Politechnika Świętokrzyska

Laboratorium

Programowanie w języku C++ 2

Ćwiczenie 3

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)

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 1

Pomiar czasu z wykorzystaniem <ctime>

```
#include <iostream>
#include <ctime>
#include <cmath>
using namespace std;

int main ()
{
	float x,y;
	clock_t time_req;

	// Using pow function
	time_req = clock();
	for(int i=0; i<100000; i++)
	{
		y = log(pow(i,5));
	}
	time_req = clock() - time_req;
	cout << "Using pow function, it took " <<
(float)time_req/CLOCKS_PER_SEC << " seconds" << endl;

	// Without pow function
	time_req = clock();
	for(int i=0; i<100000; i++)
	{
		y = log(i*i*i*i*i);
	}
	time_req = clock()- time_req;
	cout << "Without using pow function, it took " <<
(float)time_req/CLOCKS_PER_SEC << " seconds" << endl;

	return 0;
}
```

Przykład 2

Pomiar czasu z wykorzystaniem <chrono>

```
#include <stdio.h>
#include <chrono>

int main () {
    double sum = 0;
    double add = 1;

    // Start measuring time
    auto begin = std::chrono::high_resolution_clock::now();

    int iterations = 1000*1000*100;
    for (int i=0; i<iterations; i++) {
        sum += add;
        add /= 2.0;
    }

    // Stop measuring time and calculate the elapsed time
    auto end = std::chrono::high_resolution_clock::now();
    auto elapsed =
std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin);

    printf("Result: %.20f\n", sum);

    printf("Time measured: %.3f seconds.\n", elapsed.count() * 1e-9);

    return 0;
}
```

Przykład 3

Generowanie liczb losowych

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main()
{
      srand((int)time(0)); // inicjacja generatora
      int i = 0;
      while(i++ < 10) {
            int r = (rand() % 100) + 1;
            cout << r << " ";
      }
      return 0;
}
```

Przykład 4

Generowanie wektora liczb losowych

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

int main()
{
      srand((int)time(0)); // inicjacja generatora
      vector <int> v1;
      int i = 0;
      while(i++ < 20) {
            v1.push_back((rand() % 100) + 1);
      }

     for (int x : v1)
          cout << x << " ";
      cout << endl;

      return 0;
}
```

Przykład 5

Zamiana list

```
// swap lists
#include <iostream>
#include <list>

int main ()
{
  std::list<int> first (3,100);   // three ints with a value of 100
  std::list<int> second (5,200);  // five ints with a value of 200

  first.swap(second);

  std::cout << "first contains:";
  for (std::list<int>::iterator it=first.begin(); it!=first.end(); it++)
    std::cout << ' ' << *it;
  std::cout << '\n';

  std::cout << "second contains:";
  for (std::list<int>::iterator it=second.begin(); it!=second.end(); it+
+)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Przykład 6

Sortowanie z wykorzystaniem algorytmu sort

```
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector
#include <deque>       // std::vector


int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  std::deque<int> myvector1 (myints, myints+8);
  std::deque<int> myvector2 (myints, myints+8);

  std::sort (myvector1.begin(), myvector1.end());

  std::sort (myvector2.begin(), myvector2.begin()+4);

  std::cout << "vector 1 after sort:";
  for (std::deque<int>::iterator it=myvector1.begin(); it!
=myvector1.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  std::cout << "vector 2 after sort:";
  for (std::deque<int>::iterator it=myvector2.begin(); it!
=myvector2.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Przykład 7

Wstawianie elementów do listy.

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

int main ()
{
  std::list<int> mylist;
  std::list<int>::iterator it;

  // set some initial values:
  for (int i=1; i<=5; ++i) mylist.push_back(i);

  it = mylist.begin();
  ++it;

  mylist.insert (it,10);
```

```
  mylist.insert (it,2,20);

  --it;

  std::vector<int> myvector (2,30);
  mylist.insert (it,myvector.begin(),myvector.end());

   std::cout << "mylist contains:";
  for (it=mylist.begin(); it!=mylist.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Przykład 7

Wyszukiwanie elementów z wykorzystaniem `std::find`.

```
#include <iostream>     // std::cout
#include <algorithm>    // std::find
#include <vector>       // std::vector

int main () {
  // using std::find with array and pointer:
  int myints[] = { 10, 20, 30, 40 };
  int * p;

  p = std::find (myints, myints+4, 30);
  if (p != myints+4)
    std::cout << "Element found in myints: " << *p << '\n';
  else
    std::cout << "Element not found in myints\n";

  // using std::find with vector and iterator:
  std::vector<int> myvector (myints,myints+4);
  std::vector<int>::iterator it;

  it = find (myvector.begin(), myvector.end(), 30);
  if (it != myvector.end())
    std::cout << "Element found in myvector: " << *it << '\n';
  else
    std::cout << "Element not found in myvector\n";

  return 0;
}
```

Literatura

Zadania

1. Przeanalizować, uruchomić, zmodyfikować i opisać kod wszystkich przykładów z instrukcji.
2. W przykładzie 6 zamiast kontenera vector wykorzystać kontener deque i list.

3. Utworzyć bardzo dużą tablicę liczb losowych, następnie zainicjować obiekt typu vector, deque i typu lista wartościami utworzonej wcześniej tablicy.
   a) Zmierzyć czas sortowania listy z wykorzystaniem metody sort().
   b) Zmierzyć czas sortowania deque z wykorzystaniem metody sort().
   c) Zmierzyć czas sortowania vectora z wykorzystaniem algorytmu sort biblioteki STL.
   d) Zmierzyć sumaryczny czas kopiowania nieposortowanej listy do vectora, sortowania vectora z wykorzystaniem algorytmu sort biblioteki STL i kopiowania vectora z powrotem do listy.
4. Zmierzyć czas dodawania na końcu lub początku dużej liczby elementów do vectora, deque i listy.
5. Zmierzyć czas dodawania do dużego kontenera typu vector i list dużej liczby elementów z wykorzystaniem metody insert.
6. Zbadać efektywność wyszukiwania dla różnych kontenerów.
7. Utworzyć książkę adresową z wykorzystaniem kontenera vector zawierającego struktury z danymi adresowymi. Utworzyć funkcje do wyszukiwania, sortowania i wstawiania nowych pozycji w zadanym miejscu i do wstawiania na końcu.