**МДК 05.02 Разработка кода информационных систем**

**Вопросы по теории (обязательно с примерами!):**

1. Рекурсия и рекурсивные алгоритмы Прямая рекурсия, косвенная рекурсия. Рекурсивный спуск и рекурсивный подъем. Рекурсивная триада. Глубина рекурсии, объем рекурсии. Сравнение рекурсии и итерации.

(

Под рекурсией понимают прием последовательного сведения решения некоторой задачи к решению совокупности "более простых" задач такого же класса и получению на этой основе решения исходной задачи.

Рекурсивный алгоритм – это алгоритм, в определении которого

содержится прямой или косвенный вызов этого же алгоритма

Прямая рекурсия- обращение функции к самой себе предполагает, что в

теле функции содержится вызов этой же функции, но с другим

набором фактических параметров

Косвенная рекурсия — функция содержит вызовы других функций из своего

тела. При этом одна или несколько из вызываемых функций на

определенном этапе обращаются к исходной функции с измененным

набором входных параметров.

**Последовательный вызов функцией самой себя называют рекурсивным спуском, последовательный выход из многократного вызова — рекурсивным подъёмом**

рекурсивную триаду:

параметризация – выделяют параметры, которые

используются для описания условия задачи, а

затем в решении;

база рекурсии – определяют тривиальный случай,

при котором решение очевидно, то есть не

требуется обращение функции к себе;

декомпозиция – выражают общий случай через

более простые подзадачи с измененными

параметрами.

Глубина рекурсивных вызовов– наибольшее одновременное количество

рекурсивных обращений функции, определяющее максимальное

количество слоев рекурсивного стека, в котором осуществляется хранение

отложенных вычислений.

Объем рекурсии - количество вершин полного рекурсивного дерева без

Единицы

**Рекурсия** - это такой способ организации обработки данных, при котором программа вызывает сама себя непосредственно, либо с помощью других программ.

**Итерация** - это способ организации обработки данных, при котором определенные действия повторяются многократно, не приводя при этом к рекурсивным вызовам программ.

**Пример рекурсии** : фракталы, факториал

)

1. Фракталы. Определение, свойства. Применение. Примеры построения.

(

**Фрактал** (fractus — дроблёный, сломанный, разбитый) — это сложная, бесконечно самоподобная геометрическая фигура, каждый фрагмент которой повторяется при уменьшении масштаба.

**Свойство:**

1. обладает сложной структурой при любом увеличении;
2. является (приближенно) самоподобной;
3. обладает дробной фрактальной [размерностью](https://elementy.ru/posters/fractals/dimensions), которая больше топологической;
4. может быть построена рекурсивными процедурами.

Принципы построения фракталов используются в физике, в таких разделах, как гидродинамика, физика плазмы, электродинамика и радиоэлектроника. Одно из самых заметных изобретений в этой области — фрактальная антенна*,* которая была разработана американским инженером Натаном Коэном в 1995 году.

Компьютерная графика и анимация: Фракталы используются для создания реалистичных изображений природных объектов и сцен, таких как горы, облака, деревья и береговые линии

**Кривая Коха ("снежинка Коха"):**

1. Начните с прямого отрезка линии.
2. Разделите линию на три равные части и удалите среднюю часть.
3. Постройте равносторонний треугольник на месте удаленной части.
4. Повторите эти шаги для каждой из четырех новых линий, полученных после добавления треугольника.
5. Продолжайте процесс итеративно для получения всё более сложной кривой.

Примеры: Кривая минковская, кривая коха, кривая дракона

)

1. Динамическая память. Динамическое распределение памяти. Размещение объектов в куче. Освобождение памяти.

(

Динамическая память - это **оперативная память, которая выделяется в процессе работы программы**. При динамическом размещении заранее не известны ни тип, ни количество размещаемых данных, к ним нельзя обращаться по именам, как к статическим переменным.

При **динамическом распределении памяти** объекты размещаются в т. н. «[куче](https://ru.wikipedia.org/wiki/%D0%9A%D1%83%D1%87%D0%B0_(%D0%BD%D0%B5%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C))» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *heap*): при конструировании объекта указывается размер запрашиваемой под объект памяти, и, в случае успеха, выделенная область памяти, условно говоря, «изымается» из «[кучи](https://ru.wikipedia.org/wiki/%D0%9A%D1%83%D1%87%D0%B0_(%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C))», становясь недоступной при последующих операциях выделения памяти. Противоположная по смыслу операция — освобождение занятой ранее под какой-либо объект памяти: освобождаемая память, также условно говоря, возвращается в «[кучу](https://ru.wikipedia.org/wiki/%D0%9A%D1%83%D1%87%D0%B0_(%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C))» и становится доступной при дальнейших операциях выделения памяти.

Операции

**New**(P\_int);   
**Dispose**(P\_int);

)

1. Понятие указатель. Использование типизированных и не типизированных указателей. Проблемы использования указателей. Утечка памяти.

(

**Указатель** представляет собой переменную целого типа, которая интерпретируется как адрес байта памяти, содержащий некоторый элемент данных. Этим элементом может быть переменная, константа, адрес другой переменной и т.д.

Pr\_int : ^integer

***типизированные*** (содержат адрес ячейки памяти данного типа) и ***бестиповые*** (содержат адрес оперативной памяти, не связанный с данными какого-либо определенного типа).

Проблема указателей в том, что програмист должен сам отслеживать время жизни указателя.

**Утечка памяти:** Это происходит, когда программист выделяет память в куче (используя New или GetMem), но забывает освободить её (с помощью Dispose или FreeMem). В результате память остается занятой до окончания работы программы.

)

1. Организация элементарных структур на основе динамической памяти. Очередь, Стек, Дек, Списки (односвязные, двусвязные, линейные, кольцевые).

( Очередь — структура данных, поддерживающая операцию удаления с начала и добавление в конец

Стек — структура данных, поддерживающая операцию удаления и добавление в конец

Дек — структура данных, поддерживающая операцию удаления и добавление в начало и в конец

Список — структура данных, поддерживающая операции добавления и удаления любого узла

Односвязные списки — Имеют в своей структуре указатель на следующий узел списка

Двусвязные списки — Имеют в своей структуре указатель на следующий и предыдущий узел списка

Линейный список — Список имеющий начало и конец

Кольцевой список — список, у которого начало и конец связанны вместе

Пример узла:

TList = record

Data: int

Next:^TList

)

1. Организация элементарных структур на основе динамической памяти. Деревья.

Дерево — граф не имеющий циклов

Корнеевой узел — узел не имеющий родителей

Внутренний узел — узел имеющий и родителей и потомков

Листья — узел не имеющий потомственных элементов

Методы :

**Начало с корня:** Сначала создаётся корневой узел, который станет основой всего дерева.

**Добавление узлов:** Когда вы добавляете новый узел, вы должны решить, куда он пойдет. В бинарном дереве, например, если новый элемент меньше текущего, он идет налево, если больше – направо.

**Поиск места для узла:** Если вы хотите вставить элемент, начните с корня и сравните значения, чтобы найти место, где новый узел должен быть присоединен к дереву.

**Соединение узлов:** Когда вы нашли место, присвойте новому узлу указатели на его потенциальных детей (если они есть) и обновите указатель родителя, чтобы он указывал на новый узел.

**Удаление узлов:** Если вы удаляете узел, необходимо убедиться, что все дочерние узлы этого узла переподключены так, чтобы структура дерева оставалась целостной.

**Обход дерева:** Это процесс посещения всех узлов дерева для выполнения какой-либо операции, например, для печати всех значений.

**Очистка:** После того, как узлы больше не нужны, нужно освободить за ними память, чтобы предотвратить утечки памяти.

Tree\_unit = record

Data: int

Left:^Tree\_unit

Right: ^Tree\_unit

1. Парадигмы программирования. Языки парадигм программирования.

Императивное — в данной парадигме программист должен самостоятельно описывать шаги, который должен выполнить программа для достижения результатов(C++, pascal)

Декларативное — в данной парадигме программист описывает нужный результат, а компьютер самостоятельно выполняет действия для достижения цели(SQL)

Функциональное — в данной парадигме программист пишет функции(блоки кода), которые позже вызываются в ходе работы программы( C++, Pascal)

ООП — в данной парадигме программист описывает класс, который содержит собственные свойства и методы (C#)

1. Интегрированная среда разработки приложений Lazarus. Структура программы Lazarus, элементы языка.

**Lazarus** — [открытая](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [среда программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) на языке [Object Pascal](https://ru.wikipedia.org/wiki/Object_Pascal) для компилятора [Free Pascal](https://ru.wikipedia.org/wiki/Free_Pascal). Основная цель — предоставление [кроссплатформенных](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) и свободных средств разработки в [Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8))-подобном окружении Событийно-ориентированное программирование. Основные компоненты в Lazarus и их характеристики.

Любая программа в Lazarus состоит из файла проекта (файл с расширением lpr) и одного или нескольких модулей (файлы с расширениями pas). Каж­дый из таких файлов описывает программную единицу Lazarus.

**Элементы языка в Lazarus –** **это инструменты, которые вы используете для написания кода:**

**Переменные:** Это как ящики для хранения информации, каждый из которых имеет своё имя.

**Типы данных:** Это как разные виды ящиков, предназначенные для хранения определенных вещей, например, чисел или текста.

**Операторы:** Это действия, которые вы можете совершать с информацией, например, складывать числа или сравнивать их.

**Процедуры и функции:** Это как рецепты или инструкции, которые говорят программе, что делать в определенной ситуации.

**Классы и объекты:** Если вы используете Object Pascal, это как более сложные рецепты, которые помогают организовать код, связанный с определенной частью программы, например, с управлением окном или обработкой данных.

1. Событийно-ориентированное программирование. Основные компоненты в Lazarus и их характеристики.

**Компоненты управления** (Control components): Это элементы, с которыми пользователь может взаимодействовать, например:

**TButton:** Кнопка, на которую пользователь может нажать.

**TEdit:** Текстовое поле, в которое пользователь может ввести текст.

**TLabel:** Текстовая метка для отображения текста на форме.

**TMemo:** Большое текстовое поле для ввода или вывода нескольких строк текста.

**TListBox** и **TComboBox:** Списковые элементы для выбора из предопределенных опций.

**Компоненты нестандартного управления:** Используются для реализации функционала, который не имеет графического представления на форме, например:

**TTimer:** Можно использовать для выполнения кода через регулярные интервалы времени.

**TOpenDialog** и **TSaveDialog:** Диалоговые окна для открытия и сохранения файлов.

1. Подпрограммы в Lazarus. Основные способы передачи параметров в подпрограмму, их сравнение.

Передача параметров по значению

При таком способе передачи параметров в функцию или процедуру передается копия переменной. Внутри функции или процедуры можно менять значения переданных параметров, однако в вызывающей программе значения параметров остаются неизменными.

procedure example(x: integer);

begin

x:= x + 5;

WtiteLn(x) //

End

Begin

Var x := 5

Example(x);

Writeln (x);

end

Как видите, после вызова процедуры значения переменных x не изменились.

Передача параметров по ссылке

Другая ситуация при передаче параметров по ссылке. В этом случае изменение параметра внутри функции (процедуры) влечет за собой и изменение значения переменной в вызывающей программе. Для передачи параметра по ссылке нужно перед именем параметра в заголовке указать ключевое слово var. Рассмотрим предыдущий пример, но передадим параметры x и n по ссылке.

procedure example( var x: integer);

begin

x:= x + 5;

WtiteLn(x) //

End

Begin

Var x := 5;

Example(x);

Writeln (x);

End

1. Подпрограммы в Lazarus. Область видимости. Локальные и глобальные идентификаторы.

**Глобальная область видимости** — **идентификатор** доступен во всём тексте программы (во многих языках действует ограничение — только в тексте, находящемся после объявления этого **идентификатора**).

**Локальная область видимости** — **идентификатор** доступен только внутри определённой функции (процедуры).

**Переменная в процедуре локальная**

**В начале программы — глобальная**

1. Подпрограммы в Lazarus. Фактические и формальные параметры.

Фактические параметры — это параметры(значения) передающиеся в функцию или процедуру

**Формальные параметры — это идентификаторы входных данных для подпрограммы.** **Если формальные параметры получают конкретные значения, то они называются фактическими**.

1. Возможности интегрированной среды разработки Lazarus по организации и использования диалогов, диалоговых окон.

**Работа с диалоговыми окнами:**

* **Диалоги сообщений:** Используются для вывода сообщений пользователю (информационных, предупреждений, ошибок) и получения ответа от пользователя. («Вы уверены что хотите удалить выбранный файл?»)
* **Диалоги ввода данных:** Позволяют пользователю ввести данные через текстовое поле или выпадающий список. («Введите ваше имя…»
* **Диалоги выбора файла/папки:** Используются для выбора файлов или папок на диске. («Выберите файл, который вы хотите открыть»)

1. Структуры и принципы организации программных модулей.

**Проекты** –Основная единица организации кода. включать один или несколько модулей, форм и других файлов, необходимых для разработки приложения.

**Модули** –Файлы с расширением .pas. Они содержат определения типов данных, констант, процедур и функций, необходимых для реализации определенных функций или логики в программе.

**Формы** –Используются для создания пользовательского интерфейса приложения. Они представляют собой визуальные компоненты (кнопки, текстовые поля, списки и т.д.), размещенные на окне приложения.

**Библиотеки и компоненты** – uses

1. Технология отладки приложений в интегрированной среде разработки Lazarus. Исключительные ситуации. Структурная обработка исключительных ситуаций.

В Lazarus встроен отладчик — [компьютерная программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) для автоматизации процесса [отладки](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B): поиска [ошибок](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3) в других [программах](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), [ядрах операционных систем](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B), [SQL](https://ru.wikipedia.org/wiki/SQL)-запросах и других видах кода. В зависимости от встроенных возможностей, отладчик позволяет выполнять [трассировку](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D1%81%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), отслеживать, устанавливать или изменять значения [переменных](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) в процессе выполнения кода, устанавливать и удалять [точки останова](https://ru.wikipedia.org/wiki/%D0%A2%D0%BE%D1%87%D0%BA%D0%B0_%D0%BE%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%B0) или условия остановки и так далее.

Исключения в программировании (exceptions) — это **механизм, который позволяет программе обрабатывать нетипичную ситуацию и при этом не прекращать работу**. Благодаря этому разработчик может обработать такие ситуации как деление на ноль, пустое значениие(в случае приведения типов)

Операторы try except —  
**begin**  
  **try**  
   {Выполняемый код}  
  **except**  
   {Код, выполняемый в случае ошибок в основном};  
  **end**;  
**end**.

1. Организация работы с базами данных. Понятие БД, СУБД, виды СУБД. Модели данных. Типы связей. Ссылочная целостность. Механизм доступа к БД в Lazarus.

**База данных** (БД) – это организованное хранилище данных, где информация сохраняется в структурированном виде. Это позволяет легко и быстро находить и использовать нужные данные.

**Система управления базами данных** (СУБД) – это программа, которая помогает создавать, управлять и обслуживать базы данных. СУБД предоставляет инструменты для добавления, изменения, удаления и поиска данных.

**Виды СУБД:**

* Реляционные СУБД (РСУБД);
* Объектно-ориентированные СУБД (ОСУБД);
* NoSQL СУБД. (не знаю что это)

**Модели данных:**

* Реляционная модель;
* Иерархическая модель;
* Объектная модель.

**Типы связей:**

* Один к одному;
* Один ко многим;
* Многие ко многим.

**Ссылочная целостность** гарантирует, что связи между таблицами остаются корректными и согласованными. Это означает, что каждое значение внешнего ключа (foreign key) в одной таблице должно существовать как значение первичного ключа (primary key) в другой таблице.

Механизм доступа к БД в Lazarus:

* T**SQLConnection** (компонент для установления соединения с базой данных.)
* **TSQLQuery** (компонент для выполнения SQL-запросов к базе данных.)
* **TDataSource** (компонент, связывающий данные из базы данных с визуальными компонентами.)
* **TDBGrid** (компонент для отображения данных в табличной форме.)

1. Организация работы с файлами. Характеристики файла. Классификация файлов. Алгоритм работы с файлами.

**Файл** – это набор данных, который хранится на внешних носителях. . Основные характеристики файла включают:

* Имя файла
* Расширение файла (.pas, .exe)
* Размер файла
* Тип файла (.png, .mp3)
* Дата создания

**Классификация файлов:**

* По типу содержимого (текстовые, исполняемые файлы)
* По доступу (только чтение, чтение/запись, скрытые)
* По формату хранения данных (текстовые)

**Алгоритм работы с файлами:**

1. Открытие файла;
2. Чтение данных из файла;
3. Запись данных в файл;
4. Закрытие файла.
5. Организация работы с файлами. Основные операции для работы с последовательными файлами. Пример работы.

**Основные операции для работы с последовательными файлами:**

* **AssignFile:** Связывает файловую переменную с физическим файлом на диске.
* **Reset:** Открывает файл для чтения.
* **Rewrite:** Открывает файл для записи, при этом старое содержимое файла удаляется.
* **Append:** Открывает файл для записи, при этом новые данные добавляются в конец файла.
* **CloseFile:** Закрывает файл.
* **Readln:** Читает строку из файла.
* **Writeln:** Записывает строку в файл.

1. Организация работы с файлами. Основные операции для работы с файлами прямого доступа. Пример работы.

**Основные операции для работы с файлами прямого доступа**

* **AssignFile:** Связывание файловой переменной с физическим файлом на диске.
* **Reset:** Открытие файла для чтения или записи. Для файлов прямого доступа это делается с указанием типа файла.
* **Rewrite:** Создание нового файла для записи или перезапись существующего.
* **Seek:** Перемещение указателя текущей позиции в файле на заданную запись.
* **BlockRead:** Чтение блока данных из файла.
* **BlockWrite:** Запись блока данных в файл.
* **CloseFile:** Закрытие файла.

1. Объектно-ориентированное программирование. Классы и объекты. Обращение к членам класса.

**Объектно-ориентированное программирование** (ООП) — это парадигма программирования, основанная на концепции объектов. Объекты могут содержать данные в виде полей (часто называемых атрибутами или свойствами) и код в виде процедур (часто называемых методами).

**Классы и объекты**

**Класс** – это шаблон или чертеж для создания объектов. Он определяет набор свойств и методов, которые будут у объектов, созданных на его основе.

**Объект** – это экземпляр класса, который содержит реальные значения свойств и может выполнять методы, определенные в классе.

**Основные концепции ООП**

* **Инкапсуляция:** Сокрытие внутренней реализации объекта и предоставление доступа к данным только через методы.
* **Наследование:** Возможность создания нового класса на основе существующего, при этом новый класс наследует свойства и методы родительского класса.
* **Полиморфизм:** Способность объектов разных классов обрабатывать вызовы методов с одинаковыми именами по-своему.
* **Абстракция:** Выделение общих характеристик объекта, скрывая при этом сложные детали реализации.

1. Объектно-ориентированное программирование. Инкапсуляция. Спецификаторы доступа.

**Инкапсуляция** – это принцип, который объединяет данные и методы, работающие с этими данными, в один модуль (класс). Он также скрывает детали внутренней реализации от внешнего мира. Основные цели инкапсуляции

* **Скрытие деталей:** Прячем сложные внутренние механизмы, показывая только необходимые части.
* **Защита данных:** Предотвращаем несанкционированный доступ и изменения.
* **Упрощение использования:** Предоставляем простой интерфейс для взаимодействия.

**Спецификаторы доступа** определяют, кто может видеть и использовать члены класса (поля и методы). В Object Pascal (Lazarus) есть следующие спецификаторы доступа:

* **public** (публичный);
* **private** (приватный);
* **protected** (защищенный);
* **published** (опубликованный).

1. Объектно-ориентированное программирование. Наследование. Полиморфизм.

**Наследование** это способ создать новый класс на основе уже существующего. Новый класс (подкласс) берет свойства и методы базового класса (родителя) и может добавлять новые или изменять существующие.

**Полиморфизм** позволяет использовать объекты разных классов через один и тот же интерфейс. Это означает, что методы базового класса могут быть переопределены в наследниках, и какой метод будет вызван, зависит от типа объекта, который использует этот метод.

1. Объектно-ориентированное программирование. Конструкторы и деструкторы.

**Конструкторы** являются специальными методами класса, которые вызываются при создании объекта этого класса. Основные задачи конструкторов:

1. **Инициализация объекта:** Устанавливают начальные значения полей объекта.
2. **Выделение ресурсов:** Могут выполнять дополнительные операции, такие как выделение памяти или открытие файлов.

**Особенности конструкторов:**

* **Имя совпадает с именем класса:** В большинстве языков программирования имя конструктора совпадает с именем класса.
* **Могут быть перегружены:** В одном классе может быть несколько конструкторов с разными параметрами (перегрузка конструкторов).
* **Вызов базового конструктора:** В случае наследования можно вызывать конструктор базового класса из конструктора производного класса.

**Деструкторы** – это методы класса, которые вызываются при удалении объекта из памяти. Основные задачи деструкторов:

1. **Освобождение ресурсов:** Освобождают выделенные объекту ресурсы, такие как память или открытые файлы.
2. **Финальные операции:** Могут выполнять финализирующие операции перед удалением объекта.

**Особенности деструкторов:**

* **Имя совпадает с именем класса с префиксом ~:** В большинстве языков программирования имя деструктора совпадает с именем класса, но имеет префикс ~.
* **Вызов деструктора:** Деструктор вызывается автоматически при удалении объекта из памяти или явном вызове оператора delete (в языках, поддерживающих явное управление памятью).

**Основное практическое задание**

**Возможен один из вариантов:**

1. Программа на расчет (например, калькулятор процентов, расчет страховых выплат и тд)
2. Будильник/таймер, календарь, ежедневник.
3. Работа с БД/типизированным/текстовым файлом. Вывод данных из файла в таблицу, добавление/изменение записей, сохранение в файл.

**Примечание:** Программа должна быть написана на Лазарусе, иметь обработку исключений, правильные названия компонентов, масштабируемость.

**Дополнительно (освобождены те, кто сдал все ДКР и ЛР):**

1. Задание на построение фрактала на Паскале (один из изученных).
2. Задание на динамическую структуру данных на Паскале (одну из изученных).
3. Задание на разработку приложения на расчет в Лазарусе (с помощью разных типов подпрограмм, параметров и переменных).
4. Задание на разработку приложения в Лазарусе для работы с определенной структурой и сохранение ее в типизированном файле.