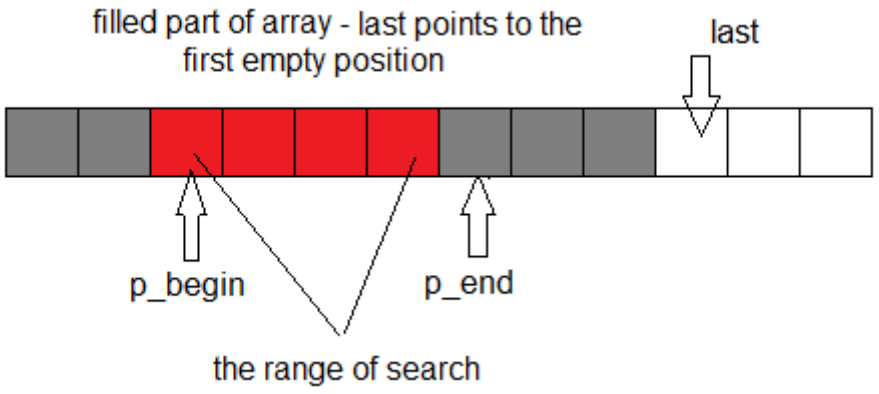
Lab 6

1. Utwórz funkcję-szablon *my_add*, która wykonuje c = a + b, gdzie a, b, c są typami ogólnymi T. Ta funkcja powinna działać dla każdego typu danych (nie wolno zmieniać żadnej linii kodu w funkcji-szablonie zastępując jakikolwiek rzeczywisty typ danych). Dlatego każda klasa danych pracująca z funkcją-szablonem powinna mieć przeciążenie operatorów +, = oraz konstruktora kopiującego. Poniższy kod testuje funkcję my_add dla typów danych double i NODE_COORD. Klasa NODE_COORD została opracowana w ramach implementacji Lab 4.

```
int _tmain(int argc, _TCHAR* argv[])
{

	//	type NODE_COORD
	NODE_COORD ob1(2, 3), ob2(3,4), ob;
	ob1.disp("ob1");
	ob2.disp("ob2");
	ob = my_add(ob1, ob2);
	ob.disp("ob");

	//	type int
	int a = 2, b = 3, c;
	c = my_add(a, b);

	system("pause");
	return 0;
}
```

2. Utwórz funkcję-szablon, która znajduje wymagany element tablicy ogólnego typu T. Prototyp tej funkcji odpowiada funkcji poszukiwania *find* z biblioteki STL. *p_begin* – wskaźnik do elementu tablicy, z którego rozpoczyna się wyszukiwanie, *p_end* – wskaźnik do pierwszego elementu tablicy, który znajduje się poza zasięgiem wyszukiwania, *key* – obiekt typu K, który jest używany jako klucz do wyszukiwania. To jest ten obiekt, który należy znaleźć. Może to być obiekt typu T (K == T), lub może to być obiekt innego typu - zależy to od kryteriów wyszukiwania. Funkcja my_find zawiera instrukcję t == k, gdzie t jest bieżącym elementem tablicy typu T, a k ma typ K - należy przeciążyć operator == dla klasy T, aby można było porównać element tablicy typu T z kluczem wyszukiwania typu K.

Element k może wystąpić kilka razy w tablicy typu T lub nie wystąpić a ni razu. W tym drugim przypadku *my_find* musi zwrócić NULL. Pierwsze wyszukiwanie: ustawiamy *p_begin* do początku tablicy. Jeśli wynik nie jest NULL, poszukiwany element jest znaleziony. Musimy sprawdzić, czy ten element nie występuje ponownie - powtórz wyszukiwanie, ustawiając *p_begin* do pierwszego elementu poza odnalezionym elementem z poprzedniego wyszukiwania – patrz przykład *main*.



```
template <class T, class K>
T * my_find(const T *p_begin, const T *p_end, const K &key)
/*=======================================================
Find element of array T *
Start from p_begin - points to the element to be started
          p_end   - points to the first element to right from search interval
          key - key of search
There is nesessary to overload operator == for class T according with type of
K.
1. First search: p_begin = &array[0]
2. Second search: p_begin = &array[previous search+1]
=======================================================*/

int _tmain(int argc, _TCHAR* argv[])
{

	//		type double
	int ndim = 10;
	int it;
	size_t dist;
	double *p_tab;  //declare an array of double type
	double *p_tmp, search_item;

	try
	{
		p_tab = new double[ndim];
	}
	catch(bad_alloc aa)
	{
		cout << "mem_alloc_err\n";
		system("pause");
		exit(1);
	}

	for(it=0; it<ndim-1; it++)
	{
		p_tab[it] = (double)(it+1);
	}
	search_item = p_tab[ndim-1] = 3;

	cout << "how many " << search_item << " is met in p_tab?\n";
	cout << "which elements contain search_item?\n";

	p_tmp = p_tab;    //set p_tmp to the begin of array
	while(p_tmp)
	{
		p_tmp = my_find(p_tmp, p_tab+ndim, search_item);
		if(p_tmp)
		{
			dist = p_tmp-p_tab;
			cout << "tab [ " << dist << " ] = " << *p_tmp << endl;
			p_tmp++;
		}
		else
			cout << "End of search\n";
	}
	delete [] p_tab;

	//		type node_coord
	NODE_COORD tab_coord[] = {
		NODE_COORD(0, -1),
		NODE_COORD(3.0000000000001, 0),
		NODE_COORD(2, -1),
		NODE_COORD(0, 5)
	};
```

```
        double search_key = 3;
        // Find all vertices that contain at least one coordinate equal to 3.0
        //write your code, please!
        system("pause");
        return 0;
}
```

Możesz skorzystać się przykładem W22 do wykładów jako wzorcem.