the back of the list
    l.push_back(13);

    // Insert an integer before 16 by searching
    auto it = std::find(l.begin(), l.end(), 16);
    if (it != l.end()) {
        l.insert(it, 42);
    }

    // Iterate and print values of the list
    for (int n : l) {
        std::cout << n << '\n';
    }
}
```

Iteratory

Przykład 6

Kontenery i iteratory

```
#include <list>
#include <vector>
#include <deque>
#include <iostream>

using namespace std;

int main()
```

```
{
    //containers
    vector<int> v;
    deque<int> d;
    list<int> l;
    //iterators
    vector<int> ::iterator it1;
    vector<int> ::const_iterator it2;
    vector<int> ::reverse_iterator it3;
    vector<int> ::const_reverse_iterator it4;

    deque<int> ::iterator it5;
    deque<int> ::const_iterator it6;
    deque<int> ::reverse_iterator it7;
    deque<int> ::const_reverse_iterator it8;

    list<int> ::iterator it9;
    list<int> ::const_iterator it10;
    list<int> ::reverse_iterator it11;
    list<int> ::const_reverse_iterator it12;

    return 0;
}
```

Przykład 7

Przykłady użycia iteratorów – normal iterators

```
#include <list>
#include <vector>
#include <deque>
#include <iostream>

using namespace std;

int main()
{
    //containers
    vector <int> v(10);
    deque <int> d(10);
    list <int> l(10);

    int i = 1;
    //vector
    vector<int>::iterator itV;
    for(itV = v.begin()  ; itV != v.end(); ++itV,++i)
    {
        *itV = i;
    }
    for(itV = v.begin();  itV != v.end(); ++itV)
    {
        cout << *itV << " ";
    }
    cout<<endl;
```

```
    //deque
    deque<int>::iterator itD = d.begin();
    for(itD = d.begin()  ; itD != d.end(); ++itD,++i)
    {
        *itD = i;
    }
    for( itD = d.begin() ; itD != d.end(); ++itD)
    {
        cout << *itD << " ";
    }
    cout<<endl;

    list<int>::iterator itL = l.begin();
    for( ; itL != l.end(); ++itL,++i)
    {
        *itL = i;
    }
    for( itL = l.begin() ; itL != l.end(); ++itL)
    {
        cout << *itL << " ";
    }
    cout<<endl;
    return 0;
}
```

Przykład 8

Przykłady użycia iteratorów – reverse iterators

```
#include <list>
#include <vector>
#include <deque>
#include <iostream>

using namespace std;

int main()
{
    //containers
    vector <int> v(10);
    deque <int> d(10);
    list <int> l(10);

    int i = 1;
    //vector
    vector<int>::iterator itV;
    for(itV = v.begin()  ; itV != v.end(); ++itV,++i)
    {
        *itV = i;
    }

    for(vector<int>::reverse_iterator it = v.rbegin();  it != v.rend(); +
+it)
    {
        cout << *it << " ";
```

```
    }
    cout<<endl;
    //deque
    i = 1;
    deque<int>::iterator itD = d.begin();
    for(itD = d.begin()  ; itD != d.end(); ++itD,++i)
    {
        *itD = i;
    }
    for( deque<int>::reverse_iterator it = d.rbegin() ; it != d.rend(); +
+it)
    {
        cout << *it << " ";
    }
    cout<<endl;
//list
    i = 1;
    list<int>::iterator itL = l.begin();
    for( ; itL != l.end(); ++itL,++i)
    {
        *itL = i;
    }
    for(list<int>::reverse_iterator it = l.rbegin() ; it != l.rend(); +
+it)
    {
        cout << *it << " ";
    }
    cout<<endl;
    return 0;
}
```

Przykład 9

Przykłady użycia iteratorów – const iterators

```
#include <list>
#include <vector>
#include <deque>
#include <iostream>

using namespace std;

int main()
{
    int a[] = {1,2,3,4,5,6,7,8,9,10};
    //containers
    vector <int> v(a,a+10);
    deque <int> d(a,a+10);
    list <int> l(a,a+10);

    //vector
    for(vector<int>::const_iterator it = v.begin()  ; it != v.end(); +
+it)
    {
        cout << *it << " ";
    }
```

```
    cout<<endl;
    //deque
    for(deque<int>::const_iterator it = d.begin(); it != d.end(); ++it)
    {
        cout << *it << " ";
    }
    cout<<endl;
//list
    for(list<int>::const_iterator it = l.begin(); it != l.end(); ++it)
    {
        cout << *it << " ";
    }
    cout<<endl;
    return 0;
}
```

Przykład 10

Inicjalizacja iteratorów – niepoprawne użycie

```
#include <list>
#include <vector>
#include <deque>
#include <iostream>

using namespace std;

int main()
{
    int a[] = {1,2,3,4,5,6,7,8,9,10};
    //containers
    vector<int>  v(a,a+10);
    deque<int>  d(a,a+10);
    list<int>  l(a,a+10);

    vector<int> ::const_iterator it1 = v.begin();
    *it1 = *it1+1;
    deque<int> ::const_iterator it2 = d.begin();
    *it2 = *it2+1;
    list<int> ::const_iterator it3 = l.begin();
    *it3 = *it3+1;
    return 0;
}
```

Literatura

Zadania

1. Przeanalizować działanie, zmodyfikować, uruchomić i opisać kod wszystkich przykładów z instrukcji.
2. Przeanalizować i omówić działanie poniższego programu, rozbudować go o testowanie deque i list.

```
#include <list>
#include <vector>
#include <deque>
#include <iostream>

using namespace std;

int main()
{
    int a[] = {11,22,33,44,55,66,77,88,99,100};
    //containers
    vector<int>  v(a,a+10);
    deque<int>  d(a,a+10);
    list<int>  l(a,a+10);

    cout << "test 1" << endl;

    vector<int> ::iterator it1 = v.begin();
    deque<int> ::iterator it2 = d.begin();
    list<int> ::iterator it3 = l.begin();


    cout << *it1 << endl;
    cout << *it1+1 << endl;
    cout << *it1+2 << endl;
    cout << v[3] << endl;
    cout << v[4] << endl;


    cout << "test 2" << endl;

    it1 = v.begin();

    cout << *it1 << endl;
    cout << ++*it1 << endl;
    cout << ++*it1 << endl;
    cout << *it1++ << endl;
    cout << *it1 << endl;
    cout << v[6] << endl;
    cout << v[7] << endl;

    cout << "test 3" << endl;

    auto itr = v.begin();

    cout << *itr<<endl;
    advance(itr, 1);
    cout << *itr<<endl;
    advance(itr, 2);
    cout << *itr<<endl;
    cout << *next(itr)<<endl;
    cout << *next(itr, 3)<<endl;

    cout << "test 4" << endl;

    cout << *next(v.begin(), 1) << endl;
    cout << *next(v.begin(), 2) << endl;
```

```
    return 0;
}
```

3. Na własnych, praktycznych przykładach zaprezentować działanie iteratorów z różnymi kontenerami.