Тема 9

Конвертиращи конструктори. Композиция и агрегация. Шаблони. Необходими функции в шаблонен клас/шаблонна функция

Всички конструктори с по един параметър се наричат конвертиращи конструктори.
Нека имаме клас A и в него конвертиращ конструктор, който приема const char*:

```
class A
{
 A(const char*)
  { // do something }
};
```

и функция, която приема обект от тип A:

```
void f(const A&)
```

В такива функции можем да извикаме самата функция като й подадем const char*. Така се създава обект, благодарение на конвертиращия конструктор, който приема параметър const char*

Пример:
f("ABC") – валидно

Можем да забраним на конструктор да бъде конвертиращ с ключовата дума explicit

Пример:

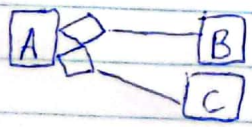
```
explicit A(int x)
```

f(10) -> не се изпълнява, защото няма такъв конвертиращ конструктор

Композиция (has-a relationship)

```
struct A
{
  B obj1;
  C obj2;
  int* arr;
};
```

диаграма:



Композиция - класове като елементи на други класове (влагане на класове). Един клас може да включва в себе си обекти от други класове като член-данни. Когато обект от даден клас се дефинира, и член-данни, неговия конструктор трябва да се грижи конструкторите на член-данните да се извикат.

Пример:

```
Коли конструктор
A(const A& other) : obj1(other.obj1), obj2(other.obj2)
{
  copyFrom(other); // грижи се само за динамичните данни
}

A& operator=(const A& other) {
if (this != &other)
{
  free();
  obj1 = other.obj1; // operator= и за двата обекта
  obj2 = other.obj2;
  copyFrom(other);
} return *this; }
```

Извод: Когато имаме композиция и в главния клас има необработена динамична памет, то трябва в коли-конструктора и operator= експлицитно да се извикат коли-конструкторите и оператор= за всички композирани обекти

```
~A()
{
  free();
}
```

Деструкторите на композираните обекти НЕ трябва да се извикват експлицитно в деструктора на главния клас

Пример 1:

```
class A
{
  ....
};
class B
{
  int n;
  A obj;
};
```

В тази ситуация се извикват default-ните конструктори на n и на obj.

Пример 2:

```
class A
{ public:
  A(int a, int b)
  ....
};
class B
{ public:
  B(): A(1,2) { }
private: int n;
  A obj;
};
```

Тук A няма default-ен конструктор. Това означава, че в конструктора на B трябва да се извика експлицитно някой от неговите конструктори.

Шаблони - това са функции или класове с общо предназначение или иначе казано функция или клас, която не работи с променливи от някакъв дефиниран тип, а с абстрактни променливи.

Пример:

```
template <typename T>
T sum(const T& a, const T& b)
{ return a+b; }
```

sum<int>(3,4) → всяко T се замества с int

Когато правим такива шаблонни функции, трябва да се уверим, че вътре в нея всички оператори са дефинирани над обектите, които се подават (в случая оператор +)

Пример:

```
int main()
{
  int a=5;
  int b=10;
std::cout<< sum<int>(a,b) <<std::endl;   //15
  double c=3.14;
  double d=4.5;
std::cout<<sum<double>(c,d)<<std::endl; // 7.64
}
```

Компилаторът генерира т.нар. шаблонна функция, като замества параметрите на шаблона с типовете на съответните фактически параметри.

Важно правило: Шаблонните класове се пишат в един файл (обикновено .hpp), защото се компилират отделно и няма как да се знае за кои класове да се направи функция