Объектно-ориентированное программирование

Особенности работы присваивания, копирования, типизации классов

Ограничение на наследование

```
1  class alpha
2  {
3  private:
4      int data;
5  public:
6      alpha() //конструктор без аргументов
7          { }
8      alpha(int d) //конструктор с одним аргументом
9          { data = d; }
10     alpha(alpha& a) //конструктор копирования
11     {
12         data = a.data;
13         cout << "\nЗапущен конструктор копирования";
14     }
15     void display() //display
16         { cout << data; }
17     void operator = (alpha& a) //overloaded = operator
18     {
19         data = a.data;
20         cout << "\nЗапущен оператор присваивания";
21     }
22 };
23 //////////////////////////////////////////////////////////
24 int main()
25 {
26     alpha a1(37);
27     alpha a2;
28     a2 = a1;                    //запуск перегружаемого =
29     cout << "\na2="; a2.display(); //вывести a2
30     alpha a3(a1); //запуск конструктора копирования
31     // alpha a3 = a1; //эквивалентное определение a3
32     cout << "\na3="; a3.display(); //вывести a3
33     cout << endl;
34     return 0;
35 }
```

Оператор присваивания уникален среди остальных операторов тем, что он не может наследоваться. Перегрузив присваивание в базовом классе, вы не сможете использовать ту же функцию в порожденных классах.

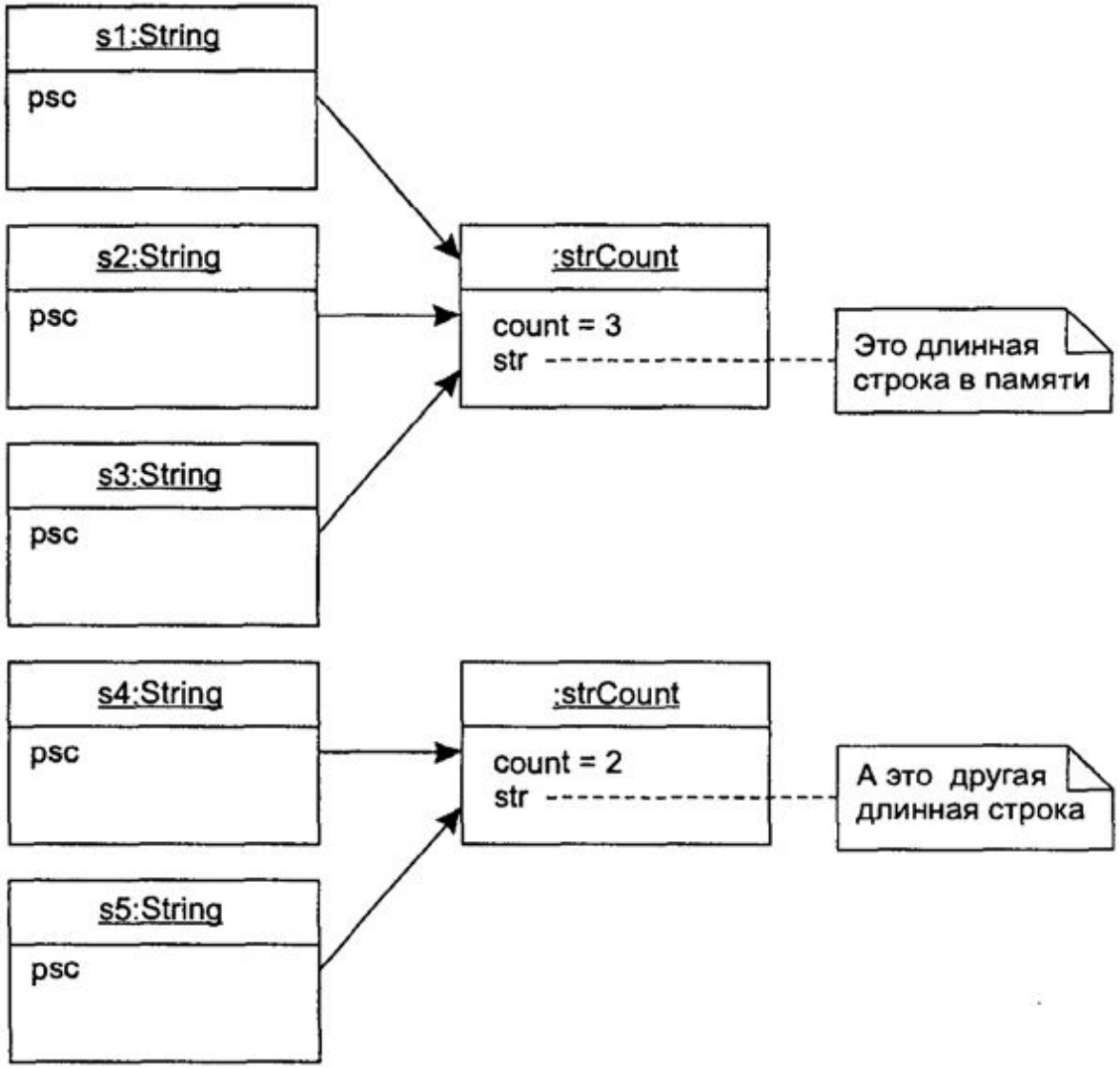
alpha(alpha a)

Запрещение копирования

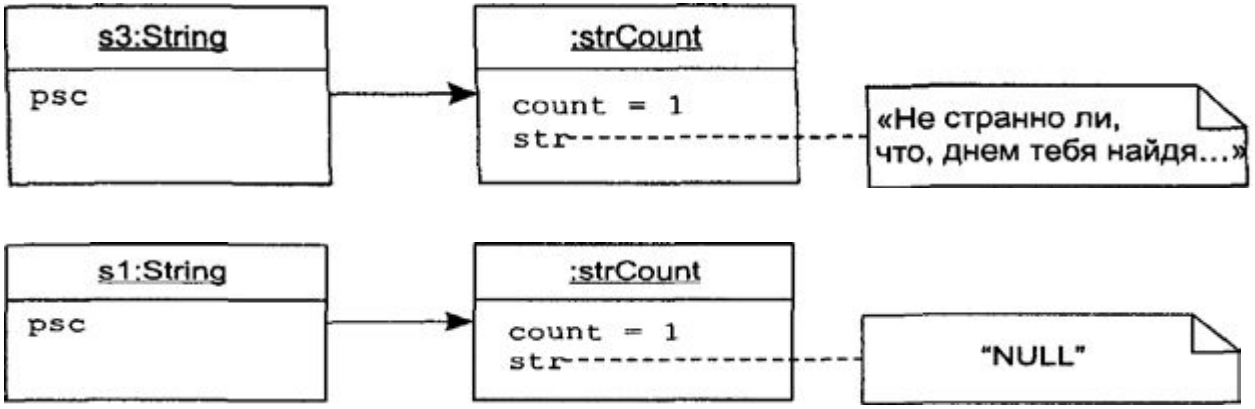
```
class alpha
{
    private:
    alpha& operator = (alpha&); //Скрытое присваивание
    alpha(alpha&); //Скрытое копирование
};

alpha a1,a2;
a1 = a2; //присваивание
alpha a3(a1); //копирование
```

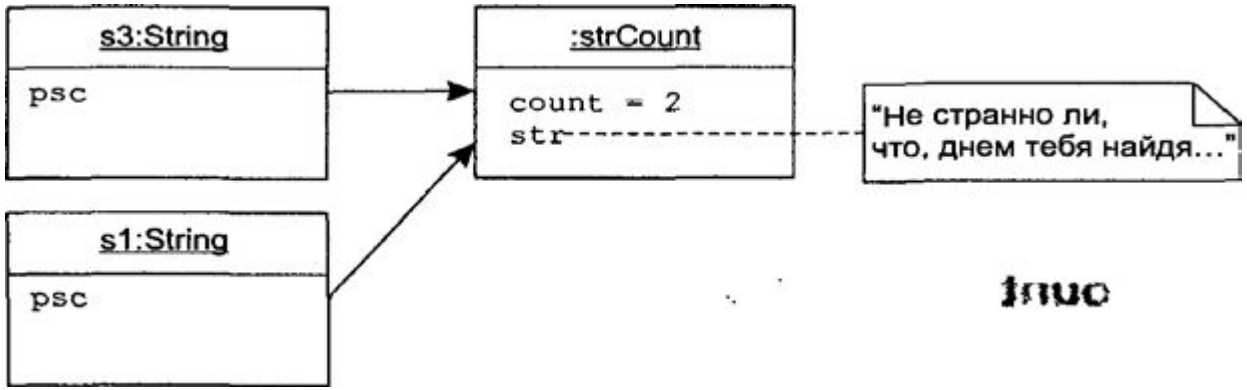
Эффективное использование памяти классом String

Эффективное использование памяти классом String



Эффективное использование памяти классом String





Эффективное использование памяти классом String

```
1  class strCount // Класс-счетчик уникальных строк
2  {
3  private:
4      int count; // собственно счетчик
5      char* str; // указатель на строку
6      friend class String; // сделаем себя доступными
7      //методы скрыты
8      strCount(char* s) // конструктор с одним аргументом
9      {
10         int length = strlen(s); // длина строкового
11         // аргумента
12         str = new char[length+1]; // занять память
13         // под строку
14         strcpy(str, s); // копировать в нее аргументы
15         count=1; // считать с единицы
16     }
17     ~strCount() // деструктор
18         { delete[] str; } // удалить строку
19 };
```

Эффективное использование памяти классом String

```
class String // класс String
{
private:
    strCount* psc; // указатель на strCount
public:
    String() // конструктор без аргументов
        { psc = new strCount("NULL"); }
    //-------------------------------------------------------
    String(char* s) // конструктор с одним аргументом
        { psc = new strCount(s); }
    //-------------------------------------------------------
    String(String& S) // конструктор копирования
    {
        psc = S.psc;
        (psc->count)++;
    }
    //-------------------------------------------------------
    ~String() // деструктор
    {
        if(psc->count==1) // если последний
        // пользователь,
        delete psc; // удалить strCount
        else // иначе
        (psc->count)--; // уменьшить счетчик
    }
    //-------------------------------------------------------
    void display() // вывод String
    {
        cout << psc->str; // вывести строку
        cout << " (addr=" << psc << ")"; // вывести адрес
    }
    //-------------------------------------------------------
    void operator = (String& S) // присвоение String
    {
        if(psc->count==1) // если последний пользователь,
            delete psc; // удалить strCount
        else // иначе
            (psc->count)--; // уменьшить счетчик
        psc = S.psc; //использовать strCount
        //аргумента
        (psc->count)++; //увеличить счетчик
    }
};
```

```
int main()
{
    String s3 = "Муха по полю пошла, муха денежку нашла";
    cout << "\ns3="; s3.display(); //вывести s3
    String s1; //определить объект String
    s1 = s3; //присвоить его другому объекту
    cout << "\ns1="; s1.display(); //вывести его
    String s2(s3); //инициализация
    cout << "\ns2="; s2.display(); //вывести
    //инициализированное
    cout << endl;
    return 0;
}
```

Указатель this

```
1  class what
2  {
3  private:
4      int alpha;
5  public:
6      void tester()
7      {
8          this->alpha = 11; //то же,что alpha = 11;
9          cout << this->alpha; //то же, что cout <<
10         //alpha;
11     }
12 };
13 ////////////////////////////////////////////////
14 int main()
15 {
16     what w;
17     w.tester();
18     cout << endl;
19     return 0;
20 }
```

Использование this для возврата значений

```
class alpha
{
private:
    int data;
public:
    alpha() // конструктор б
        { }
    alpha(int d) // конструк
        { data = d; }
    void display() // вывест
        { cout << data; }
    alpha& operator = (alpha& a) // перегружаемая операция =
    {
        data = a.data; // не делается автоматически
        cout << "\nЗапущен оператор присваивания";
        return *this; // вернуть копию this alpha
    }
};
//////////////////////////////////////////////////////////
int main()
{
    alpha a1(37);
    alpha a2, a3;
    a3 = a2 = a1; // перегружаемый =, дважды
    cout << "\na2="; a2.display(); // вывести a2
    cout << "\na3="; a3.display(); // вывести a3
    cout << endl;
    return 0;
}
```

```
String& operator = (String& S)//присвоение String
{
    cout << "\nПРИСВАИВАНИЕ";
    if(psc->count==1) //если последний  пользователь,
        delete psc; //удалить strCount
    else //иначе
        (psc->count)--;// уменьшить счетчик
    psc = S.psc; //использовать strCount
    //аргумента
    (psc->count)++; //увеличить счетчик
    return *this; //вернуть этот объект
}
```

```
if(this == &S)
    return *this;
```

Динамическая информация о типах

Чтобы заработали dynamic_cast и typeid, компилятор должен активизировать механизм, который позволяет определять и изменять тип объекта во время выполнения программы — RTTI (Run-Time Type Information). В системе Borland C++ Builder этот механизм включается по умолчанию, а в Microsoft Visual C++ нужно

```
class Base
{
    virtual void vertFunc() //для dynamic cast
        { }
};
class Derv1 : public Base
    { };
class Derv2 : public Base
    { };
    //проверка, указывает ли pUnknown на Derv1
bool isDerv1(Base* pUnknown) //неизвестный подкласс базового
{
    Derv1* pDerv1;
    if( pDerv1 = dynamic_cast<Derv1*>(pUnknown) )
        return true;
    else
        return false;
}
```

```
Derv1* d1 = new Derv1;
Derv2* d2 = new Derv2;
```

Изменение типов указателей с помощью dynamic_cast

```
class Base
{
protected:
    int ba;
public:
    Base() : ba(0)
        { }
    Base(int b) : ba(b)
        { }
    virtual void vertFunc() //для нужд dynamic_cast
        { }
    void show()
        { cout << "Base: ba=" << ba << endl; }
};
////////////////////////////////////////////////////////////
class Derv : public Base
{
private:
    int da;
public:
    Derv(int b, int d) : da(d)
        { ba = b; }
    void show()
        { cout << "Derv: ba=" << ba << ", da=" << da << endl; }
};
////////////////////////////////////////////////////////////////
int main()
{
    Base* pBase = new Base(10); //указатель на Base
    Derv* pDerv = new Derv(21, 22); //указатель на Derv
    //приведение к базовому типу: восхождение по дереву -
    //указатель поставлен на подобъект Base класса Derv
    pBase = dynamic_cast<Base*>(pDerv);
    pBase->show(); //"Base: ba=21"
    pBase = new Derv(31, 32); //обычное нисходящее
    //приведение типов -- pBase должен указывать на Derv)
    pDerv = dynamic_cast<Derv*>(pBase);
    pDerv->show(); //"Derv: ba=31, da=32"
    return 0;
}
```

Оператор typeid

```
// RTTI должен быть активизирован
#include <iostream>
#include <typeinfo> // для typeid()
using namespace std;
//////////////////////////////////////////////////////////
class Base
{
    virtual void virtFunc() // для нужд typeid
        { }
};
class Derv1 : public Base
    { };
class Derv2 : public Base
    { };
//////////////////////////////////////////////////////////
void displayName(Base* pB)
{
    cout << "указатель на объект класса "; // вывести имя класса
    cout << typeid(*pB).name() << endl; //на который
                                        //указывает pB
}
//----------------------------------------------------------
int main()
{
    Base* pBase = new Derv1;
    displayName(pBase); //"указатель на объект класса Derv1"
    pBase = new Derv2;
    displayName(pBase); //" указатель на объект класса Derv2"
    return 0;
}
```