Семинар #3: Инициализация. Домашнее задание.

Задача 1. Виды инициализации

Определите, каким из видов инициализации (default, value, direct или copy) была инициализирована переменная в приведённых ниже строках.

1. `int a = 10;`
2. `int a{};`
3. `int a{10};`
4. `int a;`
5. `int a = {10};`
6. `int a(10);`

Для того чтобы сдать эту задачу, нужно создать файл в формате `.txt` и, используя любой текстовый редактор, записать в него ответы в следующем формате (ответы ниже неверны):

```
1) Default
2) Direct
3) Copy
```

После этого файл нужно поместить в ваш репозиторий на github.

Задача 2. Печать методов

Рассмотрим следующий класс:

```
class Cat
{
    int x;
public:
    explicit Cat(int x) : x(x)   {std::cout << "Constructor from int\n";}
    Cat() : x(0)                 {std::cout << "Default Constructor\n";}
    Cat(const Cat& c) : x(c.x)   {std::cout << "Copy Constructor\n";}
    Cat& operator=(const Cat& c) {x = c.x; std::cout << "Assignment\n"; return *this;}
    ~Cat()                       {std::cout << "Destructor\n";}
};
```

Определите, скомпилируется ли следующий код, использующий этот класс, и, если скомпилируется, то что будет напечатано в следующих программах:

1. ```
   int main()
   {
       Cat a;
   }
   ```
2. ```
   int main()
   {
       Cat a = 10;
   }
   ```
3. ```
   int main()
   {
       Cat a{10};
   }
   ```
   ```

4. ```
   int main()
   {
       Cat a{};
       Cat b{a};
   }
   ```

5. ```
   int main()
   {
       Cat a;
       Cat b = a;
   }
   ```

6. ```
   int main()
   {
       Cat a;
       Cat b;
       b = a;
   }
   ```

7. ```
   void func(Cat a) {}
   int main()
   {
       Cat b;
       func(b);
   }
   ```

8. ```
   void func(Cat& a) {}
   int main()
   {
       Cat b;
       func(b);
   }
   ```

9. ```
   void func(Cat a) {}
   int main()
   {
       func(10);
   }
   ```

10. ```
    void func(Cat a) {}
    int main()
    {
        func(Cat{10});
    }
    ```

11. ```
    class Dog
    {
        Cat y;
    public:
        Dog(const Cat& a) : y(a) {}
    };

    int main()
    {
        Cat a;
        Dog b(a);
    }
    ```

12.
```
class Dog
{
    Cat y;
public:
    Dog(const Cat& a) {y = a;}
};

int main()
{
    Cat a;
    Dog b(a);
}
```

13.
```
int main()
{
    Cat* p = new Cat;
    delete p;
}
```

14.
```
#include <cstdlib>
int main()
{
    Cat* p = (Cat*)std::malloc(3 * sizeof(Cat));
    std::free(p);
}
```

15.
```
int main()
{
    Cat* p = new Cat[3];
    delete[] p;
}
```

16.
```
#include <vector>
int main()
{
    std::vector<Cat> v(3);
}
```

17.
```
struct Dog
{
    static Cat y;
};
Cat Dog::y{};

int main()
{
    Dog a;
    Dog b;
}
```

Для того, чтобы сдать эту задачу, нужно создать файл в формате `.txt` и, используя любой текстовый редактор, записать в него ответы в следующем формате (ответы ниже неверны):

```
1) Copy Constructor
   Destructor
2) Constructor from int
   Assignment
   Destructor
3) Error
```

После этого, файл нужно поместить в ваш репозиторий на github.

Задача 3. Наличие особых методов

- Класс является *default constructible*, если у него есть доступный конструктор по умолчанию, который можно использовать для создания объекта.
- Класс является *copy constructible*, если у него есть доступный конструктор копирования, который можно использовать для создания копии объекта.
- Класс является *copy assignable*, если у него есть доступный оператор копирующего присваивания, который можно использовать для присваивания объектов.

Определите, являются ли следующие классы default constructible, copy constructible и copy assignable.

1. ```
   struct Alice
   {
       Alice() {}
       Alice(const Alice& other) {}
       Alice& operator=(const Alice& other) {return *this;}
   };
   ```

2. ```
   struct Bob
   {
       Bob(int x) {}
   };
   ```

3. ```
   struct Casper
   {
       Casper() = default;
       Casper(const Casper& other) = delete;
       Casper& operator=(const Casper& other) = default;
   };
   ```

4. ```
   struct Diana {};
   ```

Для того, чтобы сдать эту задачу нужно создать файл в формате `.txt` и, используя любой текстовый редактор, записать в него ответы в следующем формате (ответы ниже неверны):

```
1) Default Constructible, Copy Constructible
2) Default Constructible
...
```

После этого, файл нужно поместить в ваш репозиторий на github.

Задача 4. `new`

Используйте операторы `new` или `new[]`, чтобы создать в куче и сразу инициализировать следующие объекты:

- Один объект типа `int`, равный `123`.
- Один объект типа `std::string`, равный `"Cats and Dogs"`.
- Массив объектов типа `int`, содержащий элементы `10, 20, 30, 40, 50`.
- Один объект типа `std::vector`, содержащий элементы `10, 20, 30, 40, 50`.
- Массив объектов типа `std::string`, равный `{"Cat", "Dog", "Mouse"}`.

Все эти объекты обязательно должны быть созданы в куче, а не на стеке. Напечатайте все созданные объекты на экран. Удалите все созданные объекты с помощью операторов `delete` и `delete[]`.

Задача 5. Создание строки

Создайте объекты класса `std::string` следующими способами:

- Создайте объект класса `std::string` на стеке обычным образом и инициализируйте его строкой `"Cat"`.
- Создайте объект класса `std::string` в куче, используя оператор `new`, и инициализируйте его строкой `"Dog"`. Напечатайте этот объект на экран. Удалите этот объект с помощью оператора `delete`.
- Пусть у нас есть массив:

  ```
  char x[sizeof(std::string)];
  ```

  Создайте объект класса `std::string` в этом массиве с помощью оператора `placement new` и инициализируйте объект строкой `"Elephant"`. Напечатайте этот объект на экран. Корректно уничтожьте этот объект.

Задача 6. Обёртка

Напишите класс `RestrictWrapper`, который бы оборачивал переменную типа `int`. Этот класс должен иметь как минимум следующие поля:

1. Ссылку на переменную типа `int`.
2. Константу `minValue`.
3. Константу `maxValue`.

и методы:

1. Конструктор, который задаёт поля.
2. Метод `get`, который возвращает значение оборачиваемой переменной.
3. Метод `set`, который задаёт значение оборачиваемой переменной, если значение находится в диапазоне от `minValue` до `maxValue`. Если же в `set` передаётся значение меньше, чем `minValue`, то переменная должна установиться в значение `minValue`. Если в `set` передаётся значение больше, чем `maxValue`, то переменная должна установиться в значение `maxValue`.
4. Метод `getCount` который возвращает количество вызовов метода `get` этого объекта.
5. Метод `setCount` который возвращает количество вызовов метода `set` этого объекта.

```
#include <iostream>
// Тут нужно написать класс RestrictWrapper

int main()
{
    int x = 100;
    RestrictWrapper w(x, 0, 300);
    std::cout << w.get() << std::endl;          // Напечатает 100
    w.set(-200);
    std::cout << w.get() << std::endl;          // Напечатает 0
    w.set(300);
    w.set(400);
    w.set(500);
    std::cout << w.get() << std::endl;          // Напечатает 300

    std::cout << w.getCount() << std::endl; // Напечатает 3
    std::cout << w.setCount() << std::endl; // Напечатает 4
}
```

Задача 7. Сумма из строки

Напишите функцию, которая принимает на вход строку в следующем формате: `"[num1, num2, ... numN]"`. Функция должна возвращать целое число типа `int` – сумму всех чисел в квадратных скобках. В случае если на вход приходит некорректная строка, то функция должна бросать исключение `std::invalid_argument`. Протестируйте эту функцию в `main`, обработав бросаемое исключение.

аргумент	возвращаемое значение
`"[10, 20, 30, 40, 50]"`	`150`
`"[4, 8, 15, 16, 23, 42]"`	`108`
`"[20]"`	`20`
`"[]"`	`0`

Необязательные задачи

Задача 1. `StringView`

Создайте свой класс `mipt::StringView`, аналог класса `std::string_view` для класса `mipt::String`. Этот класс должен содержать 2 поля указатель `mpData` (тип `const char*`) и размер `mSize` (тип `size_t`). Класс `mipt::String` можно найти в файле `miptstring.hpp`.

Методы, которые нужно реализовать:

- Конструктор по умолчанию. Должен устанавливать указатель в `nullptr`, а размер в 0.
- Конструктор копирования.
- Конструктор от `mipt::String`.
- Конструктор от `const char*`
- Перегруженный `operator[]`
- Метод `at`, аналог `operator[]`, но если индекс выходит за границы, то данный метод должен бросать исключение `std::out_of_range`
- Перегруженный `operator<`
- Перегруженный `operator<<` с объектом `std::ostream`.
- Метод `size`.
- Метод `substr`, должен возвращать объект типа `std::string_view`.
- Методы `remove_prefix` и `remove_suffix`.

Также придётся изменить класс `mipt::String`. Нужно будет добавить конструктор от `mipt::StringView`. Протестируйте этот класс.

Задача 2. Создание `mipt::String`

В файле `miptstring.hpp` содержится класс `mipt::String`. Создайте объекты этого класса в следующим образом:

- Создайте объект класса `mipt::String` на стеке обычным образом и инициализируйте его строкой `Cat`.
- Создайте объект класса `mipt::String` в куче, используя оператор `new`, и инициализируйте его строкой `"Dog"`. Напечатайте этот объект на экран. Удалите этот объект, с помощью оператора `delete`.
- Пусть у нас есть массив:

  ```
  char x[sizeof(mipt::String)];
  ```

  Создайте объект класса `mipt::String` в этом массиве с помощью оператора `placement new` и инициализируйте объект строкой `"Elephant"`. Напечатайте этот объект на экран. Удалите этот объект.

Задача 3. Ссылка как поле

Есть некоторое количество групп кошек, которые соревнуются в том, кто поймает больше мышек. Соревнование командное, побеждает та группа, которая суммарно поймала больше мышек.

Ваша задача - написать класс `Cat`, который оказался бы полезным для автоматизации подсчёта пойманных мышек. Сумарное количество пойманых мышек для каждой группы будет хранится вне класса `Cat`, например в локальной переменной (так как кошки из других групп не должны иметь доступ к этой переменной). В самом же классе должна храниться *ссылка* на эту переменную. Хоть задачу можно решить и с помощью указателя, но в данной задаче обязательно использовать ссылку. В классе вам нужно написать следующие методы:

- Конструктор. Должен конструироваться от переменной, в которой будет храниться суммарное количество мышек.

- Метод `catchMice` - поймать мышек. Принимает число и увеличивает суммарно количество пойманых мышек данной группы на это число.

Пример использования такого класса (код ниже должен работать и с вашим классом):

```
int miceCaughtA = 0;
int miceCaughtB = 0;

Cat alice(miceCaughtA), alex(miceCaughtA), anna(miceCaughtA);
Cat bob(miceCaughtB), bella(miceCaughtB);

alice.catchMice(2);
alex.catchMice(1);
bella.catchMice(4);
bob.catchMice(2);
anna.catchMice(1);
bella.catchMice(1);
alex.catchMice(4);
bella.catchMice(5);
alice.catchMice(2);

cout << miceCaughtA << endl; // Должно напечатать 10
cout << miceCaughtB << endl; // Должно напечатать 12
```

Задача 4. Времена из строки

- Напишите простой класс для работы со временем:

```
class Time
{
private:
    int mHours, mMinutes, mSeconds;
public:
    Time(int hours, int minutes, int seconds);
    Time(const std::string& s); // строка в формате "hh:mm:ss"
    Time operator+(Time b) const;
    int hours() const; int minutes() const; int seconds() const;
    friend std::operator<<(std::ostream& out, Time t);
};
```

- Напишите функцию `std::vector<Time> getTimesFromString(const std::string& s)`, которая будет принимать строку в формате `"hh:mm:ss ... hh:mm:ss"`, где за место букв должны стоять некоторые числа. Например, строка может иметь вид `"11:20:05 05:45:30 22:10:45"`. Функция должна возвращать вектор времен, соответствующий временам в строке.
- Напишите функцию `Time sumTimes(const std::vector<Time>& v)`, которая будет суммировать все времена и возвращать эту сумму. Для суммирования времён используйте перегруженный оператор `+` класса `Time`.

Использовать функции можно следующим образом:

```
std::vector<Time> v = getTimesFromString("11:20:05 05:45:30 22:10:45");
v.push_back(Time("01:10:30"));
Time s = sumTimes(v);
std::cout << s << std::endl;
```

В результате исполнения этого участка кода на экран должно напечататься `16:26:50`.