1. Абстракция със структури от данни. Класове и обекти. Декларация на клас и декларация на обект. Основни видове конструктори. Управление на динамичната памет и ресурсите (“RAII”). Методи – декларация, предаване на параметри, връщане на резултат.

Класовете са типове (абстрактен тип) данни, дефинирани от потребителя. Те могат да обогатяват възможностите на вече съществуващ потребителски тип или да представят нов тип данни. Всеки клас съдържа член-данни и набор от функции наречени член-функции (методи) с различни ограничения за достъп.

Декларацията на класа се състои от заглавие и тяло

class <name-of-class> {

//<body> <- декларирани са член-данни и методи

} [a1, ..., an]; - списък от имена на обекти от класа (може и празен)

Дефинирането на клас се състои освен от декларацията му, също дефиницията на неговите методи, което може да е inline (в класа) или извън класа като имената им се предхождат от <name-of-class>:: където :: е бинарен оператор за разрешение на достъп.

След като клас е дефиниран, то може да се създават негови ексемпляри(инстанции) наречени обекти. Връзката между обект и клас в C++ е като данна и променлива, но обекта се състои от множество компоненти.

Конструктори – почти винаги, когато дефинираме класове има 4 метода, котио задължително се дефинират (ако не се създават служебни, които не се оправят с динамична памет)

Наричат се голямата 4-ка: конструктор, copy-конструктор, деструктор, оператор= (особено, ако работим с динамична памет, те присъстват). При дефинирането на обект автоматично се извиква точно един конструктор от класа (може да са множество и се различават по параметрите им, а останалите от 4-ката са само по 1 екземпляр в класа) с цел да се инициализира обектът. Ако е без явна инициализация, то се извиква default конструктор (по подразбиране).

Явна инициализация може да се извърши:

- <name-of-class> <name-of-object> (<list-of-params>);

- <name-of-class> <name-of-object> = <other-object-of-class>

Във втората инициализация се използва copy–конструкторът на класа.

Ще отбележим, че в тялото на дефиницията на методите на класа не се използва обектът, върху който е приложен методът, а неявния параметър this. Също кодът на методите се намира само на едно място в паметта. Чак присъздаване на обкет се заделя памет за класа.

Методи – по-специално за конструкторите е, че името на класа съвпада с името на конструктора. Типът на конструктора не е оказан (типът е на указателя this). Те се изпълняват автоматично при създаване на обект. Не могат да се извикват явно чрез или косвено чрез указател към обект.

Това извикване става така за член-функции:

MyClass x;

x.a; //явно извикване

x.f();

MyClass x, *py;

Py = &x;

(&x)->a; // косвено

py -> a; //косвено

(*py).a //пак явно

Декларацията на метод на класа се случва в декларацията на класа. В зависимост от правата на достъп, той може да бъде достъпен от обект на класа както се достъпват и член-данни със същия достъп. MyClass x; Оттам нататък предаването на параметри x.f(5); и типът на резултата е същото като при обикновените функции с изключения на това, че се добавя неявният this указател сочещ текущия обект върху който се извиква метода, а и има методи, които са изключения като голямата 4-ка и методи за предефиниране на оператори за класа.

Обща схема за дефиниране на big 4:

```
class MyClass {
	private:
		int x;
	public:
		MyClass() {x=0;}
		MyClass(int x) {
			this->x=x; // тъй като променливата x закрива член-данната x, то го достъпваме с this
		}
		~MyClass() {
			// delete data using dynamic memory
		}
```

```
        MyClass(const MyClass& other) { //copy-конструктор
                this->x = other.x;
        }
        MyClass& operator=(const MyClass& other) {
                If (this != &other) {
                        delete(); // функция освобождаваща динамично заделена памет
                        copy(other); // функция копираща динамично заделена памет
                }
                return this; // връща се псевдонима на обекта
        }
}
```

Имаме 3 модификатори за достъп – public, private и protected. Методи и данни попадащи в област на действие на модификатора (от началото на label-а до следващия модификатор или до края на класа ако няма следващ модификатор) могат да се достъпват явно чрез обект на класа.

private и protected не може да се достъпват извън класа, а само вътре в него. По default в class е private.

2. Наследяване. Производни и вложени класове. Достъп до наследените компоненти. Капсулация и скриване на информацията. Статични полета и методи.

Производните класове (derived classes) и наследяването са една от най-важните характеристики на ООП. От вече съществуващ клас се създава нов клас. Класът, от който се създава, се нарича базов, а този, който се създава, производен. Той може да наследи компонентите на един или няколко базови класа. На свой ред и той може да е базов на друг и така създаваме йерархии от данни. Базовите класове могат да се наследяват също като public, private и protected.

Производните класове имат достъп само до public и protected наследени компоненти, но не и до private. Ако искаме да предадем нещо по производните класове без да е public, то използваме protected. Ако не бъде специфицирано как да се наслади базовия клас по default е private.

Енкапсулация и скриване на информацията – вече разгледахме абстракция на данни (разделяне методите за използване на данните от техните конкретни представяния) и наследяване. Енкапсулацията получаваме като скрием с label private или protected конкретните имплементации на АТД-класа и оставим само public методите (винаги методи, никога член-данни) още наричани интерфейс за комуникация с външния свят. Трябва да можем на класа да гледаме като на black box с определен набор от услуги, които предоставя.

Статичните член-данни са такива данни, които са свързани с конкретния клас, а не с екземпляри на класа. По същия начин и статичните методи са свързани с класа и се ползват за обработка на статичните данни. Те не са свързани с обект и следователно нямат и this указател. Също нямат право на достъп до други не-статични данни или функции. Статичните данни и методи се декларират с думата static пред типа. Статичните променливи трябва допълнително да се инициализират извън класа. T <class-name>::<static-var-name> = <smth>;

Могат да бъдат създавани класове с изцяло статични променливи и методи, но като цяло не е добра практика, защото е еквивалентна на това да дефинираме глобален обект, който може да бъде променян от цялата програма.