Шаблонные классы. Специализация функций и классов.

Мстоян Амиран

Разбор домашнего задания

Явная специализация шаблона функции

```
template <class T>
class Repository
{
private:
    T m_value;
public:
    Repository(T value)
    {
        m_value = value;
    }

    ~Repository()
    {
    }

    void print()
    {
        std::cout << m_value << '\n';
    }
};
```

```
int main()
{
    // Инициализируем объекты класса
    Repository<int> nValue(7);
    Repository<double> dValue(8.4);

    // Выводим значения объектов класса
    nValue.print();
    dValue.print();
}
```

Пример утечки памяти

```
1  int main()
2  {
3      // Динамически выделяем временную строку
4      char *string = new char[40];
5
6      // Просим пользователя ввести свое имя
7      std::cout << "Enter your name: ";
8      std::cin >> string;
9
10     // Сохраняем то, что ввел пользователь
11     Repository<char*> repository(string);
12
13     // Удаляем временную строку
14     delete[] string;
15
16     // Пытаемся вывести то, что ввел пользователь
17     repository.print(); // получаем мусор
18 }
```

Явная специализация шаблона класса

```
template <class T>
class Repository8
{
private:
    T m_array[8];

public:
    void set(int index, const T &value)
    {
        m_array[index] = value;
    }

    const T& get(int index)
    {
        return m_array[index];
    }
};
```

```cpp
template <>
class Repository8<bool> // специализируем шаблон класса Repository8 для работы с типом bool
{
// Реализация класса
private:
    unsigned char m_data;

public:
    Repository8() : m_data(0)
    {
    }

    void set(int index, bool value)
    {
        // Выбираем оперируемый бит
        unsigned char mask = 1 << index;

        if (value) // если на входе у нас true, то бит нужно "включить"
            m_data |= mask; // используем побитовое ИЛИ, чтобы "включить" бит
        else // если на входе у нас false, то бит нужно "выключить"
            m_data &= ~mask; // используем побитовое И, чтобы "выключить" бит
    }

    bool get(int index)
    {
        // Выбираем бит
        unsigned char mask = 1 << index;
        // Используем побитовое И для получения значения бита, а затем выполняется его неявное преобразование в тип bool
        return (m_data & mask) != 0;
    }
};
```

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";
    }
};
```