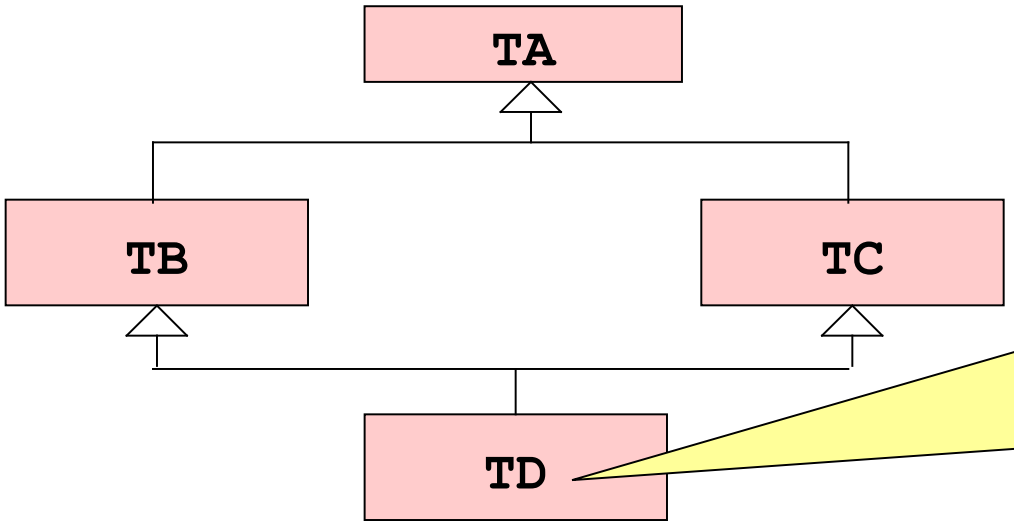
Глава 7. **Более сложные элементы объектной модели C++**

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 и виртуальное наследование



Проблема:

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

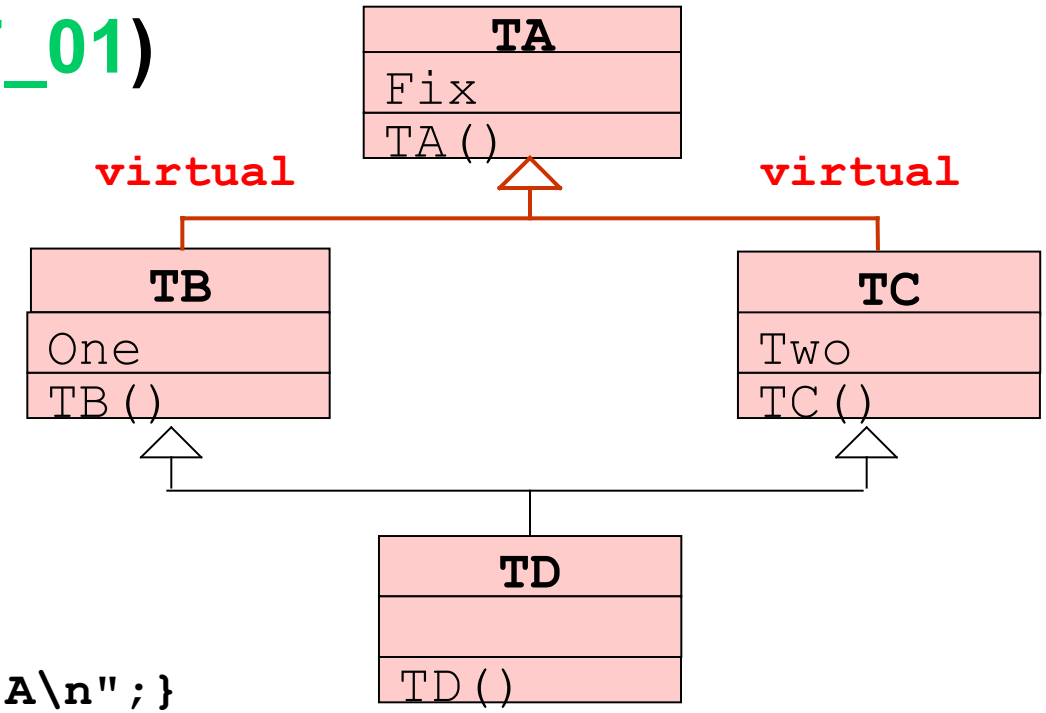
```
class <Имя>: virtual <Вид наследования> <Имя базового класса>
{ ...};
```

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(void){ cout<<"Inside TB\n";}
};
```

Пример множественного виртуального наследования (2)

```
class TC: virtual private TA
{ public:    int Two;
      TC() { cout<<"Inside TC\n";}
};
class TD:public TB,public TC
{ public:
      TD(int fix):TA(fix){cout<<"Inside TD\n";}
      void Out( ) {cout<<Fix; }
};
int main()
{     TD Var(10);
      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 и существуют даже при отсутствии объектов. Инициализация статических полей в определении класса не допустима.

Статические методы не получают параметра this и,

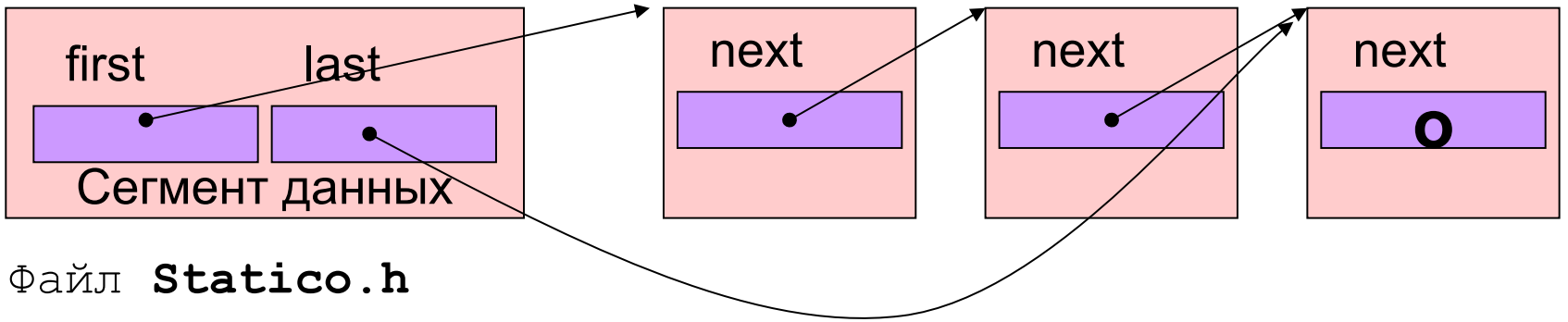
следовательно, не могут без явного указания объекта в

параметрах обращаться к нестатическим полям.

Для доступа к статическим компонентам вне компонентных функций используют квалификатор **<класс>::**

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==NULL)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;
}
```

```
S C
W O
M S
```

7.3 Дружественные функции и классы

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 Переопределение операций
Операции

Типы функций-операций:

1. Независимая функция-операция

а) **<Тип результата> operator@(<Операнд>)**

б) **<Тип результата> operator@(<Операнд1>,<Операнд2>)**

2. Компонентная функция-операция

а) **<Тип результата> operator@( )** // Операнд = Объект

б) **<Тип результата > operator@(<Операнд2>)** // Операнд1 = Объект

Формы вызова

а) стандартная	б) операторная
operator@(<Арг>)	**@<Арг>**
operator@(<Арг1>,<Арг2>)	**<Арг1>@<Арг2>**
<Арг>. operator@( )	**@<Арг>**
<Арг1>. operator@(<Арг2>)	**<Арг1>@<Арг2>**

Переопределение операций

1. Можно переопределять только операции, параметры которых – объекты.
2. Не разрешается переопределение *, sizeof, ? :, #, ##, ::, Class::.
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(int argc, char* argv[])
{    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). Файл S.h:

```
#pragma once
#include <string.h>
#include <iostream>
using namespace std;
class String
 { private:     char *str,name;        int  len;
   public:  String(int Len,char Name);
      String(const char *vs,char Name);
      String(String &S);
      ~String();
      int Length() const {return len;}
      char operator[](int n)
      {return ((n>=0)&&(n<len))?str[n]:'\0';}
      void print() { std::cout<<"Str: "<<name<<":";
          std::cout<<str<<" Length: "<<len<<endl; }
      String  operator+(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(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+(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  String::operator+(char c)
{
    int j=len+1;   cout << "Operation +c" << "\n";
    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!=NULL) delete [] str;
   str=new char[len+1];
   strcpy(str,S.str);
   cout << "Operation =" << "\n";
   return *this;
}
```

Тестирующая программа

```
#include "s.h"

int main(int argc, char* argv[])
{
    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+(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 Параметризованные классы (шаблоны)

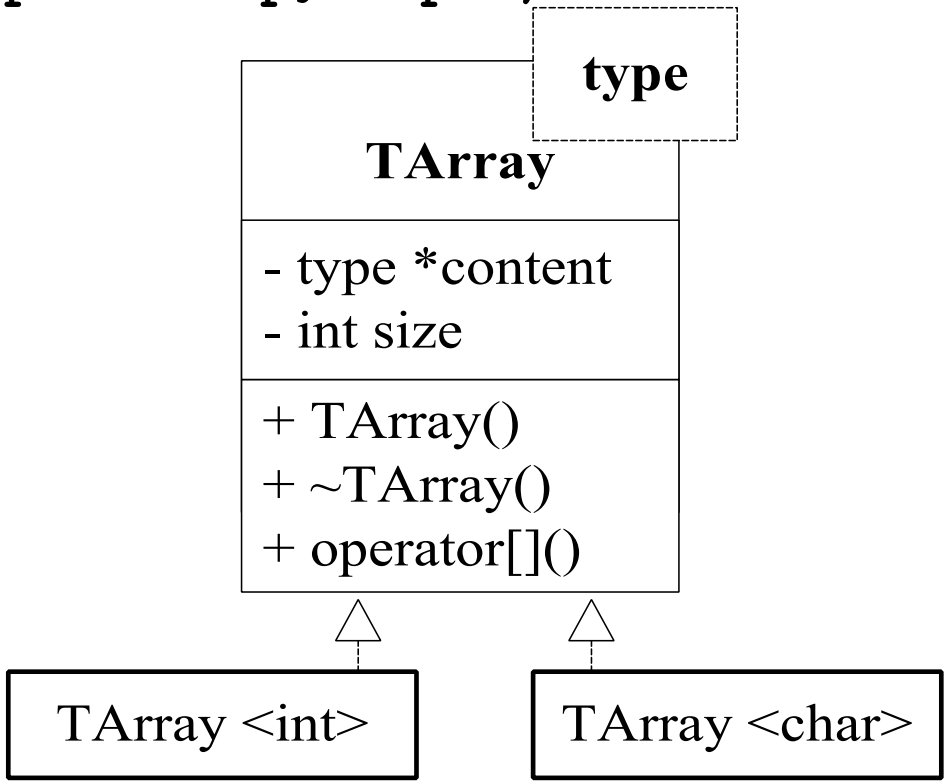
Формат описания шаблона класса:

```
template <Список параметров><Описание класса>
```

Формат объявления объектов:

```
<Имя класса> <Список аргументов>
      <Имя объекта>(<Параметры конструктора>)
```

Пример. Динамический массив (**Ex7_05**)



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 argc, char* argv[])
{
    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

7.6 Параметризованные функции

Формат описания шаблона функции:

template <Список параметров><Описание функции>

Пример. Шаблон функции определения максимального (Ex7_06)

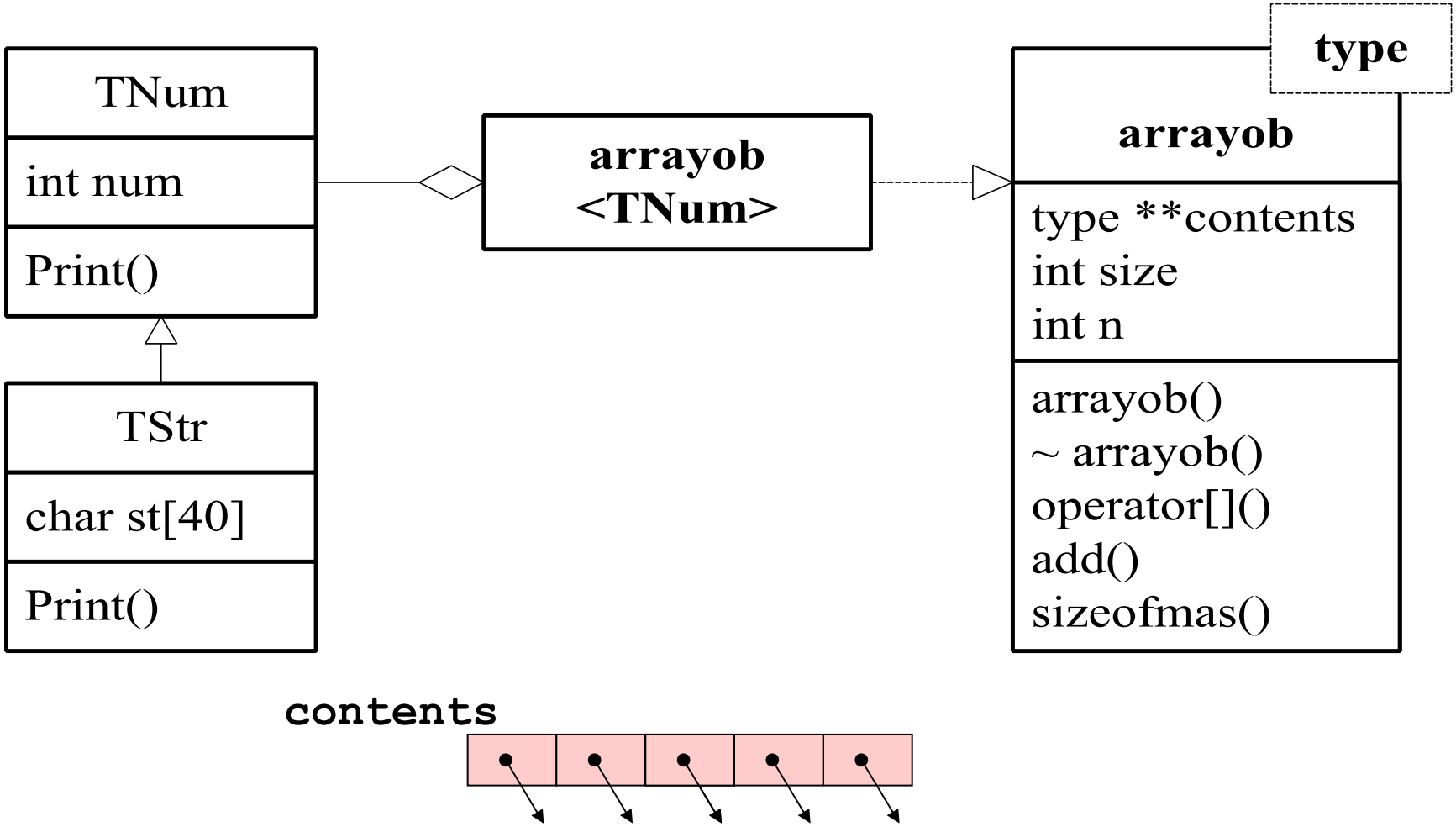
```
#include <string.h>
#include <iostream>
using namespace std;
template <class T>
      T maxx(T x, T y){ return(x>y)?x:y;}
char * maxx(char * x, char * y)
      { return strcmp(x,y) > 0? x:y;}
```

Тестирующая программа

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

7.7 Контейнер на основе шаблона (Ex7_07)

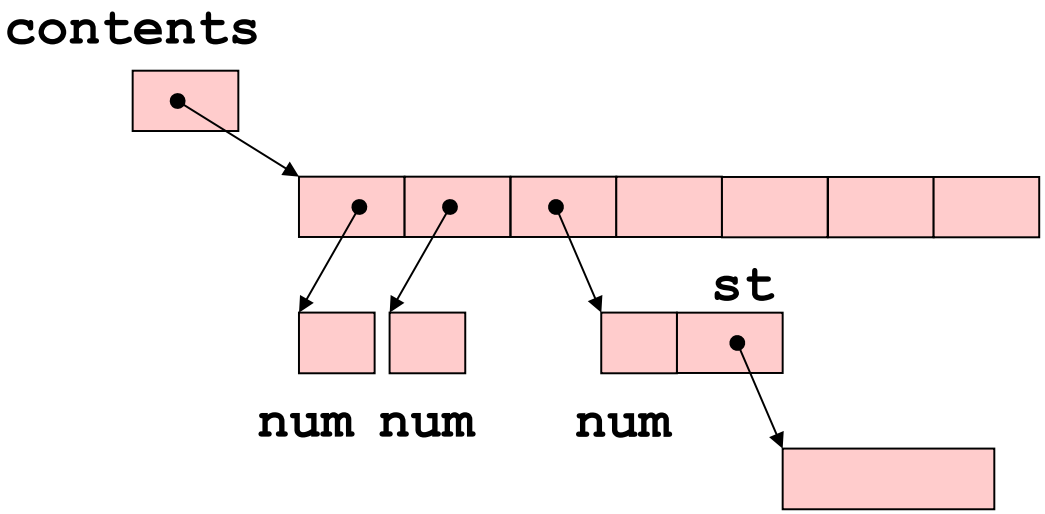


Объявление шаблона класса в файле A.h

```
#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 <iostream>
using namespace std;
class TNum
{ public:   int num;
   TNum(int n):num(n) {}
   virtual ~TNum() {cout<<"Destructor TNum "<<endl;}
   virtual void Print() { cout<< num << " " << endl; }
};
class TStr:public TNum
{ public: char *st;
   TStr(char *s):TNum(strlen(s))
     {st=new char[num+1];strcpy(st,s);}
    ~TStr(void)
     { cout<<"Destructor TStr.";  delete [] st;}
   void Print(void)
     {TNum::Print(); cout << st << " " << endl;}
};
```

Тестирующая программа

```
#include "A.h"
#include "N.h"
arrayob <TNum>  ob_a(5);
int main(int argc, char* argv[])
{ 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;
}
```