Частичная специализация функций(?) и классов. Контейнеры STL. Итераторы. Библиотека <algorithm>

Мстоян Амиран

Разбор домашнего задания

Частичная специализация шаблона

```
1  template <class T, int size> // size является non-type параметром шаблона
2  class StaticArray
3  {
4  private:
5      // Параметр size отвечает за длину массива
6      T m_array[size];
7
8  public:
9      T* getArray() { return m_array; }
10
11     T& operator[](int index)
12     {
13         return m_array[index];
14     }
15 };
```

```
template <typename T, int size>
void print(StaticArray<T, size> &array)
{
    for (int count = 0; count < size; ++count)
        std::cout << array[count] << ' ';
}
```

Проблема:

```
1  int main()
2  {
3      // Объявляем массив типа char
4      StaticArray<char, 14> char14;
5
6      strcpy_s(char14.getArray(), 14, "Hello, world!");
7
8      // Выводим элементы массива
9      print(char14);
10
11     return 0;
12 }
```

```
H e l l o ,   w o r l d !
```

Подход №1

```
// Шаблон функции print() с полной специализацией шаблона класса StaticArray для работы с типом char и длиной массива 14
template <>
void print(StaticArray<char, 14> &array)
{
    for (int count = 0; count < 14; ++count)
        std::cout << array[count];
}

int main()
{
    // Объявляем массив типа char
    StaticArray<char, 14> char14;

    strcpy_s(char14.getArray(), 14, "Hello, world!");

    // Выводим элементы массива
    print(char14);

    return 0;
}
```

Проблема подхода:

```
int main()
{
    // Объявляем массив типа char
    StaticArray<char, 12> char12;

    strcpy_s(char12.getArray(), 12, "Hello, dad!");

    // Выводим элементы массива
    print(char12);

    return 0;
}
```

Подход №2:

```
1 // Шаблон функции print() с частично специализированным шаблоном класса StaticArray<char, size> в качестве параметра
2 template <int size> // size по-прежнему является non-type параметром
3 void print(StaticArray<char, size> &array) // мы здесь явно указываем тип char
4 {
5     for (int count = 0; count < size; ++count)
6         std::cout << array[count];
7 }
```

Проблема частичной специализации методов

```
1  template <class T, int size> // size является non-type параметром шаблона
2  class StaticArray
3  {
4  private:
5      // Параметр size отвечает за длину массива
6      T m_array[size];
7
8  public:
9      T* getArray() { return m_array; }
10
11     T& operator[](int index)
12     {
13         return m_array[index];
14     }
15
16     void print()
17     {
18         for (int i = 0; i < size; i++)
19             std::cout << m_array[i] << ' ';
20         std::cout << "\n";
21     }
22 };
```

Решение?

```
1 // Не сработает
2 template <int size>
3 void StaticArray<double, size>::print()
4 {
5     for (int i = 0; i < size; i++)
6         std::cout << std::scientific << m_array[i] << " ";
7     std::cout << "\n";
8 }
```

Решение??

```
template <int size> // size является non-type параметром шаблона
class StaticArray<double, size>
{
private:
    // Параметр size отвечает за длину массива
    double m_array[size];

public:
    double* getArray() { return m_array; }

    double& operator[](int index)
    {
        return m_array[index];
    }
    void print()
    {
        for (int i = 0; i < size; i++)
            std::cout << std::scientific << m_array[i] << ' ';
        std::cout << "\n";
    }
};
```

Последовательные контейнеры

- Последовательные контейнеры поддерживают указанный пользователем порядок вставляемых элементов.

Последовательные контейнеры

- vector
- array
- deque(double ended queue)
- list
- forward_list

Ассоциативные контейнеры

- В ассоциативных контейнерах элементы вставляются в заранее определенном порядке — , например, как отсортировано по возрастанию. Также доступны неупорядоченные ассоциативные контейнеры. Ассоциативные контейнеры можно объединить в два подмножества: сопоставления (set) и наборы (map).

Ассоциативные контейнеры

- map
- set

Контейнеры-адаптеры

- queue
- priority_queue
- stack

<algorithm>

Non-modifying sequence operations:

all_of	Test condition on all elements in range (function template)
any_of	Test if any element in range fulfills condition (function template)
none_of	Test if no elements fulfill condition (function template)
for_each	Apply function to range (function template)
find	Find value in range (function template)
find_if	Find element in range (function template)
find_if_not	Find element in range (negative condition) (function template)
find_end	Find last subsequence in range (function template)
find_first_of	Find element from set in range (function template)
adjacent_find	Find equal adjacent elements in range (function template)
count	Count appearances of value in range (function template)
count_if	Return number of elements in range satisfying condition (function template)
mismatch	Return first position where two ranges differ (function template)
equal	Test whether the elements in two ranges are equal (function template)
is_permutation	Test whether range is permutation of another (function template)
search	Search range for subsequence (function template)
search_n	Search range for elements (function template)

copy	Copy range of elements (function template)
copy_n	Copy elements (function template)
copy_if	Copy certain elements of range (function template)
copy_backward	Copy range of elements backward (function template)
move	Move range of elements (function template)
move_backward	Move range of elements backward (function template)
swap	Exchange values of two objects (function template)
swap_ranges	Exchange values of two ranges (function template)
iter_swap	Exchange values of objects pointed to by two iterators (function template)
transform	Transform range (function template)
replace	Replace value in range (function template)
replace_if	Replace values in range (function template)
replace_copy	Copy range replacing value (function template)
replace_copy_if	Copy range replacing value (function template)
fill	Fill range with value (function template)
fill_n	Fill sequence with value (function template)
generate	Generate values for range with function (function template)
generate_n	Generate values for sequence with function (function template)
remove	Remove value from range (function template)
remove_if	Remove elements from range (function template)
remove_copy	Copy range removing value (function template)
remove_copy_if	Copy range removing values (function template)
unique	Remove consecutive duplicates in range (function template)

min	Return the smallest (function template)
max	Return the largest (function template)
minmax	Return smallest and largest elements (function template)
min_element	Return smallest element in range (function template)
max_element	Return largest element in range (function template)
minmax_element	Return smallest and largest elements in range (function template)