9. Тема

Масиви от указатели към обекти. Move семантика - нужда, смисъл, move CC, move ОП=, std::move

Масиви от указатели

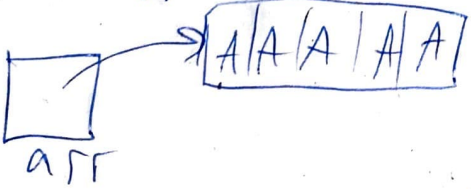
Има 3 начина за създаване на масиви:

I статичен масив
- Compiletime се решава неговия размер
- високо locality => бързодействие
- размерът му не може да бъде променян

Пример
```
A arr[10];
```

II динамичен масив от обекти (dynamic)
- размерът му може да се променя (delete, new)
- високо locality

```
A* arr = new A[5];
```



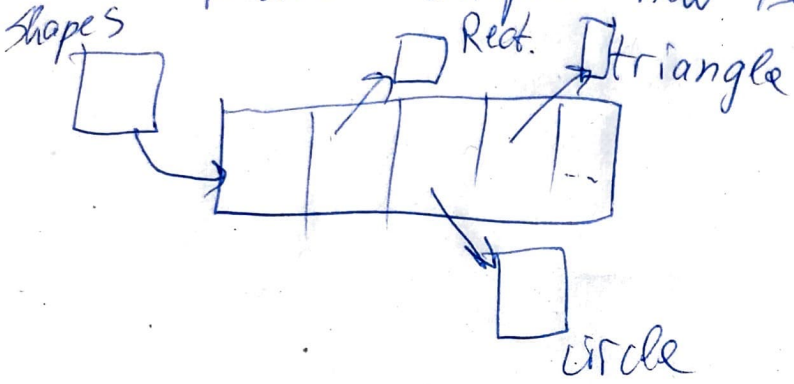
- трудно заемане на елемента
- при уголемяване се налага resize(), изтриване на целия масив и създаване на нов

III масив от указатели към обекти

- много лесно swap-ване на елементи.
- ниско locality => бавно последователно четене
- избягване на нуждата от def A(), подържа nullptr (празни елементи)

Пример:

Base** shapes = new Base*[10];



Move семантика

Ползи:

Имаме обект, който е на края на своя жизнен цикъл. Искаме да преместим ресурсите му в друг обект и да ги "откъсим" от настоящия

Value:

- референтни типове
- име на променлива, ф-ция, обекти
- променливи, които бихме поставили от лявата страна на равното

rvalue

prvalue

- неименувани променливи
- литерали - 73, "ABC", nullptr

xvalue

- expiring value
- обекти, които са на края на своя жизнен цикъл

rvalue = prvalue + xvalue;

Пример: f(int&& n)

rvalue ref

f(3); ✓

int n = 3

f(n); ✗

f(int n) - lvalue, rvalue

f(int& n) - lvalue

f(const int&) - lvalue, rvalue

Move CC

- конструктор за откраднване на данните

Диаграма:

Пример:

```
String(String&& other) {
        moveFrom(std::move(other));
}
void moveFrom(String&& other) {
        _data = other._data;
        other._data = nullptr;
}
```

move op=

```
String& operator=(String&& other)
{   if (this != &other) {
        free();
        moveFrom(std::move(other));
    }
    return *this;
}
```

std::move
- преобразува lvalue във xvalue

Пример:

```
String s1 = "ABC";
String s2 = s1; // CC
String s3(std::move(s1)) // move CC
String s4(s1) // undefined behaviour
```

⚠ Използване на std::move във ф-ция

```
f(A&& obj){

}
```

във scope-а на ф-цията obj се третира като lvalue, тъй като има име

=> g(std::move(obj));

⚠ noexcept

Runtime ако ф-цията throw-не насилва std::terminate

Пример

```
A(A&&) noexcept;
A& op=(A&&) noexcept;
```

- използва се за оптимизации от компилатора

MyString.h

```
#pragma once
#include <iostream>

class MyString
{
    char* _data = nullptr;
    size_t _size = 0;
    size_t _cap = 0;

    void copyFrom(const MyString& other);
    void moveFrom(MyString&& other) noe
    void free();

    void resize(unsigned _newSize);
    explicit MyString(size_t stringLen);
public:
    MyString();
    MyString(const char* data);
    MyString(const MyStr& other);
    MyStr(MyStr&&)noexcept;
    MyStr& op=(const MyStr&);
    MyStr& op=(MyStr&&)noexcept;
~MyStr();
  size_t getCap() const;
  size_t getSize() const;
  const char* c_str() const;
```

```
MyString& operator+=(const MyStr& other);

char& operator[](size_t index);
char operator[](size_t index) const;

friend std::ostream& operator<<(std::ostream& os,
                                const MyStr& obj);

friend std::istream& operator>>(std::istream& is, MyStr& obj);

friend MyStr operator+(const MyStr& lhs, const MyStr& rhs);
};

bool operator==(const MyStr& lhs, const MyStr& rhs);
bool operator!=(---);
bool operator<(---);
bool operator<=(---);
bool operator>(---);
bool operator>=(---);
```

MyString.cpp

```
#include "MyString.h"
#include <algorithm>
#pragma warning(disable:4996)
static unsigned roundToPowerOfTwo(unsigned v)
{
    v--;
    v|= v>>1;
    v|= v>>2;
    v|= v>>4;
    v|= v>>8;
    v|= v>>16;
    v++;
    return v;
}
static unsigned dataToAllocateByStringLen(unsigned strLen)
{
    return std::max(16u, roundToPowerOfTwo(strLen+1));
}
void MyStr::copyFrom(const MyStr& other)
{
    size=other.size;
    cap=other.cap;
    _data=new char[cap];
    strcpy(_data, other._data);
}
void MyStr::moveFrom(MyStr&& other)
{
    _data=other._data;
    size=other.size;
    cap=other.cap;
    other._data=nullptr;
    other.cap=other.size=0;
}
```

```
void MyStr::free()
{
    delete[] _data;
    _data = nullptr;
    _size = _cap = 0;
}
void MyStr::resize(unsigned newCap)
{
    char* newData = new char[newCap + 1];
    strcpy(newData, _data);
    delete[] _data;
    _data = newData;
    _cap = newCap;
}
MyStr::MyStr(size_t stringLen)
{
    _cap = dataToAllocByStringLen(stringLen);
    _data = new char[_cap];
    _data = '\0';
}
MyStr::MyStr() : MyStr("") {}
MyStr::MyStr(const char* data)
{
    if(!data)
        throw std::invalid_argument("Invalid str");
    _size = strlen(data);
    _cap = dataToAllocByStrLen(_size);
    _data = new char[_cap]; strcpy(_data, data);
}
```

```
include "MyString.h"
MyStr::MyStr(const MyStr& other)
  {copyFrom(other);
  }

MyStr::MyStr(MyStr&& other)noexcept
   {
    moveFrom(std::move(other));
   }
OP=
MOP=

MyStr::~MyStr()
  { free();
  }
MyStr::
size_t getCap() const
 {
  return cap;
}
size MyStr::getSize() const
 { return size;
 }
const char* c_str() const
 {return _data;
 }
MyString& MyStr::opt=(const MyStr & other)
{
   size+= other.size;
   if(size>=cap)
        resizeData ToAllocByStrLen(size);
   strcat(_data, other._data);
   return *this;
```

```
char & MyStr::operator[](size_t index)
{
  return _data[index];
}
char MyStr::operator[](size_t index) const
{
  return _data[index];
}

std::istream& op>>(std::istream& is, MyStr & ref)
{
  char buff[1024];
  is >> buff;
   size_t buffLen = strlen(buff);
  if (buffLen > ref.getCap())
      resize(dataToAllocByStrLen(buffLen));
  strcpy(ref._data, buff);
  ref.size = buffLen;
  return is;
}
std::ostream& op<<(std::ostream& os, const MyStr& obj)
{
   os << obj.c_str();
}
MyString op+(const MyStr& lhs, const MyStr rhs
{  MyStr res(lhs.getSize() + rhs.getSize());
   res += lhs; res += rhs;
   return res;
}
```

MyString.cpp
N=3

```
bool operator==(const MyStr& lhs, const MyStr& rhs)
{
    return !strcmp(lhs.c_str(), rhs.c_str());
}

bool operator!=(——11——)
{
    return strcmp(lhs.c_str(), rhs.c_str());
}

bool operator<(——11——)
{
    return strcmp(lhs.c_str(), rhs.c_str())<0;
}

bool operator<=(——11——)
{
    return strcmp(lhs.c_str(), rhs.c_str())<=0;
}

bool operator>(——11——)
{
    return strcmp(lhs.c_str(), rhs.c_str())>0;
}

bool operator>=(——11——)
{
    return strcmp(lhs.c_str(), rhs.c_str)>=0;
}
```