Сериализация

Калин Георгиев

28 април 2016 г.

```
f « data;
f » data;
```

Първо изискване: обратимост

```
ostream& operator << (ostream& out, const DynArr<int>& ia)
{
  for (int i = 0; i < ia.size; i++)
    out << ia.arr[i];
  return out;
}
void test ()
{
  DynArr<int> arr (5);
  //...
  ofstream out ("data.txt");

  out << arr;
}
```

```
[1,20,301,4,5] => 12030145
```

Първо изискване: обратимост

```
ostream& operator << (ostream& out, const DynArr<int>& ia)
{
  for (int i = 0; i < ia.size; i++)
    out << ia.arr[i] << "␣";
  return out;
}
void test ()
{
  DynArr<int> arr (5);
  //...
  ofstream out ("data.txt");

  out << arr;
}
```

```
[1,20,301,4,5] => 1 20 301 4 5
```

Второ изискване: еднозначност

```
ostream& operator << (ostream& out, const DynArr<int>& ia)
{
  for (int i = 0; i < ia.size; i++)
    out << ia.arr[i] << "␣";
  return out;
}
void test ()
{
  DynArr<int> arr1 (5), arr2 (4);
  //...
  ofstream out ("data.txt");

  out << arr1 << arr2;
}
```

```
[1,2,3]; [4,5,6,7] => 1 2 3 4 5 6 7
```

Второ изискване: еднозначност

```
ostream& operator << (ostream& out, const DynArr<int>& ia)
{
  out << "[";
  for (int i = 0; i < ia.size-1; i++)
    out << ia.arr[i] << ",";
  if (ia.size > 0)
    out << ia.arr[ia.size-1];
  out << "]";
  return out;
}
void test ()
{
  DynArr<int> arr1 (5), arr2 (4);
  //...
  ofstream out ("data.txt");

  out << arr1 << arr2;
}
```

[1,2,3]; [4,5,6,7] => [1,2,3][4,5,6,7]

Оптимизация: предвидимост

```
[1,2,3]; [4,5,6,7] => [1,2,3][4,5,6,7]
```

```
istream& operator >> (istream& in, DynArr<int>& ia)
{
    DynArr<int> result(0); char c; int x;
    in >> c; assert (c == '[');
    while (c != ']' && in.peek() != ']')
    {
      in >> x;
      result += x;
      in >> c;
      assert(c == ',' || c == ']');
    }
    ia = result;
    return in;
}
void test ()
{
  DynArr<int> arr (0);
  ifstream in ("data.txt");
  in >> arr;
}
```

Оптимизация: предвидимост

```
ostream& operator << (ostream& out, const DynArr<int>& ia)
{
  out << ia.length() << "␣";
  for (int i = 0; i < ia.size; i++)
    out << ia.arr[i] << "␣";
  return out;
}
void test ()
{
  DynArr<int> arr1 (5), arr2 (4);
  //...
  ofstream out ("data.txt");

  out << arr1 << arr2;
}
```

```
[1,2,3]; [4,5,6,7] => 3 1 2 3 4 4 5 6 7
```

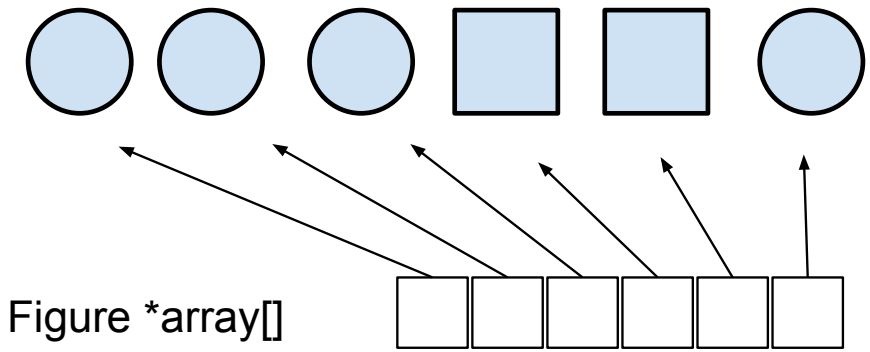
Оптимизация: предвидимост

```
[1,2,3]; [4,5,6,7] => 3 1 2 3 4 4 5 6 7
```

```
istream& operator >> (istream& in, DynArr<int>& ia)
{
    int newSize; in >> newSize; DynArr<int> result (newSize);
    for (int i = 0; i < newSize; i++)
    {
      in >> result[i];
    }
    ia = result;
    return in;
}
void test ()
{
  DynArr<int> arr (0);
  ifstream in ("data.txt");
  in >> arr;
}
```

Сериализация на хетерогенни контейнери

"Записване" на хетерогенен контейнер във файл



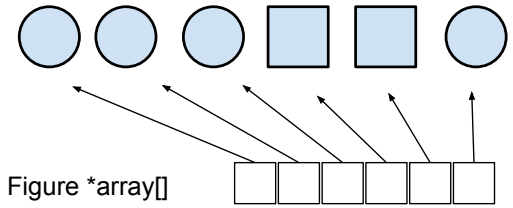

```
DynArr<Figure*> arr[10];
file << arr;
file >> arr;
```

Директен подход не работи

```
ostream& operator << (ostream& out, DynArr<Figure*>& a)
{
  out << a.length() << "␣";
  for (int i = 0; i < a.length(); i++)
    (1) out << a[i] << "␣";      //Figure* ?!?!?
    (2) out << *a[i] << "␣";     //Figure is abstract
    (3) out << a[i]->save(out);  //virtual function required
  return out;
}
void test ()
{
  DynArr<Figure*> arr[10]; //...
  ofstream out ("data.txt");
  out << arr;
}
```

- Circle::save записва радиус
- Rectangle::save записва две страни
- save трябва да отговаря на всички условия за сериализиране

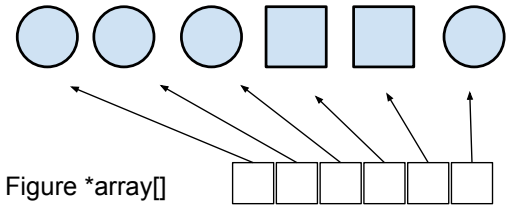
Трето изискване: Разпознаваемост



- Нека всички окръжности са с радиус 1, а всички правоъгълници със страни 2

```
array => 6 1 1 1 2 2 2 2 1
```

Трето изискване: Разпознаваемост



- Нека всички окръжности са с радиус 1, а всички правоъгълници със страни 2

```
array => 6 1 1 1 2 2 2 2 1
```

Трето изискване: Разпознаваемост

```
void circle::save (ostream& out)
{ out << "circle " << r << " "; }
void rectangle::save (ostream& out)
{ out << "rect " << a << " " << b << " "; }
```

```
array => 6 circle 1 circle 1 circle 1 rect 2 2 rect 2 2 circle 1
```

"Фабрика" за обекти

```
array => 6 circle 1 circle 1 circle 1 rect 2 2 rect 2 2 circle 1
```

```
istream& operator >> (istream &in, DynArr<Figure*> a)
{
  int newSize; in >> newSize; DynArr<Figure*> result (newSize);
  for (int i = 0; i < newSize; i++)
  {
    //what is result[i]???
    result[i]->read(in);
  }
}
```

- read!

“Фабрика” за обекти

```
array => 6 circle 1 circle 1 circle 1 rect 2 2 rect 2 2 circle 1
```

```
istream& operator >> (istream &in, DynArr<Figure*> a)
{
  int newSize; in >> newSize; DynArr<Figure*> result (newSize);
  for (int i = 0; i < newSize; i++)
  {
    result[i] = new WHAT; //WHAT?!?
    result[i]->read(in);
  }
}
```

"Фабрика" за обекти

```
istream& operator >> (istream &in, DynArr<Figure*> a)
{
  int newSize; in >> newSize; DynArr<Figure*> result (newSize);
  string type;
  for (int i = 0; i < newSize; i++)
  {
    in >> type;
    result[i] = new type; //unfortunately NOT!!!
    result[i]->read(in);
  }
}
```

"Фабрика" за обекти

```
istream& operator >> (istream &in, DynArr<Figure*> a)
{
  int newSize; in >> newSize; DynArr<Figure*> result (newSize);
  string type;
  for (int i = 0; i < newSize; i++)
  {
    in >> type;
    result[i] = new Figure::factory (type);
    result[i]->read(in);
  }
}
```

"Фабрика" за обекти

```
class Figure
{
  //....
  static Figure* factory (string type)
  {
    if (type == "circle") return new Circle (0);
    if (type == "rect") return new Rectangle (0,0);
    assert (false);
    return NULL;
  }
};
```

"Фабрика" за обекти

```
class Figure
{
  //....
  virtual void read (istream &in) = 0;

  static Figure* factory (string type)
  {
    if (type == "circle") return new Circle (0);
    if (type == "rect") return new Rectangle (0,0);
    assert (false);
    return NULL;
  }
};
void Circle::read (istream &in)
{ in >> r; }
void Rectangle::read (istream &in)
{ in >> a >> b; }
```

"Фабрика" за обекти

```
istream& operator >> (istream &in,
                      DynArr<Figure*> a)
{
  int newSize;
  in >> newSize;
  DynArr<Figure*> result (newSize);
  string type;
  for (int i = 0; i < newSize; i++)
  {
    in >> type;
    result[i] = new Figure::factory (type);
    result[i]->read(in);
  }
}
```

```
class Figure
{
  //....
  virtual void read (istream &in) = 0;

  static Figure* factory (string type)
  {
    if (type == "circle") return new Circle (0)
    if (type == "rect") return new Rectangle (0
    assert (false);
    return NULL;
  }
};
void Circle::read (istream &in)
{ in >> r; }
void Rectangle::read (istream &in)
{ in >> a >> b; }
```

Благодаря ви за вниманието!