1. Виртуални функции и подтипов полиморфизъм. Динамично свързване. Абстрактни методи и класове. Масиви от обекти и от указатели към обекти.

Виртуални функции се декларират в клас като се сложи virtual пред името на резултата на метода в декларацията на класа. Чрез тях в C++ се реализира механизмът полиморфизъм – едни и същи действия да се реализират по различен начин в зависимост от обекта, към който се прилагат, т.е. действията са полиморфни (много форми). Те могат да бъдат предефинирани в производните класове и чрез указател към базовия клас, когато сочи към производни обекти на клас, да се активира неговия вариант на функцията, а не на базовия. Един виртуален метод може и да няма дефиниция в базовия клас:

virtual <return-type> <name>(...) = 0;

Такива се наричат чисти виртуални функции. Клас с поне една чисто виртуална функция се нарича абстрактен и от него не се издават инстанции, а само указатели или псевдоними. При наследяване, ако в производния не бъде дефинирана виртуалната функция и той става абстрактен. Обикновено абстрактните базови класове определят интерфейса за различните типове обекти в йерархията на класовете. Всички обработки могат да предлагат един и същ интерфей, използвайки полиморфизъм – дефинира се указател към базовия и се използва полиморфно опериране с обекти от производни класове.

... още нещо?

2. Параметричен полиморфизъм. Шаблони на функция и на клас

Пример за оператор извършващ параметричен полиморфизъм е оператор +. Ако подадем две int числа връща int, две double връща double, ако го предефинираме в класа ни за нашите обекти, то ще се държи такак, както сме му задали. Съответно спрямо типа на зададените параметри операторът или методът се държат по различен начин (полиморфно).

Типовете като параметри към функции и клас – в C++ такива средства са шаблоните (templates). Те позволяват създаването на функции и класове, използвайки неопределен тип данни за своите аргументи.
Примери за дефиниция на шаблонни функция и клас:

```
template <typename/class T = int>

void sum(T a, T b) {

        return a+b;

}
```

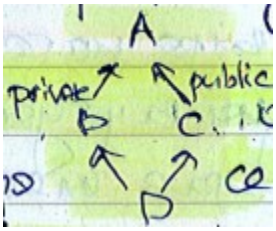
```
template <typename T, typename U>
class MyClass {
	private:
		T a;
		U b;
	public:
		U func(T c, T d, U e);
}
template <typename T, typename U = int>
U MyClass<T, U>::func(T c, T d, U e) {
	return c + b*d + e;
}

Int main() {
	int c = sum<int>(5,6);
	int d = sum(5,6);
	MyClass<int, double> obj;
	double e = obj.func(5,6,7.8);
	return 0;
}
```

3. Множествено наследяване.

Производен клас може да наследи компонентите на един или на няколко базови класа. В първия случай имаме просто, във втория случай множествено наследяване. Така се избягва многократно описание на едни и същи програмни фрагменти. Има различни правила за наследяване на методи от голямата 4-ка, които поради големия обем специфики няма да бъдат разгледани в тази част. Множествено наследяване е мощен инструмент и могат да се изграждат графовидни йерархични структури.

Диамантен проблем:



Тогава се наследяват от B и C компоненти по A и в D също и компилаторът се обърква какво да прави с тези две копия. Когато наследим в B и C класът A виртуално, то се прави само 1 копие в D. Ако B наследи private, а C public примерно, то автоматично се слага public наследяването. Ако бяха двете private или protected си е private/protected.