C++: Классы. Знакомство со строками

23 мая 2017 г.

Повторение

1. Что такое инкапсуляция?
2. Что такое полиморфизм?
3. Что такое наследование?
4. Какие существуют модификаторы доступа?
5. В чем разница между структурой и классом?

Конструкторы

1. Зачем нужен конструктор?
2. Какие существуют виды конструкторов?
3. Какая проблема есть в следующем коде?

```
class MyClass
{
public:
    MyClass(int a);
    MyClass(int a, int b = 1);
};
```

Инициализация (1)

Какой недостаток есть в следующем коде?

```
class MyClass
{
private:
    int a;
    const int c;
    int b;
public:
    MyClass() : c(10), b(2), a(1) { }
};
```

Инициализация (2)

Скомпилируется ли следующий код?

```
class MyClass
{
private:
    int a = 1;
    int b = 10;
public:
    MyClass() { }
    ~MyClass() { }
};
```

Инициализация (3)

Скомпилируется ли следующий код?

```
class MyClass
{
private:
    int a = 1;
    int b{ a * 2 };
public:
    MyClass() { }
    ~MyClass() { }
};
```

Инициализация (4)

Скомпилируется ли следующий код?

```
class MyClass
{
private:
    int a{ b * 2 };
    int b = 10;
public:
    MyClass() { }
    ~MyClass() { }
};
```

Инициализация (5)

Что будет выведено в консоль?

```
struct A {
        int b = 1;
        int c{ b + 1 };
        int d;
        int a = d++;
        A(int value): d(value) { }
};
int main() {
        A m{3};
        std::cout << m.a << " " << m.b << " " << m.c << " " << m.d;
};
```

Инициализация (6)

Что будет выведено в консоль?

```
struct A {
        int b = 1;
        int c{ b + 1 };
        int d;
        int a = d++;
        A(int value): d(value), a(5) { }
};
int main() {
        A m{3};
        std::cout << m.a << " " << m.b << " " << m.c << " " << m.d;
};
```

Инициализация (7)

1) инициализация в листе инициализации в конструкторе имеет более высокий приоритет, чем инициализация в определении класса

2) значение из инициализации в определении класса может быть проигнорировано

Деструкторы

1. Зачем нужен деструктор?
2. Какая ошибка допущена в следующем коде?

```
class MyClass
{
private:
    int* p;

public:
    MyClass(int size) { p = new int[size]; }
    ~MyClass() { delete p; }
};
```

Встроенные (inline) методы

1) в точку вызова подставляется всё тело метода => не генерируется код вызова
=> растёт скорость исполнения программы

2) все методы, тело которых находится в определении класса, являются встроенными,

НО:

3) подходит, если тело функции не очень большое

Встроенные (inline) методы

4) чтобы сделать встроенным метод,
тело которого находится вне определения класса,
нужно использовать спецификатор inline

```
inline void MyClass::foo() { /*...*/ }
```

5) определение inline метода должно находиться в заголовочном файле!

6) спецификатор inline является рекомендацией компилятору

Встроенные (inline) методы: синтаксис

```
class A {
   int  value;
public:
   int  getValue() const
   {
      return value;
   }
   void setValue(int x)
   {
      value = x;
   }
};
```

```
class A {
   int  value;
public:
   int  getValue() const;
   void setValue(int x);
};

inline int A::getValue() const
   { return value; }

inline void A::setValue(int x)
   { value = x; }
```

Класс std::string

- безопасный класс для работы с символами
- строка – это массив символов произвольной длины

```
#include <string>

void f(){
    std::string myString = "Hello, world!";
}
```

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

Инициализация строк

```
#include <string>

void f(){
	std::string str0;
	std::string str1 = "";
	std::string str2 = "Hello, world!";
	const char* cStr = "Hello, world!";
	std::string str3 = cStr;  // инициализация строкой char*
	std::string str4 = str2;
	str1 = str4;
	std::string str5 = '!';     // Error
	str0 = '!';                 // OK
}
```

Длина строки

- методы size() и length()

  (нулевой байт не учитывается)
- проверка на пустоту – метод empty()

  (пустая – true, непустая – false)

```
#include <string>

void f(){
    std::string str = "Hello, world!";
    int len1 = str.length();          // len1 = 13;
    int len2 = str.size();            // len2 = 13;
    if ( str.empty() ) { str = "new value"; }
}
```

Доступ к символам

- перегружен оператор [ ]
- метод at() – генерирует исключение out_of_range при выходе за границы диапазона

```
#include <string>

void f(){
    std::string str = "abc";
    char symb1 = str[0];
    char symb2 = str.at(0);
    char symb3 = str[10];
    char symb4 = str.at(10);
}
```

Итерирование по строке

```
#include <string>
#include <iostream>

std::string str = "Hello, world!";
for (int i = 0; i < str.length(); ++i){
    std::cout << str[i];
}
```

```
#include <string>
#include <iostream>

std::string str = "Hello, world!";
for (std::string::iterator it = str.begin(); it != str.end(); ++it){
    std::cout << *it;
}
```

Итерирование по строке (C++11)

```
#include <string>
#include <iostream>

std::string str = "Hello, world!";
for (auto symbol : str){
	std::cout << symbol;
}
std::cout << std::endl;
```

Сравнение и конкатенация строк

- перегружены операторы == и !=
- перегружены операторы > и <
- перегружены операторы + и +=

```
#include <string>
#include <iostream>

void f(){
    std::string str1 = "abc";
    std::string str2 = "abcd";
    if (str1 != str2) { str1 += str2; }
}
```

Вопросы?