**Контрольні питання**

1. **Функции-члены** класса могут быть **перегружены** подобно обычным функциям, т.е. в классе может быть несколько функций с одним и тем же именем, но различным количеством и типом параметров (**function overloading**). Компилятор сам выберет нужную функция на основании количества параметров и их типа. Тип возвращаемого результата в перегрузке не участвует.
2. **Функции-члены** класса, подобно обычным функциям, могут иметь одно или несколько **значений по умолчанию** (**default parameters**)
3. Перегрузка функции **предпочтительнее** значений по умолчанию в следующих случаях:

* нет необходимости в значении по умолчанию;
* нужны различные алгоритмы реализации функций;
* нужно, чтобы список параметров поддерживал различные типы.

1. **Конструкторы**, подобно другим функциям-членам класса могут быть **перегружены** и иметь **значения** **по умолчанию**, т.е. вы можете создавать и инициализировать объект различными способами. Это очень мощный и гибкий инструмент **ООП**. Например, **объект** типа **Worker** (Служащий) может быть **создан** **несколькими** способами : конструктор без параметров устанавливает числовые поля объекта в нули, символьные - в пустые строки; конструктор с целым параметром равным 0 вводит данные о служащем с клавиатуры, если же этот параметр равен другому числу – то ввод сведений о сотруднике выполняется из базы данных.
2. Конструкторы могут вызываться в **два** этапа : этап инициализации и тело конструктора.  
    **CAT () :** // имя конструктора и его параметры   
    **itsAge(5),** // список инициализации  
    **itsWeight(8)**   
    **{**    
    < *операторы конструктора*> // тело конструктора  
    **}**
3. Компилятор **поддерживает** в классах ***:***

* конструктор класса без параметров (**default constructor**);
* деструктор (**destructor**);
* конструктор копирования (**copy constructor**);
* оператор присваивания( **=** ).

Это означает, что их можно опустить при реализации класса.

1. Необходимо **помнить** следующее :

* компиляторный конструктор по умолчанию *просто выделяет память,* не выполняя *инициализации* объекта*;*
* если вы создаете свои *личные конструкторы*, то компилятор *не создаст* конструктор по умолчанию(без параметров)*;*
* компиляторные конструктор копирования и оператор присваивания работают в режиме « *поверхностного копирования* ».

1. **Конструктор копирования** вызывается автоматически всегда, когда создается копия объекта. Обычно это происходит при передаче и возврате параметров-объектов в функции по значению. *Конструктор копирования принимает один параметр – константную ссылку на объект этого же класса* : **CAT (const Cat & theCat);** // конструктор копирования  
   Задача конструктора копирования создать копию из уже существующего объекта **theCat**!!!
2. **Конструктор копирования по умолчанию** просто копирует каждую переменную-член из объекта, передаваемого в качестве параметра в переменные-члены нового объекта. Это называется почленным (**member-wise**) или **поверхностным** (**shallow**) копированием. Если члены-данные вашего класса – указатели, то необходима реализация **личного** конструктора копирования, который должен выполнять «**глубокое копирование**» - т.е. для значений на которые указывают указатели должна также выделяться новая память, иначе возможны «висячие» указатели.
3. B C++ существуют **встроенные типы**, такие как **char, int, double** и т.д. Каждый из них располагает **встроенными операциями** (**операторами**), такими как инкремент (++), сложение (+), увеличение на (+=), сравнение на равенство (= =), присваивание (=) и др.   
   При помощи знаков операций составляются математические и логические выражения.
4. Все встроенные операции С++ (кроме =) первоначально нельзя применять для объектов классов. Но можно включить в состав класса набор методов для выполнения действий аналогичным действиям операторов. Однако пользоваться подобным классом будет не очень удобно.
5. Средства языка **С++** позволяют **добавить** необходимые операции к вашим собственным классам, т.е. перегрузить стандартные операторы языка своими, вложив в них нужный смысл. Перегрузка операций реализуется с помощью включения в состав класса специальных функций типа **operator()**. (
6. Для исследования принципов разработки операторов используем простейший личный класс **Counter** – счетчик числа повторений с одним членом данным типа безнаковое короткое целое.
7. При перегрузке операций часто используется указатель **this**. Использование этого указателя, в котором хранится адрес обрабатываемого, текущего объекта, делает ненужным создание временных объектов. Разыменованный указатель \***this** (значение объекта) часто возвращают перегруженными операторами. Необходимо уметь реализовывать следующие типы операторов:

* унарный префиксный/постфиксный операторы инкремента/декремента (**++** и **- -**) : должны увеличивать/уменьшать нужные члены-данные на 1 и возвращать измененный объект в виде константной ссылки в качестве результата для возможного последующего присваивания; постфиксность задается с помощью параметра-флага (**int**);
* бинарные математические операторы (**+**, **-**, **\***, **/**, **%**) :   
  имеют один параметр – второй объект находящийся справа от знака операции принимается по константной ссылке; затем создается временный объект для хранения результатов вычислений над нужными членами-данными текущего и ссылочного объектов; временный объект и возвращается в качестве результата для возможного последующего присваивания;
* бинарные операторы сравнения (**>, <, ==, !=, >=, <=**) :   
  имеют один параметр – второй объект находящийся справа от знака операции принимается по константной ссылке и тип возврата **int** или **bool**; выполняют сравнение нужных членов-данных текущего и ссылочного объектов; в качестве результата возвращается истина (1, **true**) или ложь (0, **false**) ;
* бинарный оператор присваивания (необходимо разрабатывать только для классов с **динамическим** выделением памяти) ( **=** ) : имеет один параметр – второй объект находящийся справа от знака операции принимается по константной ссылке; затем прежде всего необходимо сделать проверку от присваивания самому себе и если оно обнаружено, то вернуть текущий объект как результат выполнения оператора; в противном случае необходимо уничтожить текущий объект, создать новый и в него перенести данные из ссылочного объекта и вернуть новый текущий, измененный объект в качестве результата;
* операторы преобразования типов;
* операторы индексации ( **[ ]** ), вызова функции ( **( )** ), косвенной адресации ( **–>** );
* операторы ввода/вывода в поток.

1. *Операторы для встроенных типов перегружать нельзя. Нельзя изменить приоритет выполнения и арность оператора, т.е. нельзя изменить унарность или бинарность. Нельзя создавать новые операторы.* Нельзя объявить \*\* оператором «возведения в степень», как принято в языке **Fortran** – в С++ нет такого обозначения операции. Однако, как принято в **Basic** и **Excel** ( **^ )** - можно, т.к. в С++ есть битовая операция ^ .
2. Перегрузка операторов – один из тех аспектов С++, которым часто злоупотребляют начинающие программисты. Забавно сделать оператор + вычитанием, а оператор \* - сложением, но ни один профессиональный программист этого делать не будет.
3. Многие стандартные встроенные классы (**string**, **TDate**, **TTime** и т.д.) имеют в своем составе большой набор перегруженных конструкторов и операторов, упрощающих работу с объектами этих классов
4. **Цель перегрузки операторов – облегчение работы с программой и улучшение восприятия кода.**