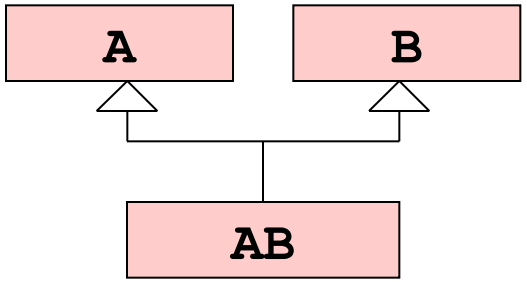
Глава 7. **Более сложные элементы объектной модели C++**

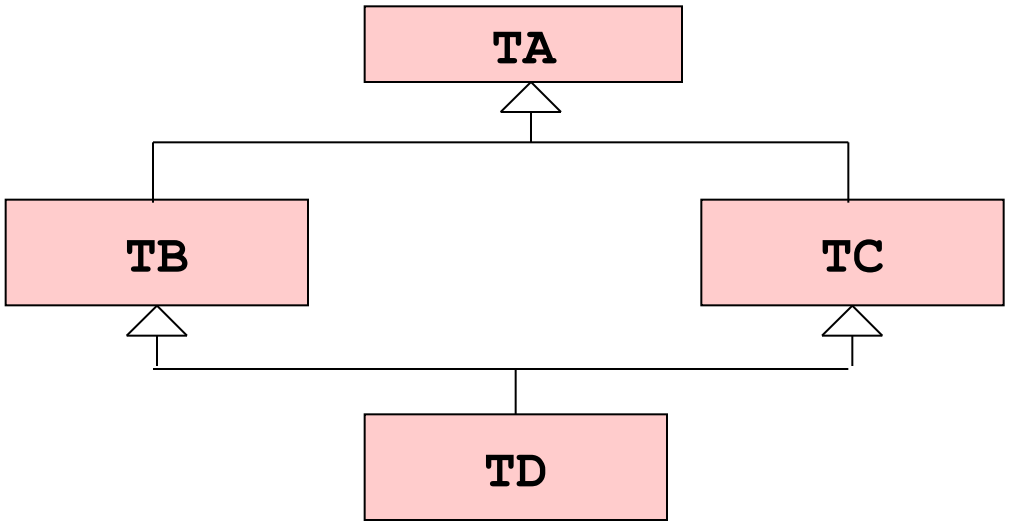
МГТУ им. Н.Э. Баумана
Факультет Информатика и системы управления
Кафедра Компьютерные системы и сети
Лектор: д.т.н., проф.
Иванова Галина Сергеевна

7.1 Множественное наследование (Ex7_08)




```
#include <iostream>
class A
{ protected: int n;
public: A(int an):n(an) {}
};

class B
{ protected: int m;
public: B(int am):m(am) {}
};

class AB :public A, public B
{ int l;
public: AB(int an, int am, int al) :A(an), B(am), l(al) {};
    void pp() {std::cout << n << ' ' << m << ' ' << l; }
};
int main()
{
  AB ab(3, 4, 5);
  ab.pp();
  return 0;
}
```

Виртуальное наследование



Проблема:

дублирование полей прародителя при наследовании от классов, производных от одного базового

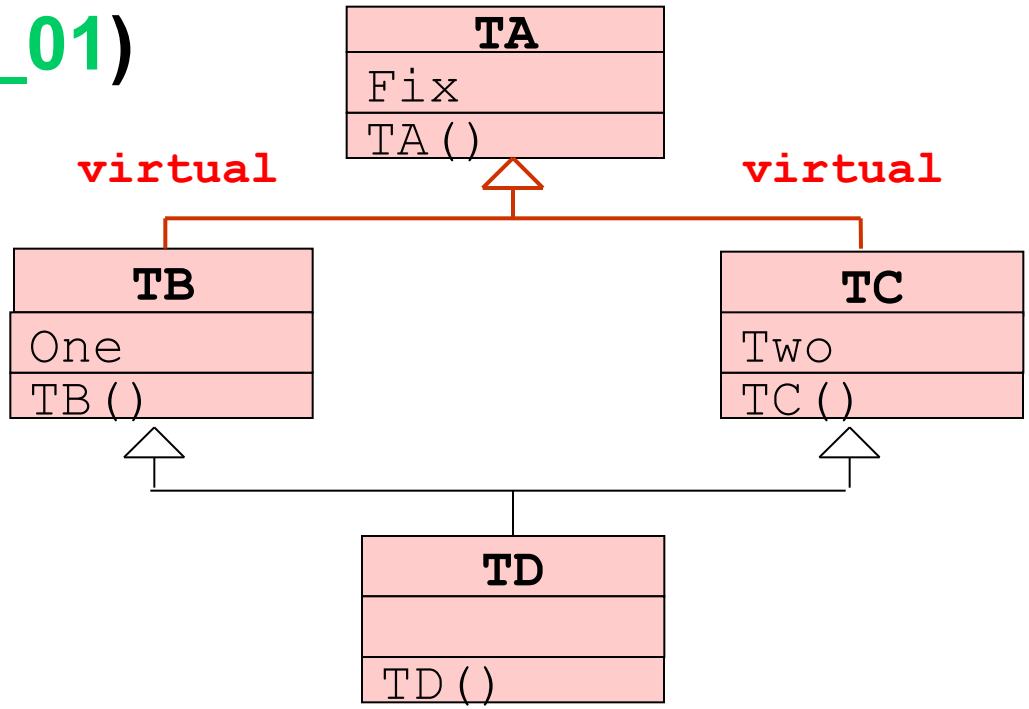
```
class Имя: virtual Вид_наследования Имя_базового_класса
{ ...};
```

Порядок вызовов конструкторов:

- **конструктор виртуально наследуемого базового класса,**
- **конструкторы базовых классов в порядке их перечисления при объявлении производного класса,**
- **конструкторы объектных полей и конструктор производного класса.**

Деструкторы соответственно вызываются в обратном порядке.

Пример множественного виртуального наследования (Ex7_01)




```
#include <iostream>
using namespace std;

class TA
{ protected:    int Fix;
public:
   TA(){ cout << "Inside A\n"; }
   TA(int  fix) :Fix(fix) { cout << "Inside TA int\n"; }
};
class TB :virtual public TA
{ public: int One;
TB(int one):One(one) { cout << "Inside TB\n"; }
};
```

Пример множественного виртуального наследования (2)

```
class TC : virtual public TA
{ public:int Two;
   TC(int two):Two(two) { cout << "Inside TC\n"; }
};
class TD :public TB, public TC
{ public:
   TD(int f,int one,int two) :TA(f),TB(one),TC(two)
   { cout << "Inside TD\n"; }
   void Out() { cout << Fix; }
};
int main()
{ TD Var(10,1,2);
   Var.Out();
   return 0;
}
```

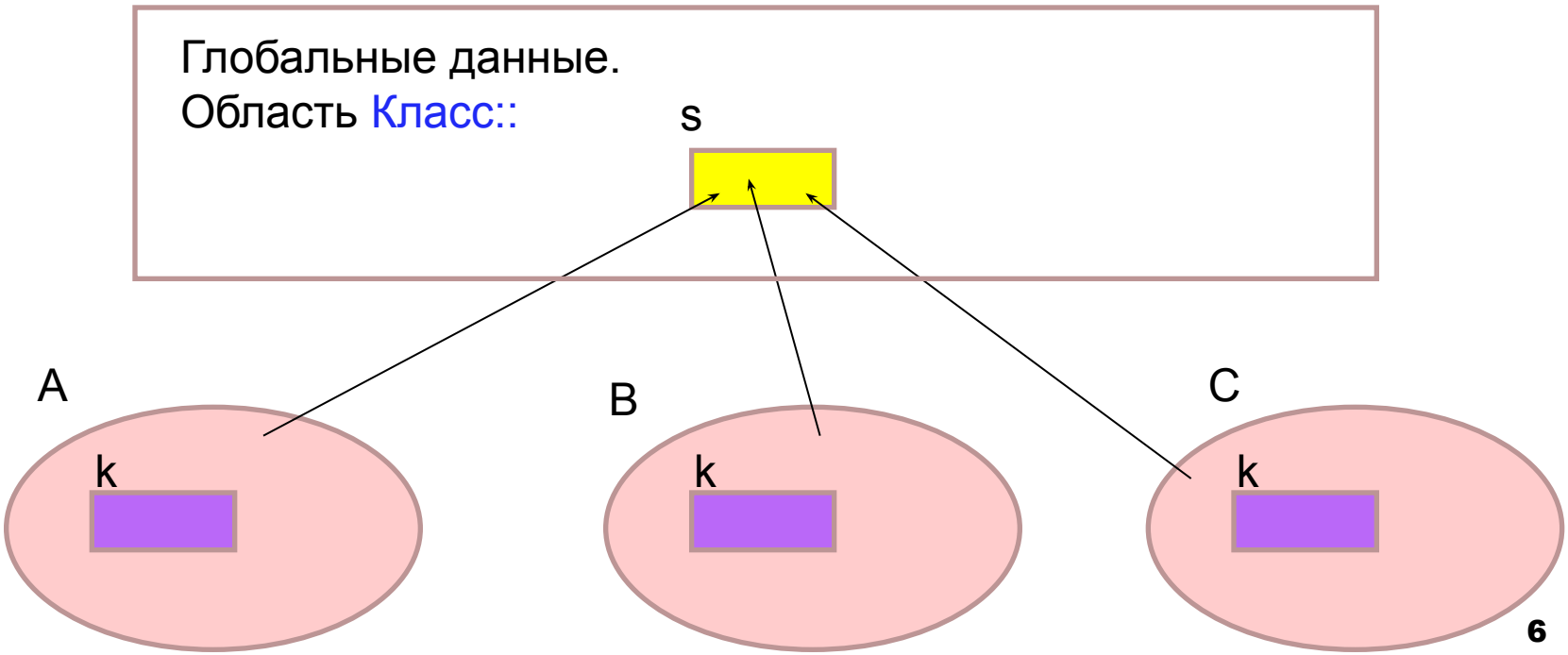
```
Inside TA int
Inside TB
Inside TC
Inside TD
10
```

7.2 Статические компоненты класса

Объявляются с описателем **`static`**

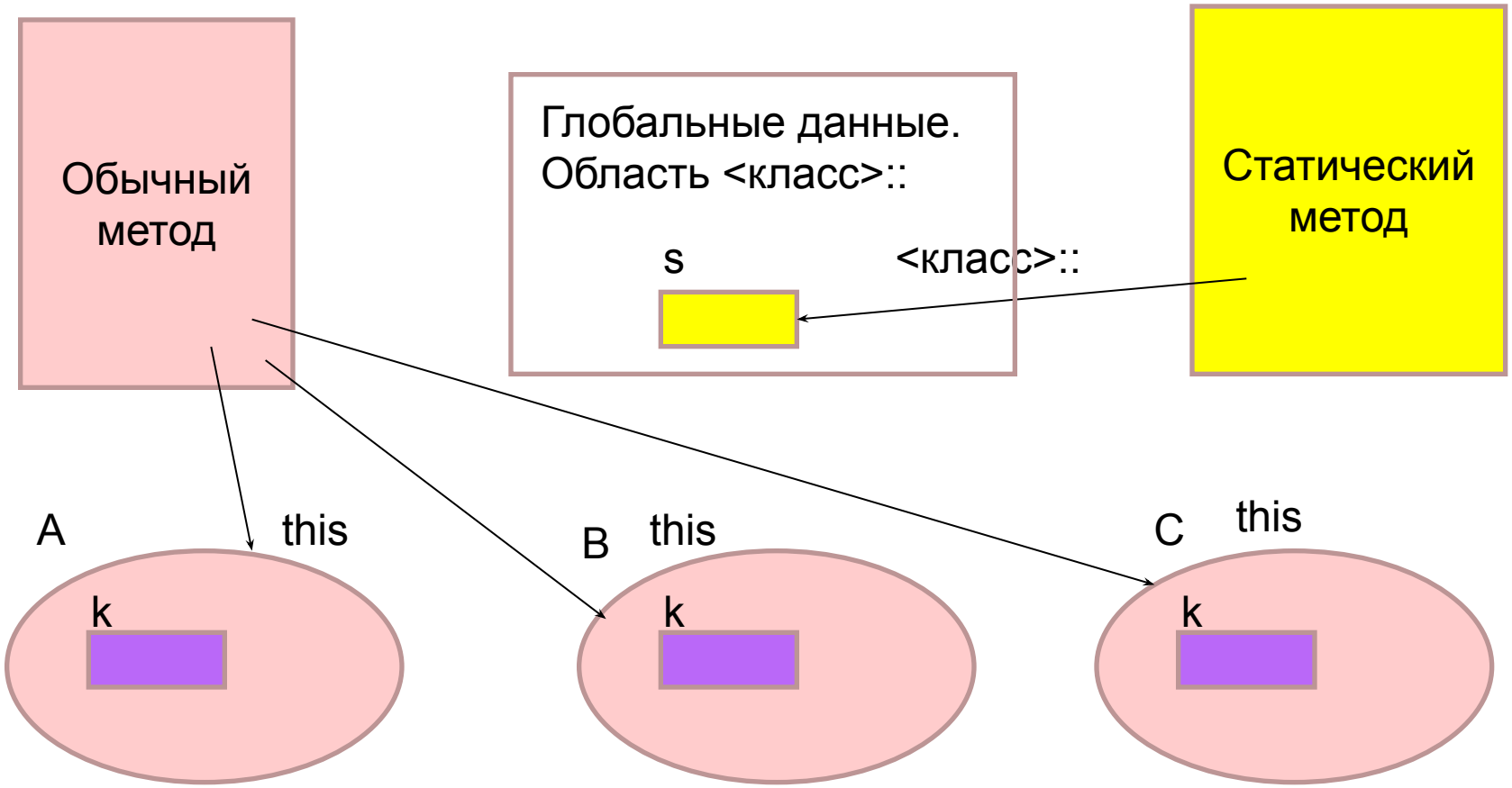
Статические поля:

- являются общими для всех объектов класса;
- существуют даже при отсутствии объектов, в этом случае для доступа к ним используют квалификатор **<класс>::** ;
- инициализация статических полей в определении класса не допустима.



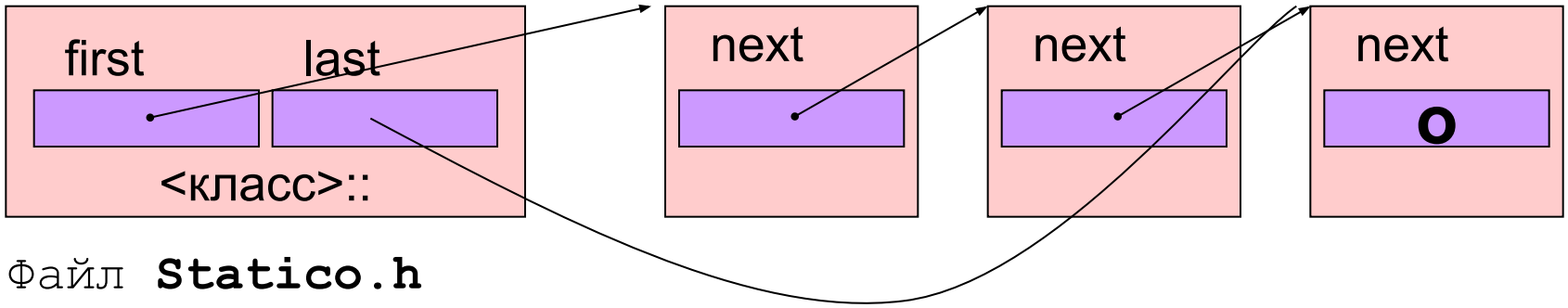
Статические методы класса

Статические методы не получают параметра this и, следовательно, требуют явного указания имени объекта в параметрах при обращении к нестатическим полям.



Статические компоненты класса (Ex7_02)

Пример. Создать список объектов



Файл **Statico.h**

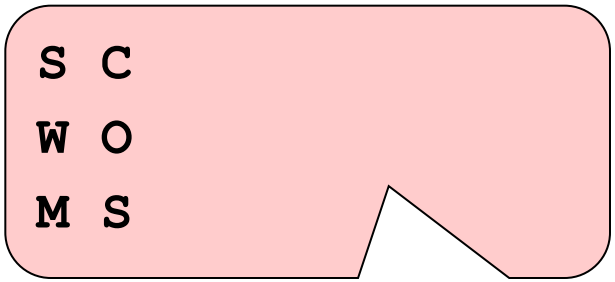
```
#include <stdio.h>
class TPoint
{
public:  char ch1,ch2;
      static TPoint *first, *last;
      TPoint *next;
      TPoint(char ach1,char ach2);
      void Draw(){ printf("%c   %c  \n",ch1,ch2); }
      static void DrawAll();
};
```

Файл Statico.cpp

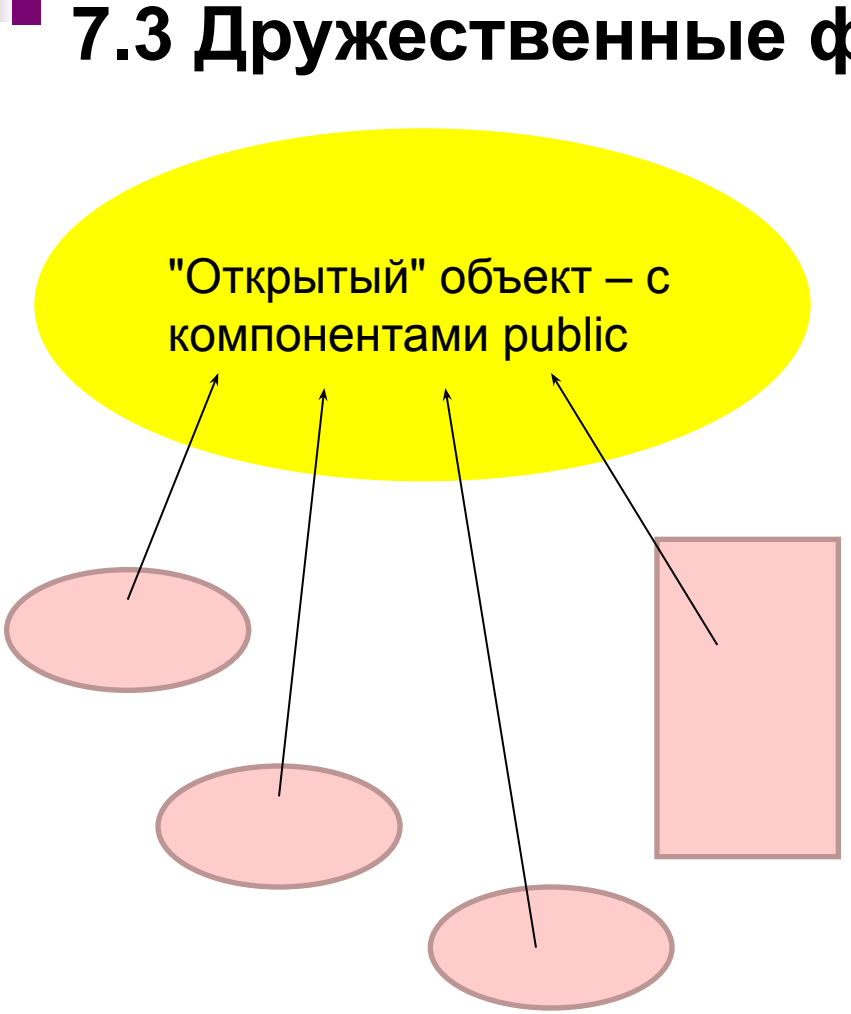
```
#include "statico.h"

TPoint *TPoint::first=NULL,*TPoint::last=NULL;

TPoint::TPoint(char ach1,char ach2)
{
   ch1=ach1; ch2=ach2;   next=NULL;
      if(first==nullptr)first=this;
      else last->next=this;
      last=this;
}
void TPoint::DrawAll()
{
      TPoint *p=first;
      if(p==NULL)return;
      do
      {  p->Draw();  p=p->next;
      }
      while(p!=NULL);
}
```

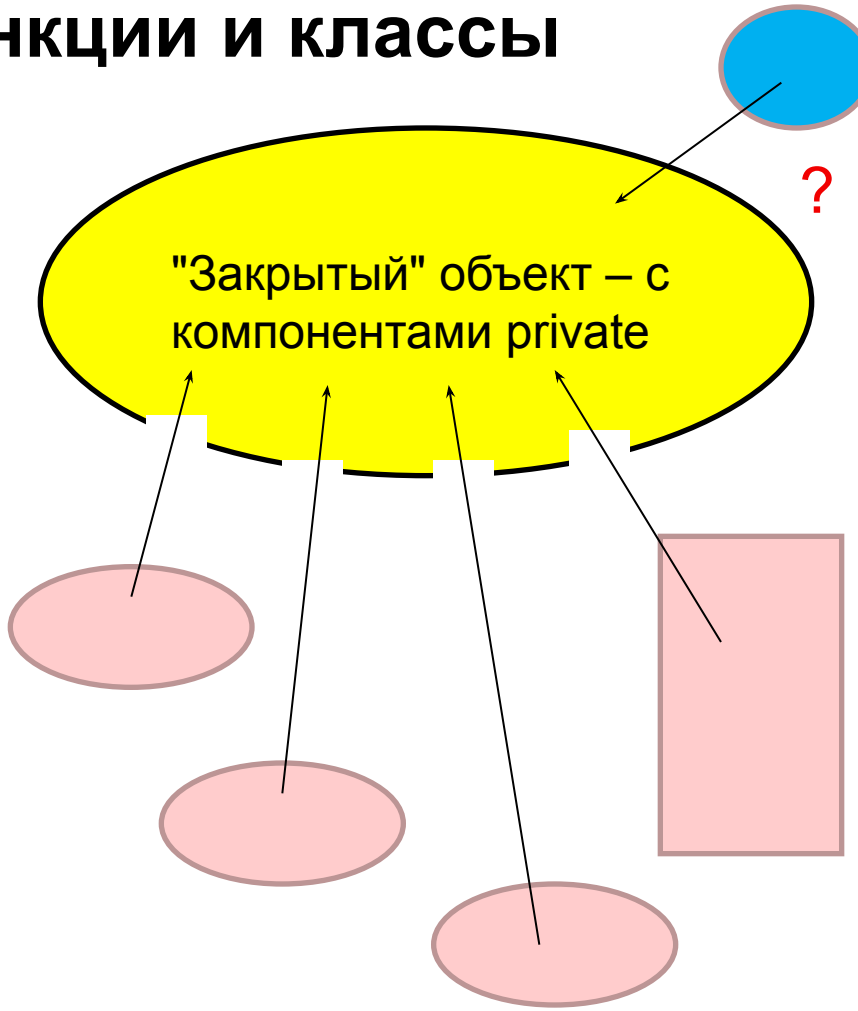
Тестирующая программа

```
#include "statico.h"

int main(int argc, char* argv[])
{
  TPoint A('S','C'),B('W','O'),C('M','S');
  if(TPoint::first!=NULL) TPoint::DrawAll();
  getch();
  return 0;
}
```



7.3 Дружественные функции и классы





Проблемы:

1) велика вероятность "испортить";
2) сложно модифицировать.

Проблема:
как осуществлять
"несанкционированные обращения"?

Дружественные функции и классы (2)

Описываются с описателем **friend**, **что обеспечивает доступ к внутренним компонентам класса**

Пример:

```
class TPoint
    {private: int x,y;
     public:...
     friend void Show(TPoint A); // функция
    };
void Show(TPoint A){cout << A.x << ' ' << A.y;}

int main()
{ TPoint W(2,3);
  Show(W);
  ... }

friend void TLine::Show(TPoint A); // метод

friend class TLine;                // класс
```

7.4 Операции. Переопределение операций

Пример выражения:

```
a = ( c + n * k , b = s / c );
```

где = (+ * , /) – операции, реализуемые функциями-операциями.

Типы функций-операций:

Реализуемая операция

Одноместная операция

Двуместная операция

1. Независимая функция-операция

а) **Тип_результата> operator@(Операнд)**

б) **Тип_результата> operator@(Операнд1,Операнд2)**

Одноместная Операнд - объект

2. Компонентная функция-операция

а) **Тип результата operator@( )**

б) **Тип результата operator@(Операнд2)**

Двуместная Операнд1 - объект

Формы вызова функций-операций

Стандартная форма	Операторная форма
Независимые функции-операции	
operator@(Аргумент)	@Аргумент
`operator++(a);`	`++a;`
operator@(Аргумент1,Аргумент2)	Аргумент1@Аргумент2
`operator+(a,b);`	`a+b;`
Компонентные функции-операции	
Аргумент. operator@( )	@Аргумент
`A.operator++();`	`++A;`
Аргумент1. operator@(Аргумент2)	Аргумент1@Аргумент2
`A.operator+(B);`	`A+B;`

Переопределение операций

1. Можно переопределять только операции, параметры которых – объекты.
2. Не разрешается переопределение:
   - * (разыменование),
   - sizeof,
   - ? : (трехместный выбор),
   - #, ##,
   - :: (пространство имен),
   - <класс>:: (область имен класса).
3. Операции =, [ ], ( ) можно переопределять только в составе класса.
4. При переопределении операций нельзя изменить ее приоритет и ассоциативность.

Пример 1. Класс «Точка» (Ex7_03) Файл Tpoint.h

```
#pragma once
#include <iostream>
using namespace std;
class TPoint{
   private:       float x,y;
   public:
      TPoint(float ax,float ay):x(ax),y(ay)
      {cout<<"Constructor\n";}
      TPoint(){cout<<"Constructor without parameters\n";}
      TPoint(TPoint &p){ cout<<"Copy Constructor\n";
     x=p.x; y=p.y;
      }
      ~TPoint(){cout<<"Destructor\n";}
      void Out(void)     { cout<<"\n{"<<x<<","<<y<<"}\n"; }
      TPoint& operator+=(TPoint &p);  // a+=b;
      TPoint operator+(TPoint &p);    // a+b;
      TPoint& operator=(TPoint const &p);   // a=b;
};
```

Файл Tpoint.cpp

```
#include  "TPoint.h"


TPoint& TPoint::operator+=(TPoint &p)
{
      x+=p.x; y+=p.y;      cout<<"operator+=\n";
      return *this;
}
 TPoint TPoint::operator+(TPoint &p)
 {
      TPoint pp(x,y);      cout<<"operator+\n";
      return pp+=p;
 }
 TPoint& TPoint::operator=(TPoint const &p)
 {
      x=p.x; y=p.y;        cout<<"operator=\n";
      return *this;
 }
```

Тестирующая программа

```
#include "TPoint.h"
int main()
{    TPoint p(2,3),q(4,5),r(7,8);
     p+=r;
     p.Out();
     q=p+r;
     q.Out();
     return 0;
}
```

Constructor
Constructor
Constructor

Operator +=

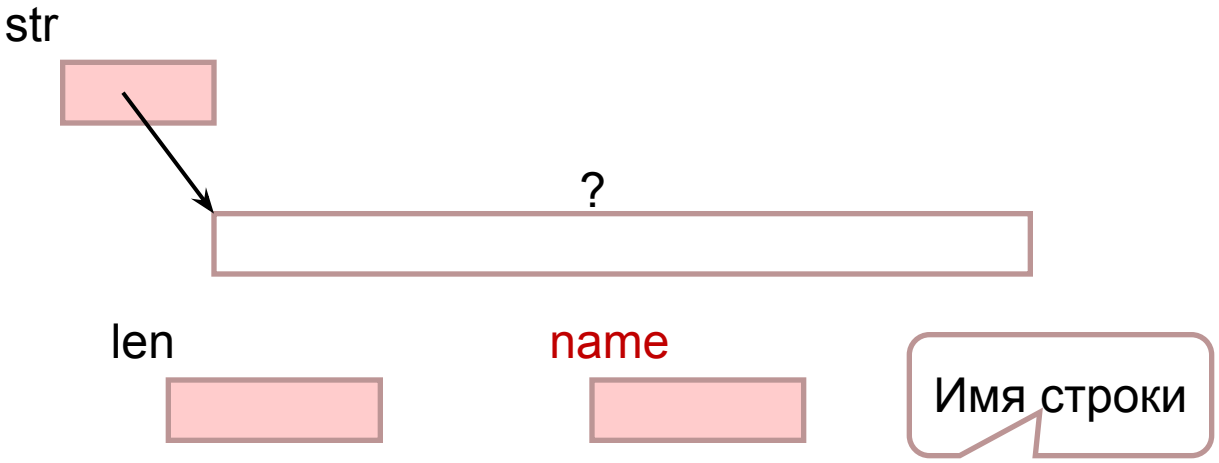
Constructor (pp)
Operator +
Operator +=
Copy constructor
Destructor (pp)
Operator =
Destructor

TPoint pp(x,y);
cout<<"operator+\n";
pp+=p;
return
}

x+=p.x; y+=p.y;
cout<<"+=\n";
return *this;

Destructor
Destructor
Destructor

Пример 2. Класс «Строка»(Ex7_04).



String
char *str int len char name
String() ~String() print() Length() operator[]() operator+() operator=()

Перечень методов:

- конструктор инициализированной строки;
- конструктор пустой строки указанной длины;
- копирующий конструктор;
- деструктор для освобождения памяти;
- метод вывода на экран строки;
- метод, возвращающий длину;
- метод доступа к символу строки по номеру;
- метод-оператор слияния строк;
- метод-оператор добавления символа к строке;
- метод-оператор присваивания;
- …

Файл S.h:

```
#pragma once
#include <string.h>
#include <iostream>
using namespace std;
class String
 { private:      char *str,name;          int  len;
   public:
      String(const char *vs,char Name);
      String(int Len,char Name);
      String(const String &S);
      ~String();
      int Length() const
      {
         return len;
      }
```

Файл S.h (2):

```
    void print() const
    {
        std::cout << "Str: "<<name<<":";
        std::cout << str;
        std::cout << " Length: " << len << endl;
    }
    char operator[](int n)
    {
        return ((n>=0)&&(n<len)) ? str[n] : '\0';
    }
    String operator+(const String &A);
    String operator+(char c);
    String& operator=(const String &S);
};
```

Файл S.cpp

```
#include "s.h"
String::String(int Len,char Name)
{
    len=Len;
    str=new char[len+1];
    str[0]='\0'; name=Name;
    cout<<"Constructor with length "<<name<<"\n";
}
String::String(const char *vs,char Name)
{
    len=strlen(vs);
    str=new char[len+1];
    strcpy(str,vs);  name=Name;
    cout<<"Constructor "<<name<<"\n";
}
```

Файл S.cpp (2)

```
String::String(const String &S)
{
    len=S.Length();
    str=new char[len+1];
    strcpy(str,S.str);
    name='K';
    cout<<"Copy from "<< S.name <<" to "<<name<<"\n";
}

String::~String()
{
   delete [] str;
   cout << "Destructor " << name << "\n";
}
```

Файл S.cpp (3)

```
String  String::operator+(const String &A)
{
    cout << "Operation +" << "\n";
    int j=len+A.Length();
    String S(j,'S');
    strcpy(S.str,str);
    strcat(S.str,A.str);
    cout<< "Operation +" << "\n";
    return S;
}
```

Файл S.cpp (3)

```
String  String::operator+(char c)
{
    cout << "Operation +c" << "\n";
    int j=len+1;
    String S(j,'Q');
    strcpy(S.str,str);
    S.str[len]=c;
    S.str[len+1]='\0';
    cout << "Operation +c" << "\n";
    return S;
}
```

Файл S.cpp (3)

```
String& String::operator=(const String &S)
{
   cout << "Operation =" << "\n";
   len=S.Length();
   if (str!=nullptr) delete [] str;
   str=new char[len+1];
   strcpy(str,S.str);
   cout << "Operation =" << "\n";
   return *this;
}
```

Тестирующая программа

```
#include "S.h"

int main()
{
    String A("ABC",'A'),B("DEF",'B'),C(6,'C');
    C.print();
    C=A+B;
    C.print();
     C=C+'a';
    C.print();
    return 0;
}
```

Выполнение операций

	Выполняемые операторы	Результаты

C=A+B;	String operator+(const String &A)
	{ cout<<"Operation +"<<"\n";
	int j=len+A.Length();
	String S(j,'S');
	strcpy(S.str,str);
	strcat(S.str,A.str);
	cout<<"Operation +"<<"\n";
	return S;
	}
	String& operator=(const String &S)
	{ cout<<"Operation ="<<"\n";
	len=S.Length();
	if (str!=NULL) delete[]str;
	str=new char[len+1];
	strcpy(str,S.str);
	cout<<"Operation ="<<"\n";
	return *this;
	}
		Operation +
		Constructor with length S
		Operation +
		Copy from S to K
		Destructor S
		Operation =
		Operation =
		Destructor K

7.5 Конструктор и оператор перемещения. Правило ТРЕХ и правило Пяти.

Если класс или структура определяет один из следующих методов, то они должны явным образом определить все три метода:

- копирующий конструктор;
- оператор присваивания;
- деструктор – если не используются "умные указатели" (см. далее).

Эти три метода являются особыми, автоматически создаваемыми компилятором в случае отсутствия их явного объявления программистом. Если один из них должен быть определен программистом, то это означает, что версия, сгенерированная компилятором, не удовлетворяет потребностям класса в одном случае и, вероятно, не удовлетворит в остальных случаях.

В настоящее время правило преобразовано в правило Большой пятерки, т.к. добавлены еще конструктор перемещения и оператор присваивания перемещением, используются для организации смены владельца фрагмента оперативной памяти.

Конструктор перемещения, оператор присваивания перемещением и функция std::move()

Конструктор перемещения вызывается, если параметр – временный объект (r-value):

Имя_класса(Имя_класса && Имя_объекта) {…}

Оператор присваивания перемещением вызывается, если присваиваемый объект – временный (r-value):

Имя_класса Имя_класса::operator=

(Имя_класса && Имя_объекта) {…}

Если объект имеет физический адрес(l-value), т.е. не является временным (r-value), а требуется организовать вызов конструктора перемещения или оператор присваивания перемещением, то используют преобразование объекта с помощью *функции move()*:

Имя_класса && `std::move`(Имя_класса & Имя_объекта);

Пример. Функция std::move() (Ex7_09)

```
#include <iostream>
 using namespace std;
void swap(string& x, string& y){
   string temp{x};
   x = y;
   y = temp;
}
int main() {
   string x{"Anton"}, y{"Ivan"};
   cout << "x: " << x << '\n';
   cout << "y: " << y << '\n';
   swap(x, y);
   cout << "x: " << x << '\n';
   cout << "y: " << y << '\n';
    return 0;
}
```

Замена на:

```
string temp(move(x));
x = move(y);
y = move(temp);
```

позволит избежать трех копирований

Пример. Правило Пяти (Ex7_10)

```
#include <iostream>
  using namespace std;
class Number{
 private: int * pnum;
 public:
    Number(int Num):pnum(new int(Num)){
      cout<<"New, Constructor"<<endl;
    }
    Number(const Number &R):pnum(new int(*R.pnum)){
      cout<<"New, Constructor copy"<<endl;
    }
    Number():pnum(nullptr){}
    Number& operator=(const Number &R){
      if (pnum!=nullptr){delete pnum;cout<<"Free"<<endl;}
      pnum=new int(*R.pnum);
      cout<<"New Operator= copy"<<endl;
      return *this;
    }
```

Конструктор

Конструктор копирующий

Конструктор без параметров

Оператор присваивания

Пример. Правило Пяти (2)

```
  ~Number(){
    if (pnum!=nullptr){delete pnum;cout<<"Free"<<endl;}
   cout<<"Destructor"<<endl;
  }
  Number(Number&& R):pnum(R.pnum){
    R.pnum=nullptr; cout<<"Constructor move"<<endl;
  }
  Number& operator=(Number&& R){
    if (pnum!=nullptr){
      delete pnum; cout<<"Free move"<<endl;
    }
    pnum=R.pnum;
    R.pnum=nullptr; cout<<"Operator= move"<<endl;
    return *this;
  }
};
```

Деструктор

Конструктор перемещения

Оператор присваивания перемещением

Пример. Правило Пяти (3)

```
Number f(int a,int b) {
  Number temp(a+b);
  return Number(move(temp));
}
int main() {
  Number A(5);
  Number B(A);
  Number C(move(A));
  Number D(6);
  D=move(A);
  Number F=f(6,7);
  return 0;
}
```

New Constructor

New Constructor copy

Constructor move

New Constructor

Free move, Operator= move

New Constructor
Constructor move
Destructor

Free, Destructor
Destructor
Free, Destructor
Free, Destructor
Destructor

Распределение и освобождение памяти

Текст программы	Тексты методов	Вывод

Number A(5);
Number B(A);
Number C(move(A));
	Number(Number&& R):
	pnum(R.pnum){
	R.pnum=nullptr;
	cout<<"Constructor move;
	}
		New Constructor
		New Constructor copy
		Объект A потерял поле
		Constructor move

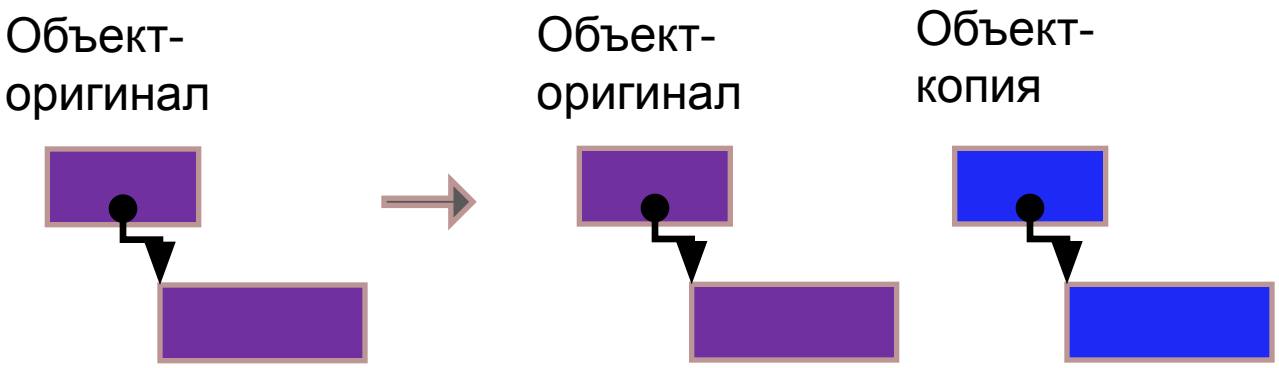
Number D(6);		New Constructor
D=move(A);	Объект D теперь без поля	Free move (D) Operator= move

Number F=f(6,7);	Number f(int a,int b) {
	Number temp(a+b);
	return Number(move(temp));
	}
		New Constructor
		Constructor move
		Destructor

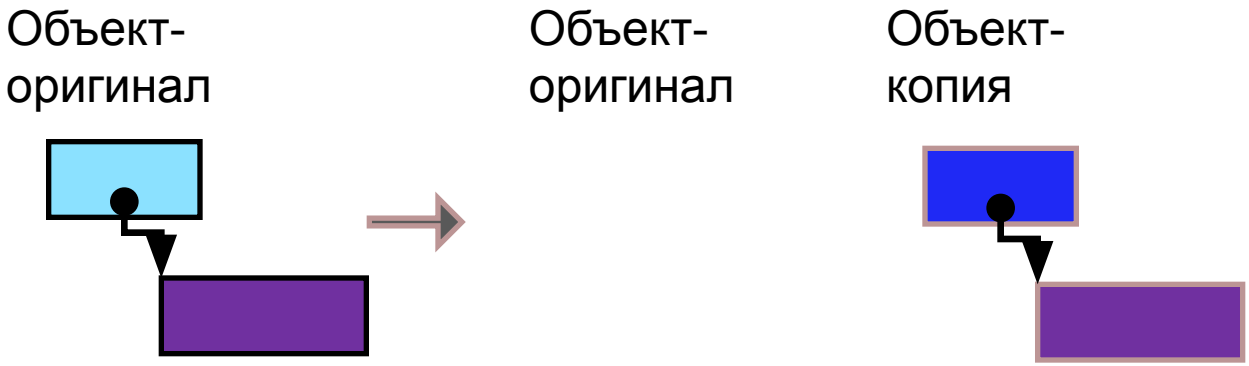
return 0; }		Free, Destructor (F) Destructor (D) Free, Destructor (C) Free, Destructor (B) Destructor (A)

Различие копирования и перемещения

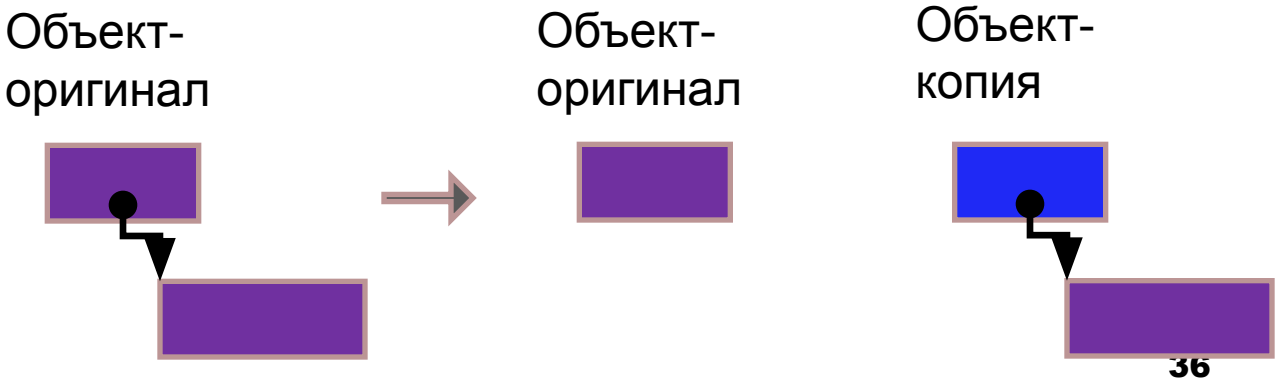
- копирующий конструктор с параметром lvalue



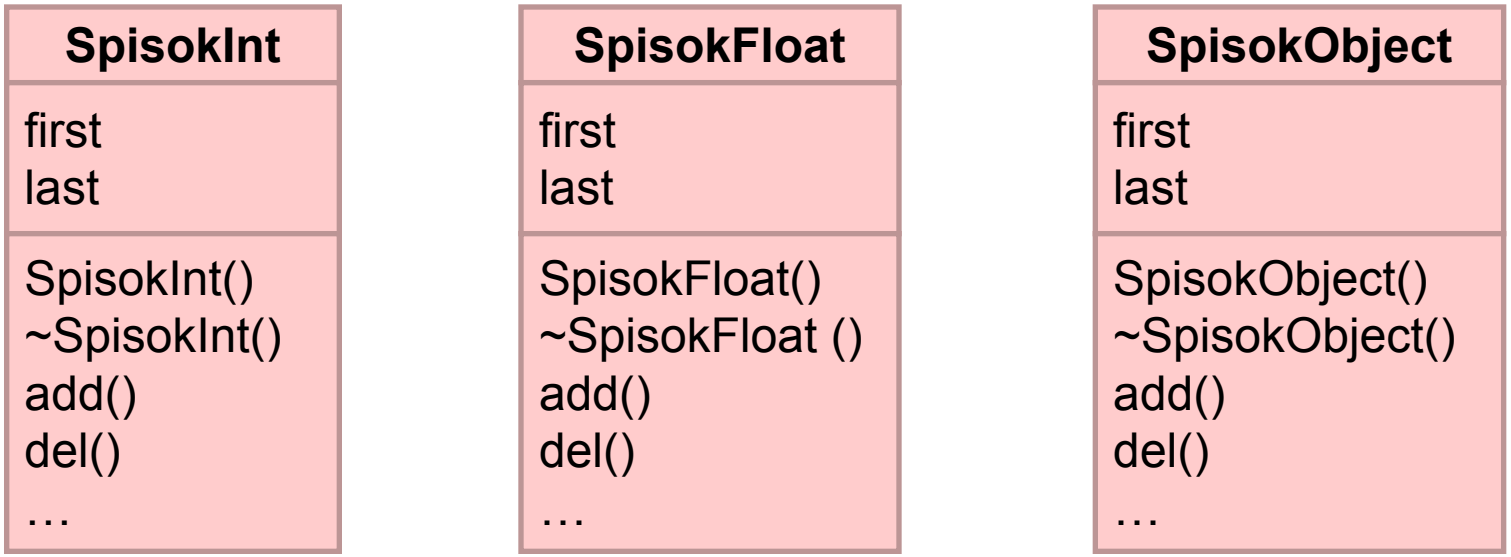
- конструктор перемещения с параметром rvalue



- конструктор перемещения с параметром move(lvalue)



7.6 Параметризованные классы (шаблоны)



Шаблон класса – обобщенное описание класса, содержащее параметры, позволяющие задавать **типы** используемых полей или других данных.

Формат описания шаблона класса

```
template Список_параметров Описание_класса
```

Формат объявления объектов:

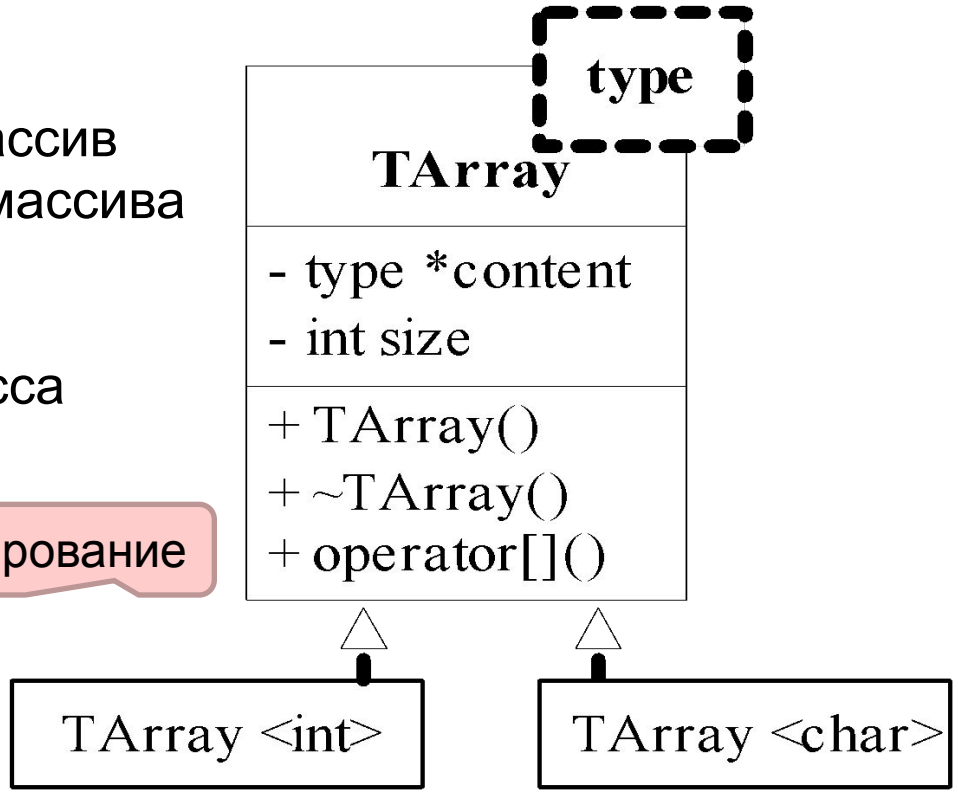
```
Имя_класса Список_аргументов
    Имя_объекта(Параметры_конструктора)
```

Параметр шаблона

Пример (**Ex7_05**).
Создать шаблон Динамический массив и использовать его для создания массива целых чисел и массива символов.

Операция создания описания класса из шаблона называется *инстанцированием*.

Инстанцирование



Описание шаблона

```
#include <iostream>
using namespase std;
template <class type>
class TArray
{   type * content;
    int size;
 public:
    TArray(int asize)
    { content = new type [size=asize];}
    ~TArray (){delete [] content;}
    type & operator[] (int x)
    { if ((x < 0)||(x >= size))
          { cerr << "Index Error"; x=0; }
      return content[x];
    }
};
```

Тестирующая программа

```
int main()
{
    int i;
    TArray<int> int_a(5);
    TArray<char> char_a(5);
    for (i=0;i<5;i++)
      {
          int_a[i]=i*3+2*(i+1);
          char_a[i]='A'+i;
      }
    cout << "Two arrays: "<< endl;
    for (i=0;i<5;i++)
      cout << int_a[i] << char_a[i] << endl;
    return 0;
}
```

Инстанцирование – создание экземпляра класса по шаблону

7.7 Параметризованные функции

Шаблон функции – обобщенное описание функции, которая может вызываться для данных разных типов.

Формат описания шаблона функции:

`template Список_параметров Описание_функции`

`Пример.` Шаблон функции определения максимального (Ex7_06)

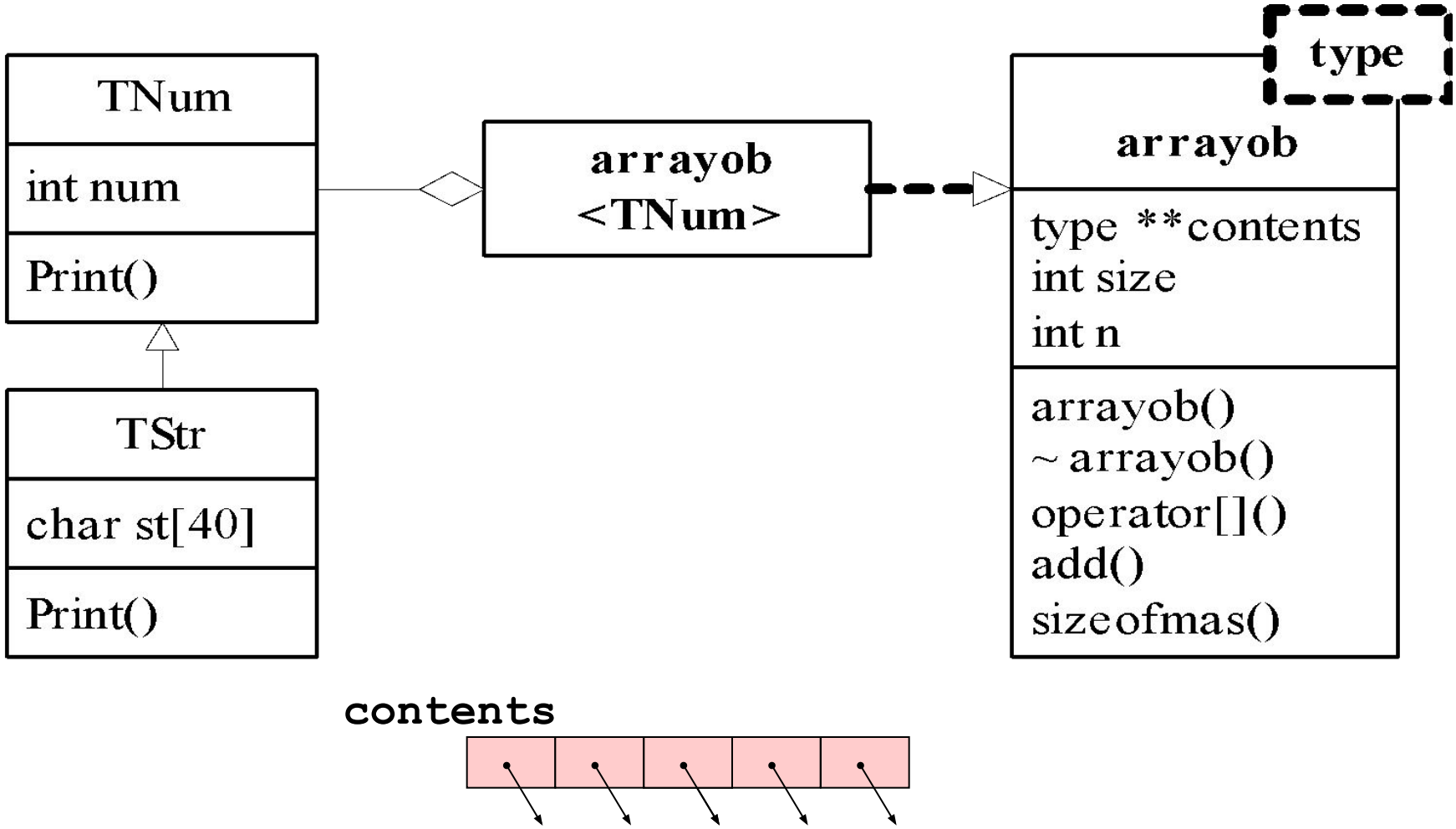
```
#include <string.h>
#include <iostream>
using namespace std;
template <class T>
      T maxx(T x, T y)
         { return(x > y)? x : y; }
char * maxx(char * x, char * y)
         { return strcmp(x,y) > 0? x:y;}
```

Перегрузка шаблона функции

Тестирующая программа

```
int main()
{   int a=1,b=2;
    char c='a', d='m';
    float e=123, f=456;
    double p=234.567,t=789.23;
    char str1[]="AVERO", str2[]="AVIER";

    cout << "Integer max=    " << maxx(a,b)<< endl;
    cout << "Character max= " << maxx(c,d)<< endl;
    cout << "Float max=      " << maxx(e,f)<< endl;
    cout << "Double max=     " << maxx(p,t)<< endl;
    cout << "String max=     " << maxx(str,str2)<< endl;
    return 0;
}
```

7.8 Контейнер на основе шаблона (Ex7_07)

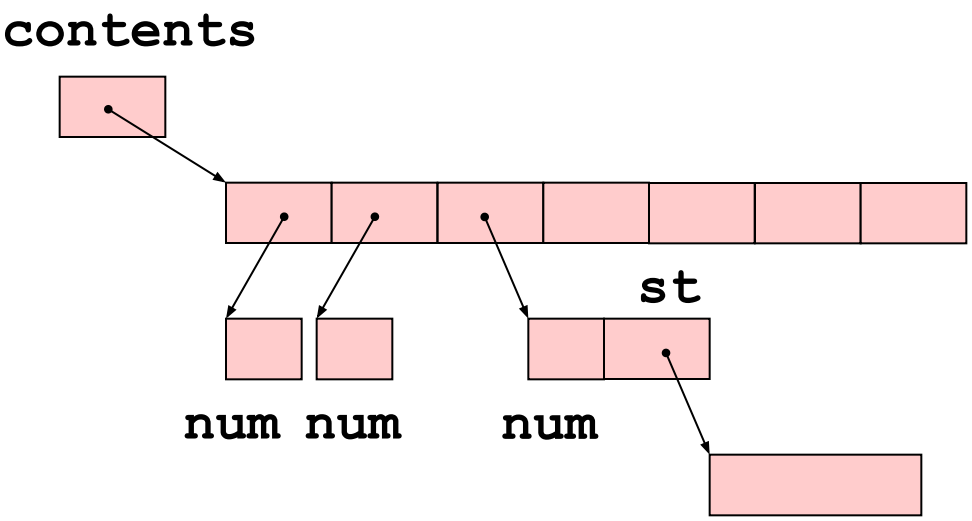


Объявление шаблона класса в файле A.h

```
#include <iostream>
using namespace std;
template <class type>
class arrayob
{  type **contents;  int size;  int n;
 public:
  arrayob(int number){contents=new type *[size=number]; n=0;}
  ~arrayob ();
  int sizeofmas(){return n;}
  void add(type *p) { if(n == size)
      std::cerr<<"Out of range";
     else contents[n++]=p;
  }
  type & operator [] (int x)
  { if  ((x<0)||(x>=n))
        { std::cerr <<"Error "<<x<<endl;x=0;}
    return *contents[x]; }
};
```

Объявление шаблона функции

```
template <class type>
   arrayob <type>::~arrayob ()
   {  for(int i=0;i<n;i++) delete contents[i];
      delete [] contents;
   }
```



Описание классов элементов (файл N.h)

```
#include <string.h>
class TNum
{ public:int num;
   TNum(int n):num(n) {}
   virtual ~TNum();
   virtual void Print();
};
class TStr:public TNum
{ public: char *st;
   TStr(char *s):TNum(strlen(s))
   {
       st=new char[num+1];
       strcpy(st,s);
   }
   ~TStr() override;
   void Print() override;
};
```

Описание классов элементов (файл N.cpp)

```
#include <iostream>
using namespace std;
#include "N.h"

TNum::~TNum() {cout<<"Destructor TNum "<<endl;}
void TNum::Print() { cout<< num << " " << endl; }
};
TStr::~TStr(){
    cout<<"Destructor TStr."; delete [] st;
}
void TStr::Print() {
    TNum::Print();
    cout << st << " " << endl;
}
```

Тестирующая программа

```
#include "A.h"
#include "N.h"
int main() {
  arrayob <TNum>  ob_a(5);
  int n,i;     char S[10];
  for(i=0;i<5;i++)
   {  cout << "Input number or string: ";
      cin >> S;
      n=atoi(S);
      if (n == 0 && S[0] == '0' || n !=0 )
           ob_a.add(new TNum(n));
      else ob_a.add(new TStr(S));}
  cout << " Contents of array" << '\n';
  for (i=0;i<ob_a.sizeofmas();i++) ob_a[i].Print();
  return 0;
}
```