

САОД

А. Задорожный
2017

Вопросы для повторения

1. Из каких шагов состоит построение программы на C++?
2. Как в тексте программы обнаружить инструкции препроцессора? Приведите примеры инструкций препроцессора.
3. Назовите 4 основных типа данных, определенных в языке C++.
4. Можно ли в C++ написать $x = y = 3$? Почему?
5. Можно ли вызвать `void foo(int & a)` следующим образом: `foo(x+y);` Почему?
6. Проинтерпретируйте объявление второго параметра `main`.
`int main(int argc, char* argv[])`
7. Сколько операций умножения в операторе `*p = *p * 2`?
8. Если массив `w` проинициализирован значениями 1, 2, 3. Чему равно значение выражения `*w`?
9. Чему равна величина `p + i`, где `p` – указатель на массив целых, а `i` – целое число?

Содержание

- Классы, структуры, объекты
- Время жизни переменных (объектов)
- Переопределение операций
- Inline функции и методы
- Управление памятью и динамические объекты
- Глобальные переменные и статические поля
- Виды конструкторов в C++
- Ссылки в C# (.Net)
- Строки STL

Классы, структуры, объекты

Классы объявляются по аналогии с C#

```
class CDate {  
    int m_nYear;  
    int m_nMonth;  
    int m_nDay;  
public:           //Указывается не перед каждым метдом  
    CDate();  
    ~CDate() {}   //Конструктор, как правило, public  
};               //Точка с запятой обязательна!
```

Объявления, как правило, в заголовочном файле

Реализация методов

Date.cpp

#include "Date.h"

```
CDate::CDate () { //Указываем класс  
    m_nYear = 1;  
    m_nMonth = 1;  
    m_nDay = 1;  
}
```

Объявление и время жизни объектов

```
#include "Date.h"
```

```
int main(){
```

```
    CDate d;    // Это не ссылка, а объект!
```

```
    return 0;
```

```
}
```

- Локальные объекты создаются при выполнении объявления.
- Локальные объекты исчезают по завершении блока в котором они были объявлены.
- Глобальные объекты объявляются вне блоков.
- Глобальные объекты создаются до начала работы программы, а разрушаются после завершения программы

Конструкторы инициализации

```
class CDate {  
    int m_nYear;  
    int m_nMonth;  
    int m_nDay;  
public:  
    CDate();  
    CDate(int Year, int Mon=1, int Day=1);  
    ~CDate() {}  
};
```

Параметры по умолчанию задаются при объявлении!

Конструкторы инициализации

```
CDate::CDate (int Year, int Mon, int Day) {  
    if (Year < 1 || Year > 10000)  
        Year = 1;  
    if (Mon < 1 || Mon > 12)  
        Mon = 1;  
    int vDays[12]={31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
    if (Mon < 1 || Mon > 12)  
        Mon = 1;  
    if (Day < 1 || Day > vDays[Mon-1])  
        Day = 1;  
    if (Mon == 2)  
        if (Day == 29 && (Year % 4 > 0 || (Year % 100 == 0 && Year % 400 > 0)))  
        {  
            Day = 1;  
            Mon = 3;  
        }  
    m_nYear = Year;  
    m_nMonth = Mon;  
    m_nDay = Day;  
}
```

Конструктор обеспечивает правильность создаваемой даты

```
CDate d(2006, 10, 12), t(2006, 11), e(2006);
```

Структуры

- Структуры – те же классы!
- По умолчанию в них действует область видимости public

```
struct CDate {  
private:  
    int m_nYear;  
    int m_nMonth;  
    int m_nDay;  
public:  
    CDate();  
    ~CDate() {} //Конструктор, как правило, public  
};              //Точка с запятой обязательна!
```

Контрольные вопросы

1. К чему приведет объявление класса в C++ без использования слова `public`?
2. Почему в конце объявления класса ставится точка с запятой?
3. Определите время жизни объекта. Сформулируйте правило, по которому возникают и исчезают объекты в C++.
4. В чем различие между объектами в C++ и C# (.Net)?
5. Опишите время жизни объектов в C# (.Net)?
6. Что такое “конструктор инициализации”?
7. Опишите правила определения и использования параметров по умолчанию. (Где они указываются и почему, как они расположены в списке параметров, как могут опускаться при вызове)
8. В чем различие в C++ между `struct` и `class`?
9. Как будет исправлена дата при вызове `CDate d(0, 15, 2006)`, если конструктор реализован как в лекции? Какая дата будет задана при вызове `CDate d(2007)`?

Вызов функции и inline

Вызов функции требует времени:

1. Вычислить и поместить в стек параметры
2. Запомнить адрес возврата
3. Переключить стек и передать управление на функцию
4. ...
5. Переключить стек и вернуть управление в вызывающий код

Частые вызовы коротких функций снижают эффективность выполнения программ

Функцию можно пометить квалификатором inline. При этом, компилятор не будет генерировать эту функцию, а в местах вызова сгенерирует код, эквивалентный вызову.

```
inline int max(int a, int b){return a>b? a: b;}
```

Переопределение операций

Inline методы

```
class CDate {
    int m_nYear;
    int m_nMonth;
    int m_nDay;
public:
    CDate();
    CDate(int Year, int Mon=1, int Day=1);
    ~CDate() {}
    bool operator < (CDate &d) {
        return m_nYear < d.m_nYear
        || (m_nYear == d.m_nYear && m_nMonth < d.m_nMonth)
        || (m_nYear == d.m_nYear && m_nMonth == d.m_nMonth
        && m_nDay < d.m_nDay);
    }
};

cout << (t < d) << " " << (d < e) << endl;
```

“Динамические” объекты

```
Char *s = new char[128];  
CDate *p = new CDate();  
CDate *pv = new CDate[5];
```

```
delete [] s;  
delete p;  
delete [] pv;
```

- “Динамические” объекты создаются оператором new
- За удаление таких объектов отвечает программа!

Глобальные объекты

Глобальный объект должен включаться в h-файл

Что бы избежать, при этом, его экземпляров во множество объектных файлов в языке введен квалификатор `extern`

Н-файл -----

```
extern CDate UnixBase;
```

Один из CPP-файлов -----

```
CDate UnixBase(1970,1,1);
```

В других CPP, где включен h-файл можно использовать UnixBase!

Статические поля

Статические поля объявляются в классе с квалификатором `static`

h-файл -----

```
class CTest
{
    public:
        static int Count;
};
```

В CPP-файле -----

```
int CTest :: Count = 0
```

В других CPP, где включен h-файл можно использовать `CTest :: Count!`

Виды конструкторов

Класс CStr

```
class CStr {  
    char* m_pszText;  
public:  
    CStr () {m_pszText = 0;} //По умолчанию  
    ~ CStr () { delete []m_pszText;}  
};
```

Здесь деструктор обязателен!

Доработка CStr

Оператор преобразования типа
(CStr=>char*)

```
operator const char *()const {return m_pszText  
? m_pszText : "";}}
```

Теперь объекты CStr можно применять
везде, где нужно использовать z-строку!

Конструктор инициализации для CStr

```
CStr (const char * p)
{
    m_pszText = new char [strlen(p) + 1];
    strcpy (m_pszText, p);
}
```

- Теперь можно:
CStr s = "abc";

Варианты использования CStr

Неправильное копирование

```
void Test1 (CStr sz)
```

```
{
```

```
    CStr Test2 () {  
        CStr sz= "123";  
        return sz;  
    }
```

```
}
```

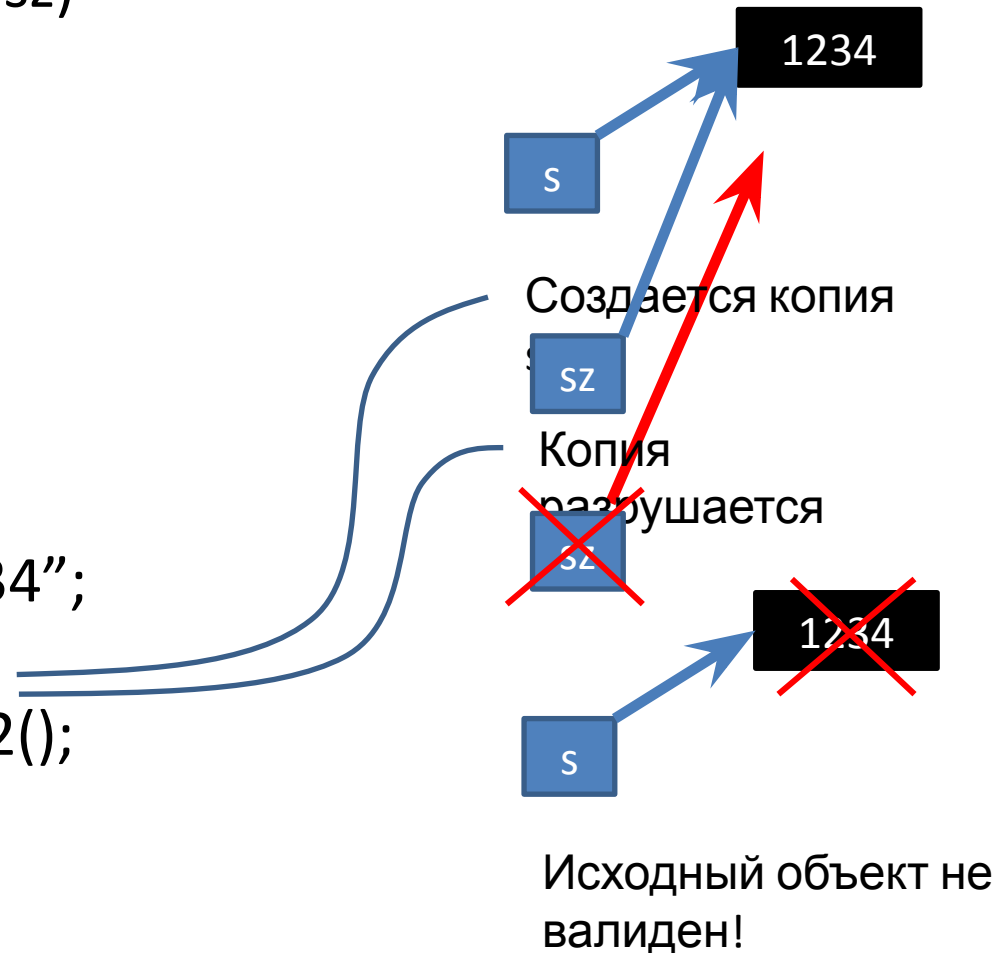
```
int main(){
```

```
    CStr s = "1234";
```

```
    Test1(s);
```

```
    CStr t = Test2();
```

```
}
```

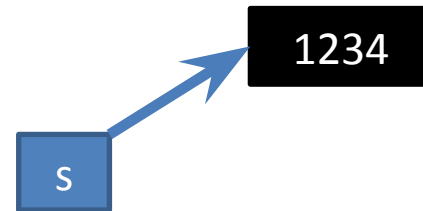


Конструктор копирования

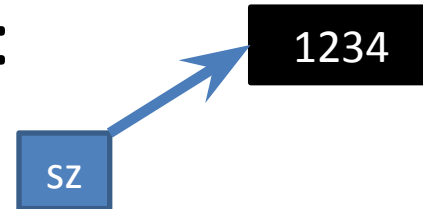
```
CStr (const CStr &s) //Константная ссылка на объект
{
    if (s.m_pszText == 0)
        m_pszText = 0;
    else {
        m_pszText = new char
[strlen(s.m_pszText) + 1];
        strcpy (m_pszText, s.m_pszText);
    }
}
```

Создание копии

Теперь есть правила
Копирования



В результате получаем:



Копии можно создавать и использовать
без ограничений!

Ссылки и объекты в С#

Reference Type (Value Types похожи на C++)

- Всегда ссылка, но !
- Ссылка в .Net – это самостоятельная переменная!
- Копирование объектов в .Net довольно сложная задача!

Ссылки и объекты в C#

Сравнение понятий

C++	C# (.Net)
Ссылка не может не ссылаться на объект (это не самостоятельная переменная, а просто другое имя объекта)	Ссылка является самостоятельной сущностью. Есть область видимости и область существования. Может никуда не ссылаться или ссылаться на разные объекты в течение времени жизни.
При передаче параметра по ссылке (&) передается адрес объекта, на который сослались при вызове метода.	При передаче в качестве параметра ссылка копируется! Что бы передать ссылку на ссылку используется квалификатор ref.
О наличии ссылок на объект узнать в программе нельзя!	За наличием ссылок на объект следит сборщик мусора (Garbage Collector)

Строки стандартной библиотеки

```
#include <iostream>
```

```
#include <string>    // Появился тип string.
```

```
using namespace std;
```

```
string s;
```

```
cin >> s;
```

```
cout << s.length() << endl;
```

```
cout << s[0] << endl;
```

```
cout << (s + s) << endl;
```

```
...
```

Более полно будет рассмотрено в дальнейшем...

Контрольные вопросы

1. Что означает запись `o.operator = (t)`? Как ее можно эквивалентно записать иначе?
2. *Переопределите операцию `==` для `CDate`.*
3. Что такое и зачем нужны inline функции и методы?
4. Как объявить inline метод в классе?
5. Опишите объявление и использование глобальных переменных в C++. Как их объявить, что бы можно было использовать в разных модулях? Как добиться того, что бы был создан только 1 уникальный экземпляр глобальной переменной?
6. Опишите объявление статических полей класса в C++.
7. Перечислите виды конструкторов применяемых в C++.
8. Почему для `CDate` не нужен деструктор и конструктор копии, а для `CStr` нужны?
9. Почему в конструктор копии передают ссылку на объект, а не копию объекта?
10. Приведите примеры, когда копии объектов возникают без явного вызова конструктора.
11. Опишите, чем понятие ссылки в .Net отличается от ссылки в C++.