

# Глава 5. Объектная модель C++

МГТУ им. Н.Э. Баумана

Факультет Информатика и системы управления

Кафедра Компьютерные системы и сети

Лектор: д.т.н., проф.

Иванова Галина Сергеевна

## 5.1 Описание класса и создание объектов

Формат описания класса:

**class** <Имя класса>

```
{ private:    <Внутренние компоненты класса>;  
  protected: <Защищенные компоненты класса>;  
  public:     <Общедоступные компоненты класса>;  
};
```

**Внутренние** компоненты класса – поля и методы – доступны только методам класса (В Delphi Pascal – в пределах модуля).

**Защищенные** – доступны методам своего класса и классов-наследников.

**Общедоступные** – доступны в пределах видимости, в том числе из программы и методов других классов.

# Пример 5.1. Класс Книга

Диаграмма  
классов

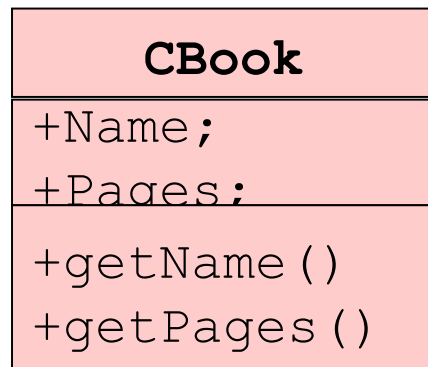
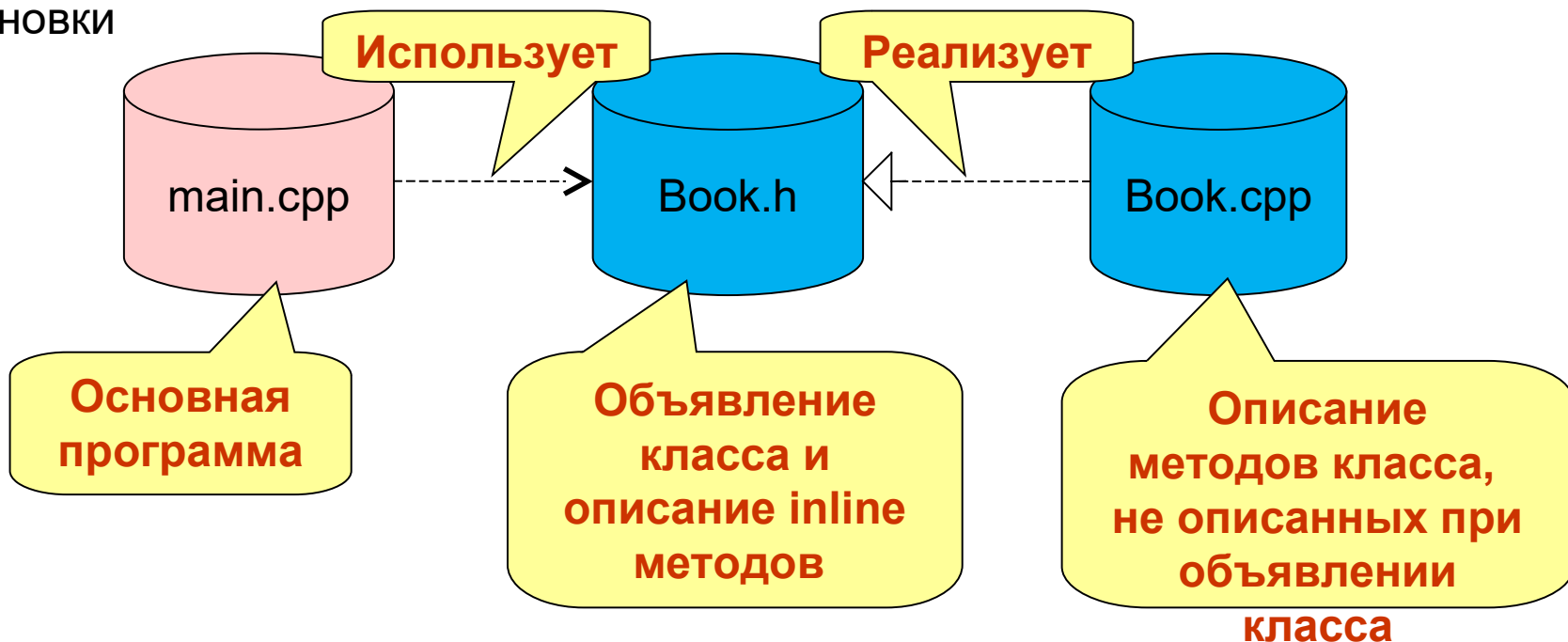


Диаграмма  
компоновки



# Объявление класса

Файл Book.h:

```
#ifndef book_h  
#define book_h
```

Защита от повторной  
компиляции описания  
класса (начало)

```
class CBook  
{  
public:  char Name[30];  
        int Pages;  
        char *getName() // метод по умолчанию считается inline,  
        {               // так как его тело описано в классе  
            return Name;  
        }  
        int getPages(); // тело метода будет описано в book.cpp  
};  
#endif
```

Защита от повторной  
компиляции описания  
класса (завершение)

# Описание методов в специальном файле

// Файл Book.cpp – секция реализации модуля book

```
#include "book.h"
```

// тело метода

```
int CBook::getPages()
```

```
{
```

```
    return Pages;
```

```
}
```

В отличие от методов, описанных в классе, методы, описанные в файле реализации модуля, не являются по умолчанию объявленными **inline**. При необходимости директива **inline** указывается явно при объявлении метода в классе.

# Объявление неинициализированных объектов при отсутствии в классе конструктора

Объявление объектов:

<Имя класса> <Список переменных и/или указателей>;

Пример:

```
#include "book.h"
```

```
void main()
```

```
{
```

```
    CBook a,    // простая переменная-объект
```

```
        *b,    // указатель на объект (память под объект  
                не выделена)
```

```
        c[5]; // массив из 5 объектов в статической памяти
```

```
...
```

# Инициализация **общедоступных** полей объектов при объявлении в случае отсутствия в класса конструктора

Для инициализации общедоступных полей объектов может быть использована та же конструкция, что и для инициализации полей структуры.

Пример:

```
#include "book.h"
```

```
int main()
```

```
{ CBook A = {"J.London. V.1", 366};  
  CBook C[] = { {"J.London. V.3", 367},  
                {"J.London. V.4", 321},  
                {"J.London. V.5", 356} };  
... }
```

**Память будет  
выделена под  
массив из трех  
объектов**

# Обращение к полям и методам класса

а) простой объект:

**<Имя объекта>.<Имя поля или метода>**

б) указатель на объект:

**<Имя указателя на объект> -><Имя поля или метода>**

в) массив объектов

**<Имя массива>[<Индекс>] .<Имя поля или метода>**

Внутренний указатель на поля объекта:

**указатель**

**this (C++)  $\Leftrightarrow$  Self (Паскаль)**

**Ссылка**

Пример: **this->Pos**



# Основная программа

```
#include "book.h"
#include <stdio.h>
int main()
{
    // объявление инициализированного объекта
    CBook B={"J.London. Smoke Bellew",267};
    printf("%s %d\n",B.getName(),B.getPages());
    // объявление массива инициализированных объектов
    CBook C[]= {{"J.London. V.3",367},
                {"J.London. V.4",321},
                {"J.London. V.5",356}};
    for(int i=0;i<3;i++)
        printf("%s %d\n",C[i].getName(),C[i].getPages());
    return 0;
}
```

## 5.2 Конструкторы и деструкторы

**Конструктор** – метод, **автоматически** вызываемый при выделении памяти под объект. Используется для инициализации полей объекта. Автоматический вызов страхует программиста от ошибки оставить поля неинициализированными.

```
class CBook
{protected: char Name[30];
    int Pages;
public:
    CBook(const char *name,int pages) {
        Pages=pages;  strcpy(Name,name);
    }...
```

При создании объектов классов с конструкторами параметры записываются в круглых скобках:

```
int main()
{   CBook D("J.London. Smoke Bellew",267);
```

# Конструкторы без параметров

Конструкторы, как и другие функции, можно перегружать. Специальный конструктор без параметров (инициализирующий или неинициализирующий) используется для создания объектов, которым при выделении памяти не могут быть переданы значения полей.

- a) `CBook () { }` // неинициализирующий конструктор без параметров, используется для создания неинициализированных объектов
- б) `CBook () { Name[0]='\0'; Pages=0; }` // инициализирующий конструктор без параметров, создает однотипно инициализированные объекты

При создании объектов посредством конструкторов без параметров круглые скобки не указывают:

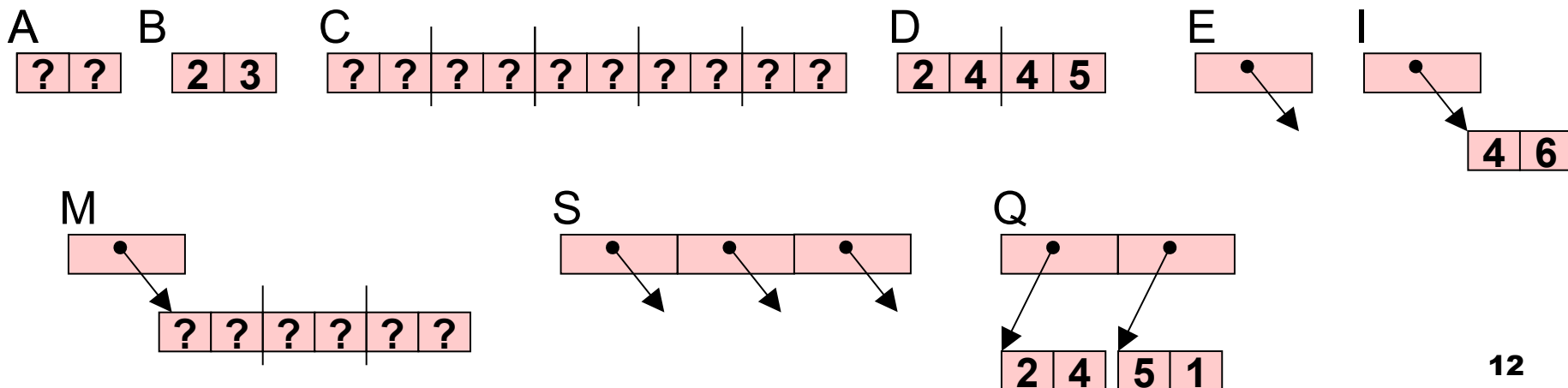
```
void main()  
{   CBook J; ...
```

# Различные способы создания объектов

Пример (Ex5\_4):

```
class CPoint
{ private: int x,y;
  public: CPoint(int ax,int ay){x=ax;y=ay;}
         CPoint(){}
         SetPoint(int ax,int ay) {x=ax;y=ay;}    ...};

int main()
{ CPoint A, B(2,3), C[5], D[2] = {CPoint(2,4),CPoint(4,5)},
  *E,*I = new CPoint(4,6), *M = new CPoint[3],
  *S[3], *Q[]={new CPoint(2,4),new CPoint(5,1)};
```



# Распределение/освобождение памяти и инициализация объектов в программе

```
A.SetPoint(2,3); A.Print(); A      B      C
B.Print();                        ??  2 3  ?? ?? ?? ?? ?? ?? ??
for (i=0;i<5;i++) {C[i].SetPoint(i,i+1); C[i].Print();}
for(i=0;i<2;i++) d[i].Print();    D      E      I
E=new CPoint(3,4); E->Print();    2 4 4 5  [ ] [ ]
I->Print();                      [ ] [ ]
for (i=0;i<3;i++) {M[i].SetPoint(i,i+1); M[i].Print();}
for (i=0;i<3;i++) {S[i]=new CPoint(i,i+1); S[i]->Print();}
for (i=0;i<2;i++) Q[i]->Print();
delete E; delete I;
delete [] M;
for (i=0;i<3;i++) delete S[i];
for (i=0;i<2;i++) delete Q[i];
}
```

Diagram illustrating memory distribution and object initialization:

- A**: A single object with values 2 and 3.
- B**: A single object with values 2 and 3.
- C**: An array of 10 objects, each with values 0, 1; 1, 2; 2, 3; 3, 4; 4, 5; 5, 6; 6, 7; 7, 8; 8, 9; 9, 10.
- D**: An array of 4 objects, each with values 2, 4; 4, 4; 4, 5; 5, 5.
- E**: A single object with values 3 and 4.
- I**: A single object with values 4 and 6.
- M**: An array of 3 objects, each with values 0, 1; 1, 2; 2, 3.
- S**: An array of 3 objects, each with values 0, 1; 1, 2; 2, 3.
- Q**: An array of 2 objects, each with values 2, 4; 5, 1.

# Список инициализации.

## Инициализация объектных полей

Формат элемента списка инициализации:

<Имя поля>(<Список выражений>)

Примеры:

а) `TPoint(int ax, ay) : x(ax), y(ay) {}`

б) без списка инициализации нельзя инициализировать:

```
class TLine
```

```
{ private:
```

```
    const int x; // константные поля
```

```
    int &y;       // ссылочные поля
```

```
    TPoint t;    // объектные поля
```

```
public: TLine(int ax, int ay, int tx, int ty) :
```

```
        x(ax), y(ay), t(tx, ty) {}
```

```
    TLine() {}
```

```
...};
```

Автоматически вызывает  
конструктор объектного поля  
без параметров `TPoint( ) !`

# Объекты с динамическими полями.

## Деструкторы

**Деструкторы** аналогично конструкторам вызываются автоматически, но в момент освобождения памяти, выделенной под объект. Деструкторы обычно используют для освобождения памяти, выделенной под динамические поля объектов.

```
class CBook
{ char *pName;    int Pages;
public:
    CBook(const char *name,int pages) {
        pName=new char[30];
        strcpy(pName,name);
        Pages=pages;
    }
    ~CBook() {
        delete [] pName;
    }
};
```

# Копирующий конструктор

Автоматически вызывается:

а) при использовании объявлений типа

```
TPoint A(2,5), B=A;
```

б) при передаче параметров-объектов по значению, например:

```
void Print(TPoint R) {...}
```

Формат:

```
<Имя конструктора>(const <Имя класса> &<Имя объекта>) {...}
```

Примеры:

а) 

```
TPoint(const TPoint &Obj)
    {x=Obj.x; y=Obj.y;}
```

Строится  
автоматически

б) 

```
TPoint(const TPoint &Obj)
    {x=Obj.x; y=2*Obj.y;}
```

Любой можем  
объявить сами



# Пример обязательного определения копирующего конструктора (Ex5\_05)

```
#include <stdio.h>
```

```
class TNum
```

```
{ public:    int *pn;
```

```
    TNum(int n){puts("new pn"); pn=new int(n);}
```

```
    TNum(const TNum &Obj)
```

```
        {puts("copy new pn"); pn=new int(*Obj.pn);}
```

```
    ~TNum(){puts("delete pn");delete pn;}
```

```
};
```

```
void Print(TNum b) { printf("%d ",*b.pn); }
```

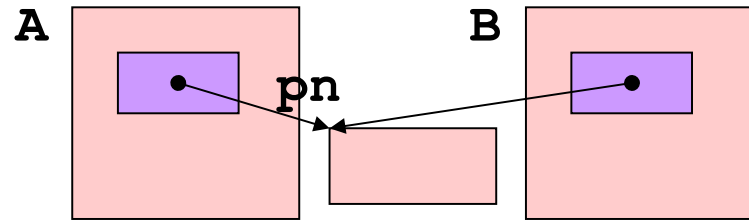
```
int main(int argc, char* argv[]) {
```

```
    TNum A(1);
```

```
    Print(A);
```

```
    return 0;
```

```
}
```



## 5.3 Наследование

`class <Имя производного класса>:`

`<Вид наследования> <Имя базового класса>{...};`

Вид наследования	Объявление компонента в базовом классе	Видимость компонента в производном классе
private	private protected public	НЕ ВИДИМЫ private private
protected	private protected public	НЕ ВИДИМЫ protected protected
public	private protected public	НЕ ВИДИМЫ protected public

# Конструкторы и деструкторы производных классов

**A**

**A () { ... }**

**B**

**B () : A () , <Конструкторы полей> { }**

**C**

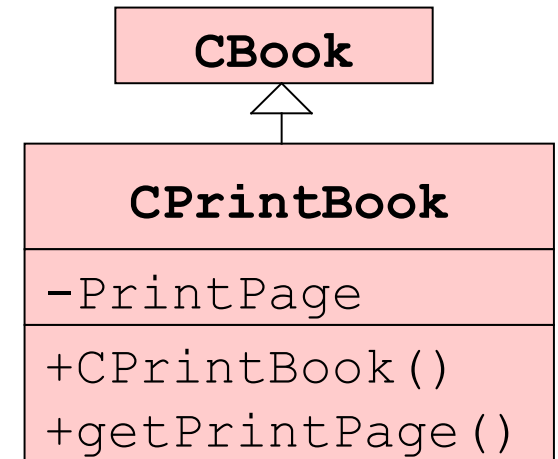
**C () : B () , <Конструкторы полей> { }**

При объявлении объектов производного класса всегда вызывается конструктор базового класса, используемый для инициализации наследуемых полей.

Если в списке инициализации конструктора производного класса вызов конструктора базового отсутствует, то **автоматически** вызывается конструктор базового класса **без параметров**

# Пример наследования Ex5\_02 (PrintBook.h)

```
#ifndef printbook_h
#define printbook_h
#include "Book.h"
class CPrintBook:public CBook{
private:    int PrintPages;
public:
    CPrintBook(const char *name,int pages):
        PrintPages=Pages/16;
    }
    int getPrintPages() {
        return PrintPages;
    }
};
#endif
```



**CPrintBook (name , pages) {**

Инициализация  
наследуемых  
полей базового  
класса

# Вызов конструкторов и деструкторов для объектов производных классов (Ex5\_06)

```
#include <stdio.h>

class TNum
{ public:      int n;
  TNum(int an):n(an) {puts("TNum(an)");}
  TNum() {puts("TNum()");}
  ~TNum() {puts("~TNum");}
};

class TNum2:public TNum
{ public:      int nn;
  TNum2(int an):nn(an) {puts("TNum2(an)");}
  ~TNum2() {puts("~TNum2");}
};

int main(int argc, char* argv[])
{ TNum2 A(1);
  return 0;
}
```

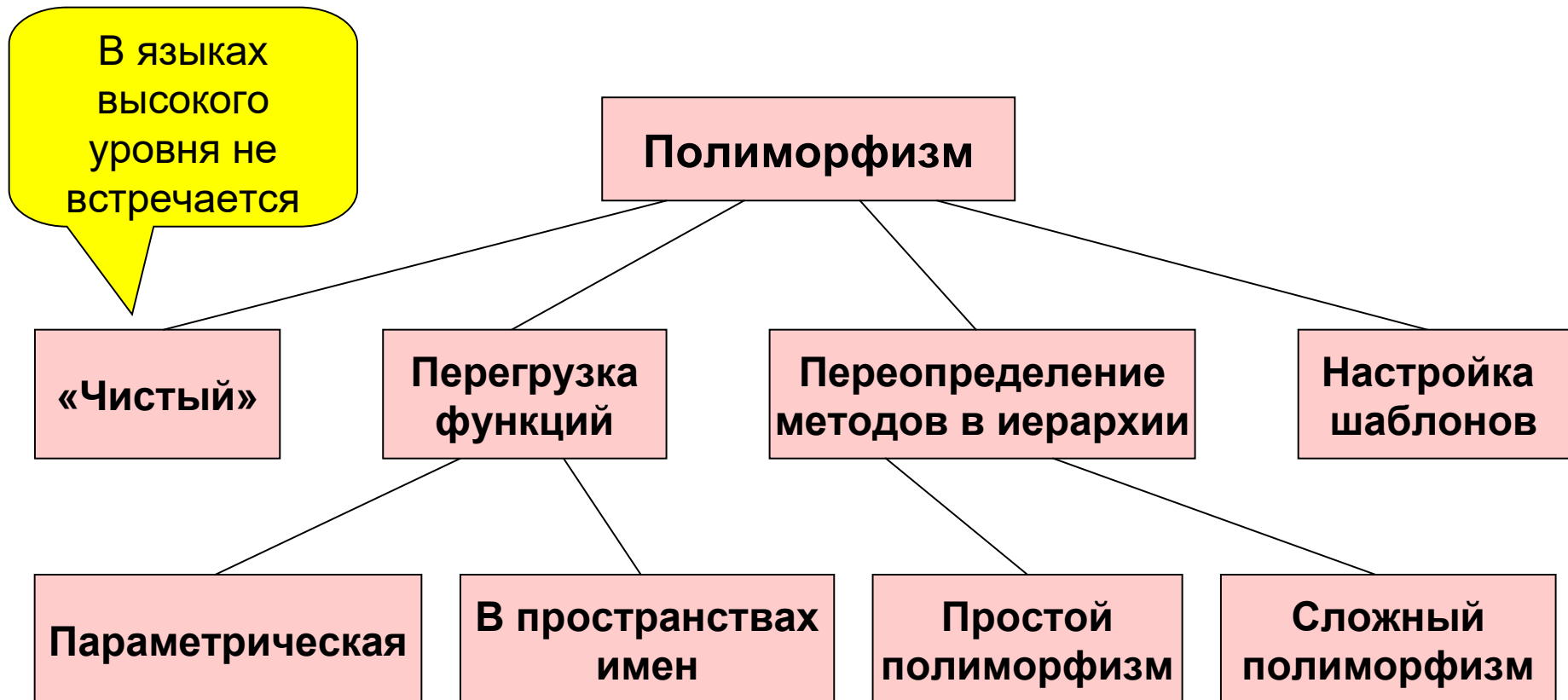
Неявный вызов  
конструктора

TNum()

TNum()  
TNum2(an)  
~TNum2  
~TNum

## 5.4 Полиморфизм. Полиморфное наследование

**Полиморфизм** – «многоформие», т.е. свойство изменения формы. В программировании встречаются следующие виды полиморфизма:



# Полиморфное наследование

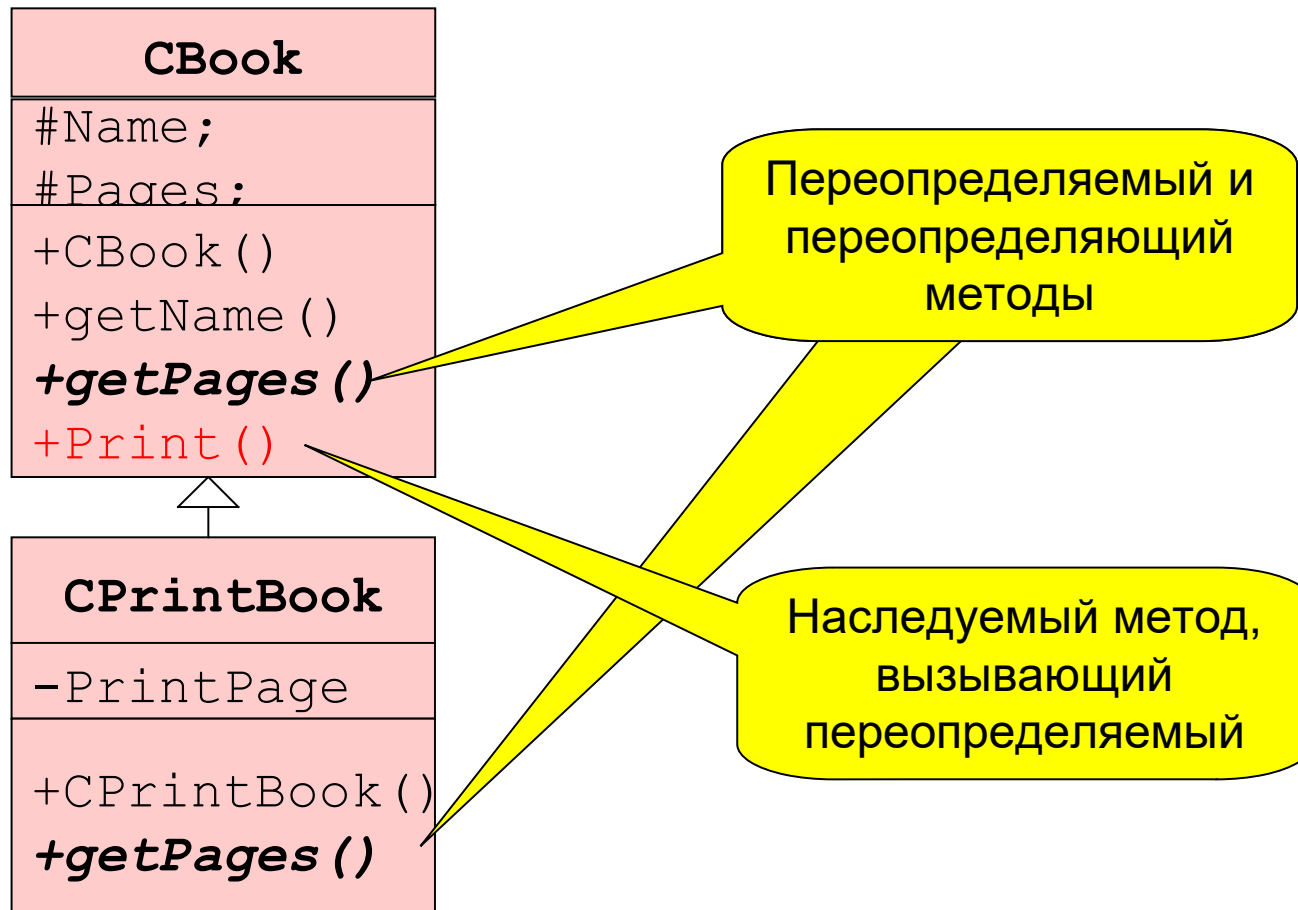
В Паскале:

простой полиморфизм  
сложный полиморфизм

В C++:

переопределение методов  
виртуализация методов

Пример использования сложного полиморфизма (**Ex5\_03**):



# Объявление класса CBook (файл Book.h)

```
#ifndef book_h
#define book_h
#include <string.h>
#include <stdio.h>
class CBook
{ protected: char Name[30];          int Pages;
public:
    CBook(const char *name,int pages):Pages(pages) {
        strcpy(Name,name) ;
    }
    char *getName() {
        return Name;
    }
    virtual int getPages() {
        return Pages;
    }
    void Print() {
        printf("%s %d\n",getName(),getPages());
    }
};
#endif
```



## Класс CPrintBook (файл PrintBook.h)

```
#ifndef printbook_h
#define printbook_h
#include "Book.h"

class CPrintBook:public CBook{
private:    int PrintPages;
public:
    CPrintBook(const char *name,int pages):
                                                CBook (name ,pages) {
        PrintPages=Pages/16;
    }
    int getPages() {
        return PrintPages;
    }
};
#endif
```

# Основная программа

```
#include "PrintBook.h"
```

```
int main()
```

```
{
```

```
    CBook F("J.London. Smoke Bellew",267);
```

```
    F.Print();
```

```
    CPrintBook D("J.London. Smoke Bellew",267);
```

```
    D.Print();
```

```
    return 0;
```

```
}
```

J.London. Smoke Bellew 267

J.London. Smoke Bellew 16

# Абстрактные методы и классы

Абстрактный метод.

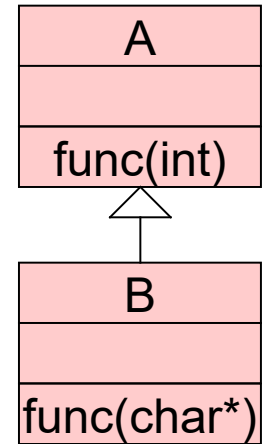
```
class AClass
{
    ...
    virtual int Fun(int,int)=0;
}
```

Класс, содержащий абстрактный метод, называется **абстрактным**.

Объекты абстрактного класса создавать запрещено !

# Использование пространств имен для перегрузки методов класса (Ex5\_09)

```
#include <iostream>
class A{
    public:      void func(int ch) ;
};
class B : public A{
    public:
        void func(const char *str) ;
        using A::func;      // перегрузка B::func
};
void A::func(int ch){      // метод базового класса
    std::cout << "Symbol\n";
}
void B::func(const char *str){ // метод производного класса
    std::cout << "String\n";
}
int main() {
    B b;
    b.func(25) ;           // вызов  A::func()
    b.func("ccc") ;        // вызов  B::func()
    return 0;
}
```



# Константные объекты и перегрузка методов для них

C++ разрешает создавать константные объекты, например:

```
<Класс> const a(1); или  
const <Класс> a(1);
```

Для константных объектов возможно написание специальных методов, в которых недопустимо изменение полей объекта, например:

```
class A {  
private:    int x;  
public:  
    void f(int a) const { // в метод передается константный объект  
        x = a; // ошибка компиляции !!!  
    }  
};
```

# Перегрузка методов для константного объекта

```
#include <iostream>
using namespace std;
class A
{
private:  int x;
public:
    A(int a) { x = a; cout << "A(int) // x=" << x << endl; }
    void f() { cout << "f() // x=" << x << endl; }
    void f() const { cout << "f() const // x=" << x << endl; }
};

int main() {
    A a1(1);
    a1.f();
    A const a2(2);
    a2.f();
    return 0
}
```

Метод  
перегружен для  
константного  
объекта

A(int) // x=1  
f() // x=1  
A(int) // x=2  
f() const // x=2

## 5.6 Приведение типов объекта

В C++ для приведения типов используют:

- 1) `(<Тип>)<Переменная>` – используется в Си/C++ для любых типов, ничего не проверяет;
- 2) `static_cast <Тип>(<Переменная>)` – используется в C++ для любых типов, ничего не проверяет;
- 3) `reinterpret_cast <Тип указателя>`  
`(<Указатель или интегральный тип>)` – используется в C++ для указателей, ничего не проверяет;
- 4) `dynamic_cast <Тип указателя на объект>`  
`(<Указатель на объект>)` – используется в C++ для **полиморфных** классов, требует указания опции компилятора `/GR` (см. меню **Project/Properties/Configuration Properties/ C\_C++/Language/Enable Run-Time Type Info = Yes**), если приведение невозможно, то возвращает `NULL`.

# Пример приведения типов объектов (Ex5\_07)

```
#include <iostream.h>
```

```
#include <string.h>
```

```
class TA
```

```
{ protected:      char c;
```

```
  public: TA(char ac):c(ac) {}
```

```
    virtual void func() {cout<<c<<endl;}
```

```
};
```

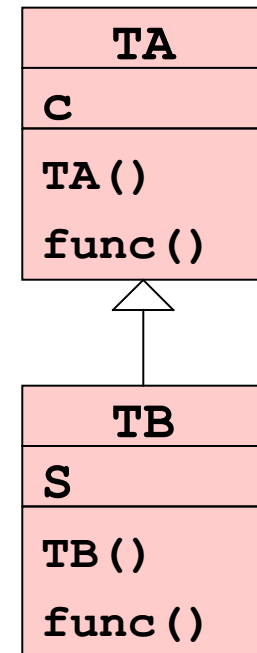
```
class TB:public TA
```

```
{      char S[10];
```

```
  public: TB(char *aS):TA(aS[0]) {strcpy(S,aS) ;}
```

```
    void func() {cout<<c<<' ' <<S<<endl;}
```

```
};
```





## Пример приведения типов объектов(2)

```
int main(int argc, char* argv[])
{ TA *pA=new TA('A'), *pC=new TB("AB");
  TB *pB=new TB("AC");
```

```
  ((TA *)pB)->func();
  reinterpret_cast<TA *>(pB)->func();
  static_cast<TA *>(pB)->func();
  dynamic_cast<TA *>(pB)->func();
```

Восходящее  
приведение

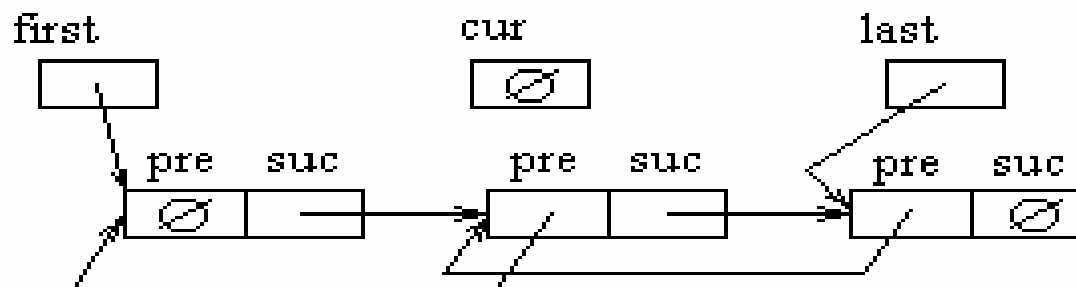
```
  ((TB *)pC)->func();
  reinterpret_cast<TB *>(pC)->func();
  static_cast<TB *>(pC)->func();
  dynamic_cast<TB *>(pC)->func();
```

Нисходящее  
приведение

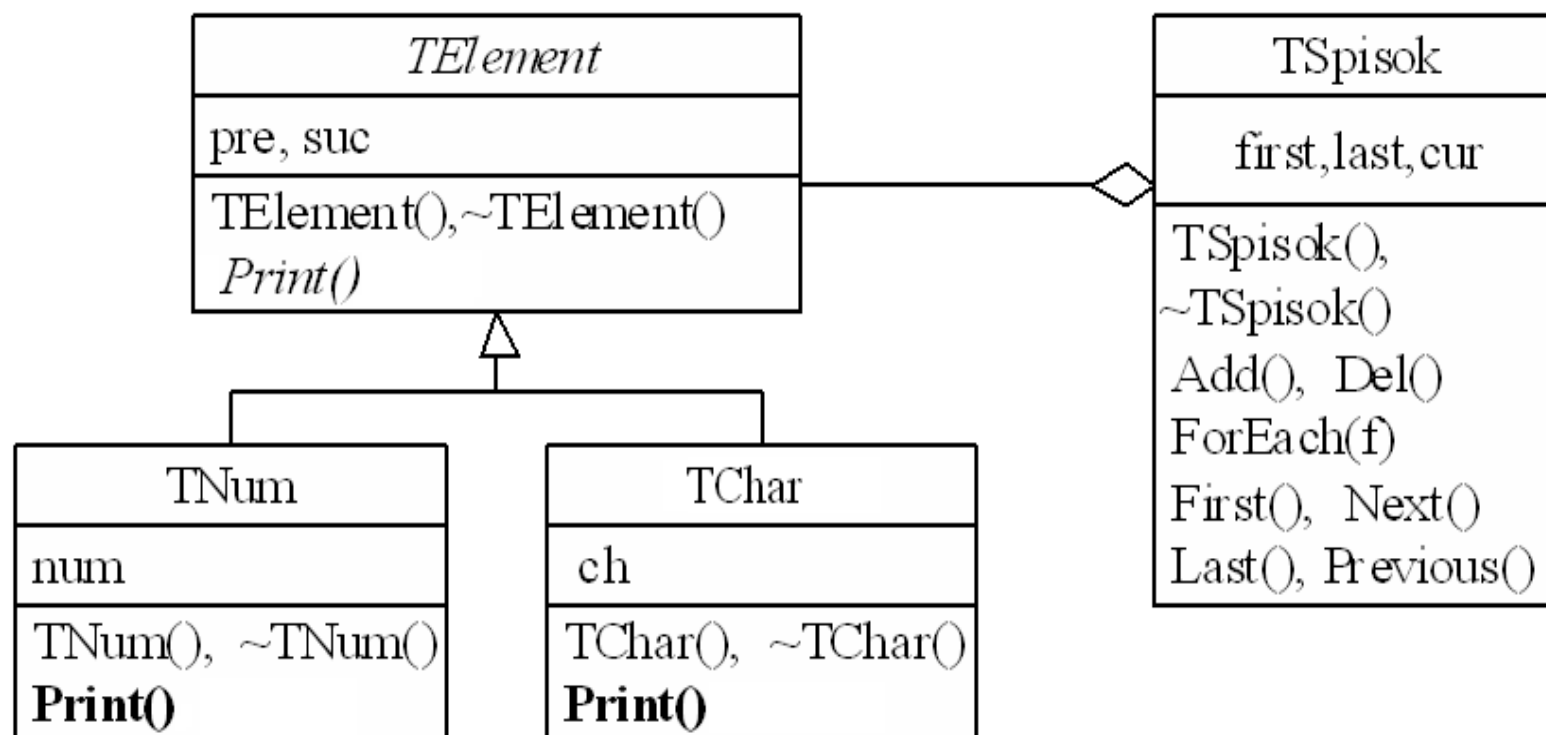
```
  ((TB *)pA)->func();
  reinterpret_cast<TB *>(pA)->func();
  static_cast<TB *>(pA)->func();
  // dynamic_cast<TB *>(pA)->func();
  if (TB *pD=dynamic_cast<TB *>(pA)) pD->func();
  else cout<<"Cast Error"<<endl;
  return 0;}
```

Ошибка!  
Приведение  
не корректно

## 5.7 Контейнер «Двусвязный список» (Ex5\_08)

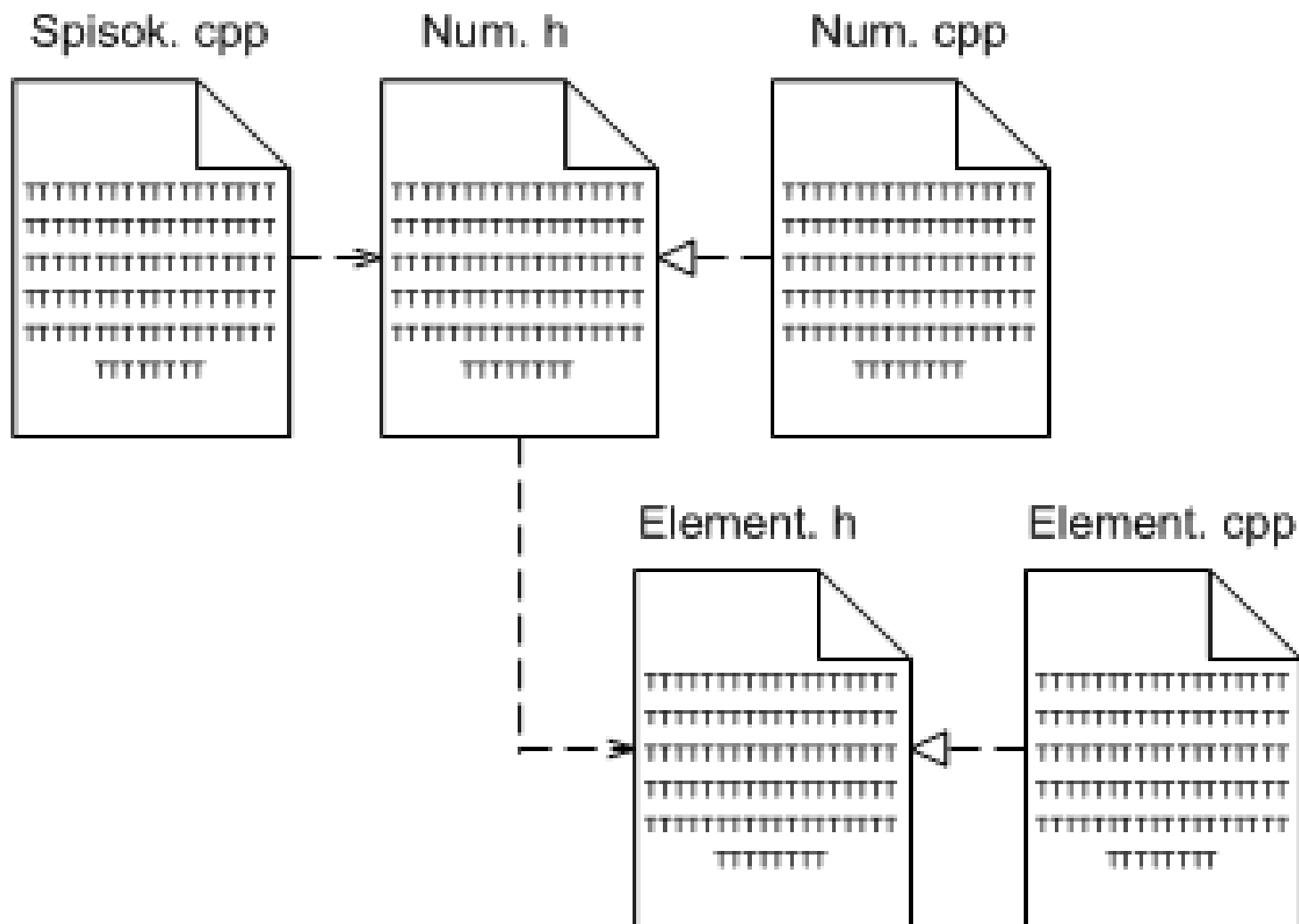


### Диаграмма классов



# Контейнер «Двусвязный список»(2)

## Диаграмма компоновки



# Файл Element.h

```
#include <stdio.h>

class TElement
{
    public:    TElement *pre,*suc;
              TElement() { pre=suc=NULL;}
              virtual ~TElement() { puts("Delete TElement.");}
              virtual void Print()=0;
};

class TSpisok
{
    private: TElement *first,*last,*cur;
    public:  TSpisok() {first=last=cur=NULL;}
            ~TSpisok();
            void Add(TElement *e);
            TElement *Del();
            void ForEach(void (*f)(TElement *e));
            TElement *First(){return cur=first;}
            TElement *Next(){return cur=cur->suc;}
            TElement *Last(){return cur=last;}
            TElement *Previous(){return cur=cur->pre;}
};
```

# Файл Element.cpp

```
#include "Element.h"

TSpisok::~TSpisok()
{
    puts("Delete TSpisok");
    while ((cur=Del())!=NULL) { cur->Print();
                                delete(cur); }
}

void TSpisok::Add(TElement *e)
{
    if (first==NULL) first=last=e;
    else { e->suc=first;
           first->pre=e;
           first=e; }
}
```

## Файл Element.cpp (2)

```
TElement *TSpisok::Del(void)
{
    TElement *temp=last;
    if (last!=NULL)
    {last=last->pre;
      if (last!=NULL) last->suc=NULL;
    }
    if (last==NULL) first=NULL;
    return temp;
}

void TSpisok::ForEach(void (*f) (TElement *e))
{
    cur=first;
    while (cur!=NULL)
    { (*f) (cur);
      cur=cur->suc;
    }
}
```

## Файл Num.h

```
#include "Element.h"
class TNum:public TElement
{ public: int num;
      TNum(int n):TElement(),num(n) {}
      ~TNum() { puts("Delete TNum.");}
      void Print() { printf("%d ",num); }
};
class TChar:public TElement
{ public: char ch;
      TChar(char c):TElement(),ch(c) {}
      ~TChar() { puts("Delete TChar.");}
      void Print() { printf("%c ",ch);}
};
void Show(TElement *e);
```

## Файл Num.cpp

```
#include "Num.h"
void Show(TElement *e)
{ e->Print();}
```

# Тестирующая программа

```
#include "Num.h"
#include <string.h>
#include <stdlib.h>
TSpisok N;
int main(int argc, char* argv[])
{  char str[10];
   int k,i;
   TElement *p;
   while (printf("Input numbers, strings or <end>:"),
          scanf("%s",str), strcmp(str,"end"))
   {  k=atoi(str);
      if (k||(strlen(str)==1 && str[0]=='0'))  p=new TNum(k);
      else p=new TChar(str[0]);
      N.Add(p);
   }
   puts("All list:");
   N.ForEach(Show);
```



## Тестирующая программа(2)

```
p=N.First(); k=0;
while (p!=NULL)
{ if (TNum *q=dynamic_cast<TNum *>(p)) k+=q->num;
// установить создание RTTI (/GR в Project\Settings...)
  p=N.Next();
}
printf("Summa= %d\n",k);
p=N.Last();
i=0;
while (p!=NULL)
{ if (TChar *q=dynamic_cast<TChar *>(p)) str[i++]=q->ch;
  p=N.Previous();
}
str[i]='\0';
printf("String= %s\n",str);
return 0;
}
```