

LAB_12 Wprowadzenie w STL

Na przykładzie klasy kontenera `vector` rozważymy kilka podstawowych zadań.

1. Wprowadźmy klasę danych – współrzędne punktu na płaszczyźnie

```
class coord
{
    double *cr;          // coordinates of a point in the plane cr [0] = x;
                        //cr [1] = y
public:
    coord() : cr(NULL) {}
    coord(double x, double y);
    ~coord() { if(cr) delete [] cr;}
    void set(double x, double y);
    double * get() { return cr;}
private:
    void alloc();
};

void coord::alloc()
{
    try
    {
        cr = new double [2];
        memset((void *)cr, 0, 2*sizeof(double));
    }
    catch(bad_alloc xx)
    {
    }
}

coord::coord(double x, double y)
{
    alloc();
    cr[0] = x;
    cr[1] = y;
}

void coord::set(double x, double y)
{
    cr[0] = x;
    cr[1] = y;
}
```

2. Stwórzmy dynamiczną tablicę jako `vector` STL:

```
int main()
{
    //numb - number of points onto plaine
    size_t numb = 20, ind, fract, it;
    const double PI = 3.14159236;
    double x, y, ro, fi, dfi = 2.0 * PI / numb;
    coord tmp(0, 0);    // Auxiliary object
    vector<coord> vec;   // dynamic array for the coord data type
    vector<coord>::iterator It_end; //introduce the iterator to the
                                // container class vector , datatype - coord
    srand(static_cast<unsigned int>(time(NULL)));

    //preparation of an array of objects of coord type presenting a points
    //with the polar coordinates {ro, fi}.
```

//Then, we calculate cartesian coordinates and push this point to vector vec.

```
for (it = 0; it < numb; it++)
{
    //radius is randomly calculated
    ind = rand() % 5;
    fract = rand() % 10;
    fi = dfi * it;
    ro = ind + 0.1 * fract + 1.0;
    x = ro * cos(fi);
    y = ro * sin(fi);
    // create a coord type object with x, y coordinates
    tmp.set(x, y);
    // put at the end of the vec array
    vec.push_back(tmp);
}
// we display the set of points on the monitor with the disp
//template function
disp(vec, "initial vec");

// sort array of coord
// in descending order (vector length)
// ob.ro[i] > ob.ro[i+1]
// algorithm sort uses a predicate-functions SortDescending
sort(vec.begin(), vec.end(), SortDescending);
disp(vec, "sorted in descending order vec");

// deletion of vectors with the same length
//algorithm unique uses a predicate-functions CoordCompare
It_end = unique(vec.begin(), vec.end(), CoordCompare);
cout << " ind = " << It_end - vec.begin() << endl;
disp(vec, "after unique vec");

// truncate vec after unique call
vec.erase(It_end, vec.end());
disp(vec, "after erase vec");

vec.clear();
system("pause");
return 0;
}
```

3. Funkcja - szablon disp()

```
template<class T>
void disp(vector<T> &v, char *str)
{
    typename vector<T>::iterator It;
    cout << str << endl;
    //view the dynamic array using the iterator
    for(It=v.begin(); It!=v.end(); It++)
    {
        cout << *(It); //get the value of the object using the iterator
    }
    cout << "-----\n";
}
```

4. Do wyżej wymienionego zadania należy przygotować klasę danych coord. **W przeciwnym razie wystąpi Page Fault podczas wykonywania programu.** Do poprawnego działania algorytmu sortowania oraz algorytmu unique należy wprowadzić funkcję SortDescending-predicate i funkcję porównania CoordCompare (patrz MSDN).

```
bool SortDescending(coord elem1, coord elem2)
```

```
// the function-predicate SortDescending is a binary predicate
// returns bool and has two arguments for sorting in descending order by
// vector // length. SortDescending should return true if the length of ro1 of
// element1 is
// greater than the length of ro2 of element2, where element1 and element2 are
// two consecutive elements of a dynamic array represented by container-class
// vector
{
    double ro1 = elem1, ro2 = elem2;
    // The expression double ro1 = elem1 makes sense only when we introduce a
    //function that converts the operator double () {return (the length of
    the //vector)}
    return ro1 > ro2;
}
```

Tu w klasie danych będzie potrzebna funkcja konwertująca – zadeklarować jako składową public:

```
operator double();
```

Jej implementacja wygląda tak:

```
coord::operator double ()
{
    double ro = cr[0]*cr[0]+cr[1]*cr[1];
    return sqrt(ro);
}
```

Porównywanie == zmiennoprzecinkowych typów danych może powodować problemy. Ten kod jest nieprawidłowy:

```
bool CoordCompare(coord elem1, coord elem2)
{
    double ro1 = elem1, ro2 = elem2;
    return(ro1 == ro2);
}
```

Z powodu błędów zaokrągleń niektóre elementy tablicy mogą być traktowane jako !=. Na przykład, 2.000000000000 i 2.0000000000001. To jest poprawny wariant kodu:

```
bool CoordCompare(coord elem1, coord elem2)
{
    double ro1 = elem1, ro2 = elem2;
    double dif = fabs(ro1-ro2);
    if(dif < 1.0e-8)
        return true;
    else
        return false;
}
```

Spróbuj użyć pierwszego wariantu kodu i przeanalizuj wyniki. Później zmień na poprawny wariant kodu. Funkcja predykatu SortDescending i funkcja porównania CoordCompare nie są metodami klasy coord.

Aby kompilator był w stanie rozpoznać prototypy kontenerów i algorytmów STL (std::sort, std::unique) (patrz MSDN), musisz dodać nagłówki

```
#include <vector>
#include <algorithm>
```

Aby rozpoznać prototypy funkcji SRT sqrt, fabs, sin, cos – dodaj nagłówek
`#include <cmath>`

Zadanie 2.

Proszę przystosować następującą klasę do funkcji main:

```
class MY_STUDENT
{
    int year;    //birthyear
    char* name;  //name of student
public:

};

void main1()
{
    vector < MY_STUDENT> vec;
    MY_STUDENT tmp(...); //use parameterized constructor

    for (int i = 0; i < 10; i++)
    {
        //randomly generate a name of student with differernt number of symbols
        tmp.setname(.....);
        tmp.setyear(...);
        vec.push_back(tmp);
    }

    //output vec to screen

    //insert a new student (tmp) with 3-th element of vector using method insert.

    //output vec to screen

    //sort vec in ascending order using algorithm sort.

    //output vec to screen

    //remove elements of vector in the range vec[3] - vec[5] using method erase.

    //output vec to screen

    vec.clear();
}
```

Przygotuj funkcję print(...) do wyprowadzenia wektora na ekran i przekaz do niej wektor vec.

Jako przykład wzorcowy patrz przykłady do wykładu W44, W45, W46, W47.