**Ответы к вопросам по экзамену по информационным технологиям и программированию (1 семестр)**

# **Основные парадигмы программирования. Преимущества и недостатки.**

1.1 Парадигма программирования и декларативное программирование

**Парадигма программирования** – совокупность идей и понятий, определяющая стиль написания программ. Является базовой программной единицей, в качестве которой может выступать *определение* (декларативное, функциональное программирование), *действие* (императивное), *объект* (объектно-ориентированное) и *другие сущности*.

Программирование подразделяется на:

* Декларативное (описание свойств некоторой базовой единицы)
	+ Функциональное (использование функций в математическом смысле 🡪 результат зависит только от входных параметров и не предполагает изменения состояния памяти. LISP, Haskell, SML, F#)
	+ Логическое (основано на математической логике. Подходит для задач, в которых действуют объекты и отношения между ними. Prolog). Сначала задаём базу из объектов, потом задаём логические вопросы.







* Императивное

1.2 Императивное программирование. Процедурное программирование

Императивное программирование начало развиваться из *процедурного* программирования. Представляет собой последовательность операторов (инструкций), задающих процедуру решения задачи. В основе лежит *оператор присваивания*, служащий для изменения содержимого областей памяти и *концепция памяти, как изменяемого хранилища значений*. Выполнение программы сводится к последовательному выполнению операторов, меняющих состояние памяти из исходного к результату.

**Особенностями являются:**

* Необходимость *явного управления памятью*, в т.ч. через описание переменных
* Малая пригодность для символьных вычислений
* Отсутствие строгой математической основы
* Высокая эффективность на традиционных ЭВМ

1.3 Структурное программирование

Ввиду усложнения вычислительных машин и задач, появляется парадигма *структурного программирования* (в 1970-х годах, авторы Дейкстра и Вирт)

**Особенности:**

* Любая программа состоит из 3 базовых конструкций:
	+ *Последовательное исполнение* – однократное исполнение инструкций в том порядке, в котором они записаны в программе
	+ *Ветвление* – однократное исполнение операторов в зависимости от условия (если …, то …)
	+ *Цикл* – многократное выполнение одной и той же операции, пока выполняется некоторое условие

Эти конструкции в программе могут быть вложены друг в друга каким-угодно образом.

* Повторяющиеся или не повторяющиеся элементы программы, представляют собой *целые логические цепочки*, которые можно выделить в *отдельные подпрограммы*. Тогда текст основной программы будет состоять из вызовов подпрограмм. После выполнения вызванной подпрограммы, исполнение основной программы продолжится с инструкции, расположенной после подпрограммы.
* Разработка идёт пошагово, сверху вниз. Т.е. сначала определяется структура основной программы, в которой вместо рабочих подпрограмм ставятся заглушки. Убедившись в правильной расстановке подпрограмм, начинаем заменять заглушки на рабочие подпрограммы. Причём разработка каждой подпрограммы ведётся таким же образом. Это исключает появление ошибки в месте, которое находится вне поля зрения программиста

**Достоинства:**

* Сокращает количество вариантов построения программы одной и той же спецификации, что снижает её сложность и облегчает понимание другими разработчиками
* Логически связанные операторы находятся визуально ближе, а слабо связанные – дальше, что позволяет обойтись без сторонних форм изображения алгоритма
* Упрощается процесс тестирования и отладки

1.4 Модульное программирование

Ещё большее усложнение проектов привело к идее *модульного программирования*. Идея в том, что программа представляет собой набор независимых блоков (набор подпрограмм), каждая из которых имеет свой контролируемый размер, четкое назначение и детально проработанный интерфейс с внешней средой. Каждая подпрограмма может быть разработана отдельно от других, а отлаженные подпрограммы помещаются в библиотеки подпрограмм. Развитие модульного программирование связано с именами Парнас, Коуэн, Цейтин.

**Достоинства:**

* Создание крупных проектов
* Уменьшение времени разработки
* Облегчение возможности модификации за счет использования стандартных подпрограмм

**Недостатки:**

* Глобальные данные и неограниченность доступа к ним усложняет структуру программы и приводит к ошибкам
* Трудоемкий процесс внесения любых изменений
* Не отражает картину реального мира

1.5 Объектно-ориентированное программирование

Финальная точка развития императивного программирования. Основные положения разработаны Кэйем и Ингалсом. ООП объединяет данные и методы их обработки в структуру *объекта*. Программа, разработанная на основе ООП представляет собой совокупность объектов, которые взаимодействуют посредством вызова подпрограмм друг друга.

Основное понятие ООП – *класс*. С точки зрения компилятора, это тип данных, определяемый пользователем. Он объединяет в себе данные, методы для их обработки и скрывает ненужную для использования этих данных информацию. Это называется инкапсуляцией. Это позволяет взаимодействовать с классом, зная только его интерфейс (описание), избавившись от знаний о его реализации.

(*Пример: пульт от телевизора – интерфейс класса Телевизор. Зная, как нажать кнопку на нём, можно включить нужный канал без знаний о том, что происходит внутри телевизора в этот момент)*

Конкретные величины класса называются *экземплярами класса или объектами*. У них могут быть разные данные, но методы их обработки, задаваемые самим классом – одинаковые. Поэтому для изменения метода всех объектов достаточно изменить метод в классе.

*Наследование* – способность создавать производные классы, наследующие свойства своих предков.

*Полиморфизм* – способность использовать одни и те же методы для объектов разных классов. (*Пример: в абстрактном классе «Летательный аппарат» есть метод «летать». Подклассы «самолёт» и «квадрокоптер» могут использовать общий метод «летать», будучи разными классами. При этом, в подклассе «самолёт» методу «летать» можно добавить требуемую и присущую только этому классу функциональность – «запустить двигатели и начать разгон по ВВП», а в подклассе «квадрокоптер», тому же методу «летать» - функцию «запустить лопасти». Таким образом, общий метод будет вызывать в разных классах выполнение разных цепочек действий)*

**Достоинства:**

* Логичность ввиду соответствия образу человеческого мышления (объекты, их характеристики и возможности)
* Повторное использование кода в других приложениях
* Высокая скорость разработки, за счёт первых двух пунктов. Приложение не пишется, а собирается из готовых кусков с написанием незначительных связок

**Недостатки:**

* Большие временные затраты на начальном этапе разработки
* Снижение скорости работы приложения из-за больших затрат по времени работы и объёму занимаемой памяти

# **Языки программирования высокого уровня**

Языки *низкого* уровня близки к машинному коду. Самым низким уровнем является *ассемблер*.

Языки же *высокого* уровня пишутся для человека. Для перевода их на машинный, созданы трансляторы, которые делятся на *интерпретаторы* и *компиляторы*.

*Интерпретатор* берёт очередной оператор языка, анализирует его структуру и сразу исполняет. Переход к следующему происходит только после успешного выполнения текущего оператора. Если требуется многократное выполнение одного и того же оператора, то интерпретатор выполняет его, будто встретил впервые

*Компилятор* полностью обрабатывает весь текст программы в поисках синтаксических ошибок, проводит смысловой анализ и переводит весь текст на машинный язык, нередко выполняя оптимизацию с помощью набора методов, ускоряющих работоспособность программы. Законченная программа получается более компактной, работает быстрее, чем с помощью интерпретатора и является портативной (может быть перенесена на другие компьютеры)

# **Основные элементы языка C++. Лексемы, ключевые слова, идентификаторы, константы**

К *лексемам* (минимальная единица языка, имеющая самостоятельный смысл) языка C++ относят:

* Идентификаторы
* Ключевые слова
* Знаки операций
* Константы
* Разделители (скобки, точка, запятая, пробельные символы)

По порядку про всё-всё-всё:

**Идентификаторы** – последовательность латинских букв, цифр и символов подчеркивания, обозначающая имя переменной (т.е. имя программного объекта). Прописные и строчные символы различаются.

*Примеры: i (переменная цикла), count, flag, q12, date\_of\_birth*

**Ключевые слова** – четко определенные, зарезервированные слова, имеющие конкретное значение.

**Операторы** – используются для составления арифметических и логических выражений. Имеют строгий приоритет и порядок выполнения. Делятся на арифметические, логические, побитовые, операторы отношения и присваивания.

*Примеры: +, -, %, ==*

**Константы** **(Литералы)** – постоянные значения.

*Примеры: 5, 0x5, ‘a’*

**Разделители (пунктуация)** – тривиально. Вся пунктуация: точки, запятые, пробелы, скобки.

# **Операнды, выражения, операции. Приоритет операций**

**Операнды** – данные, с которыми работает выражение. Может выступать константа (именованная или разыменованная), переменная или вызов некоторой функции

**Выражение** – последовательности операторов и операндов. Правило вычисления некоторого значения.

**Операции –** действия над данными. Для каждой операции существует свой знак. Бывают унарные (*инкремент, декремент, взятие адреса переменной*), бинарные (*||, &&*)*,* тернарные (*x ? y : z – если выполняется x, то y, иначе z*) по количеству используемых операнд. По назначению делятся на *арифметические, сравнения, логические, побитовые, специальные*.

По *приоритетности*: инкремент/декремент, арифметические, побитовый сдвиг, операторы сравнения, логические операторы. (*P.S. Более подробно в таблице. Всю учить не нужно*)

# **Понятие переменной. L- и R- концепция.**

Место, в котором хранятся данные в компьютере называется *объектом*. Именованный объект называется *переменной*. Данные, записанные в переменные, называют *значениями*.

Перед использованием любая переменная должна быть *определена*, т.е. указано её *имя* и *тип*. Желательно (но не обязательно), задавать начальные значения (*инициализировать* переменную)

Концепция L-/R- предполагает, что все используемые значения можно разделить на **L-Value** и **R-Value.** *L-Value* имеет свой адрес в памяти (все переменные являются l-value). *R-Value* – это то, что можно только присваивать (литералы или результаты выражений).

К примеру, **int n = 5;** здесь переменная n представляет lvalue, т.к. имеет выделеный адрес памяти, а число 5 – rvalue.

Также r-value может быть ссылкой.

# **Понятие типа данных. Основные типы данных. Приведение типов.**

**Тип данных** – характеристика набора данных, которая определяет:

* Характеристику переменной или константы (целое число, символ и т.д.)
* Диапазон значений
* Размер памяти занимаемой этой переменной или константой (зависит от разрядности системы)
* Операции, допустимые для данного типа.

Типы данных делятся на *встроенные* и *определённые пользователем*.

К первым относятся:

* Целочисленный – **int.** Имеет четыре спецификатора – *signed (знаковый), unsigned (беззнаковый), short, long*.
	+ Характеристика – целое число
	+ Диапазон значений зависит от размера памяти
	+ Размер памяти зависит от спецификатора. Short – 2 байта, int – 2/4 байта (в 32 и 64 разр. Системах). Long – 4 байта
	+ Определены операции сравнения, арифметические, определение остатка от деления.
* Вещественный – **float и double.** Один спецификатор *long.* Различаются кол-вом цифр после запятой.
	+ Характеристика – вещественное число
	+ Кол-во значащих цифр зависит от размера памяти
	+ Размер памяти зависит от спецификатора. Float – 4 байта, double – 8 байт, long double – 10 байт
	+ Определены операции сравнения, арифметические.
* Символьный – **char** для символов, определяемых по ASCII-таблице и **wchar\_t** для символов в другой кодировке.
	+ Характеристика – символ
	+ Диапазон для char – 256 символов, для wchar\_t определяется реализацией.
	+ Размер памяти для char – 2 байта, для wchar\_t – 4-8 байт.
	+ Определены операции сравнения.
* Логический – **bool**. Используется в логических выражениях и как результат логических операций и операций сравнения.
	+ Характеристика – истина или ложь
	+ Диапазон – **true и false**
	+ Размер памяти – 1 байт
	+ Определены логические операции и операции сравнения.

К определённым пользователем типам данных относят массивы, строки, перечисления, структуры, классы и т.д.

***Приведение типов*** делится на *явное* (выполняемое самим программистом) и *неявное*, происходящее автоматически.

*Явное* приведение типов требует записи необходимого типа и переменной в скобках. Тогда создаётся копия переменной нужного типа, существующая только во время выполнения оператора.

*Неявное* происходит в случаях *преобразования в операторе присваивания*

*В выражениях* (все переменные преобразуются к старшему типу, по иерархии: long double 🡪 double 🡪float 🡪 unsigned long 🡪 long 🡪 unsigned int 🡪 int. Тип char и short всегда приводится к int)

# **Основные конструкции языков программирования. Составные операторы. Время и область видимости переменной. Глобальные и локальные переменные.**

(*Напоминание!) Во всех языках программирования существует три базовых конструкции:*

1. *Последовательные операторы – операторы выполняются последовательно в порядке написания.*
2. *Операторы ветвления – заданное условие определяет, какое из двух действий будет выполнено*
3. *Операторы цикла – одно и то же действие выполняется несколько раз до выполнения условия.*

К *составным операторам* относятся операторы, которые включают в себя выполнение нескольких действий. Составные операторы заключаются в *операторные скобки*.

Каждая переменная в программе имеет свою *область видимости* и *время жизни*. *Время жизни* начинается с момента определения переменной и длится до её уничтожения. *Область видимости*, как правило ограничивается блоком кода, который заключён в операторные (фигурные) скобки. В зависимости от области видимости, переменные могут быть:

1. *Глобальные* (определены в файле программы вне любой функции или блока кода, могут быть доступны из любой точки программы. Существуют до завершения программы)
2. *Локальные* (По умолчанию, *автоматические,* *определённые* только внутри блока кода и доступные только в пределах его самого же.) Существует ещё подтип *статических локальных* переменных, определённых с помощью ключевого слова **static**. Их отличие от автоматических переменных во времени жизни. Статические переменные инициализируются лишь один раз и уничтожаются вместе с завершением программы.)

# **Основные конструкции языков программирования. Операторы ветвления. Примеры**

Операторы ветвления делятся на:

* *Условный оператор* (**if-else**. Позволяет выбрать из двух вариантов. Может быть неполным, если отсутствует условие else. Может быть вложенным. 
* *Оператор выбора*. (**switch**. Позволяет выбрать из *нескольких* вариантов.

Примеры использования операторов ветвления:





# **Основные конструкции языков программирования. Операторы цикла. Примеры.**

**Цикл с предусловием (while)** работает следующим образом:

1. Параметру присваивается начальное значение
2. Проверка *блока условий*, если истинно, то переход к следующему шагу. Если ложно, то к шагу 4.
3. Выполнение *блока операторов* и *изменение параметра*. Переход к шагу 2.
4. Выход из цикла

**Цикл с постусловием (do while)** работает аналогично циклу с предусловием, за исключением того, что проверка блока условий происходит после выполнения блока операторов. Таким образом:

1. Параметру присваивается начальное значение
2. Выполняется *блок операторов*, в котором *изменяется параметр*.
3. Проверка *блока условий*. Если истинно, то переход к шагу 2. Если ложно, то к шагу 4.
4. Выход из цикла

**Цикл с параметром (for)** работает по принципу цикла с предусловием:

1. Присваиваются начальные значения данным в *разделе инициализации*
2. Проверяется истинность логического выражения. Если истинно, то шаг 3. Иначе шаг 5.
3. Выполнение *блока операторов*.
4. Выполнение *блока модификаций*. Переход к шагу 2.
5. Выход из цикла.

В *разделе инициализации* присваивается одному или нескольким параметрам начальное значение. Раздел *блока модификаций* отвечает за изменение одного или нескольких параметров цикла. Любой из трёх разделов может отсутствовать, то «;» обязательны. Существуют и вложенные циклы с параметром.

Примеры кода далее:

# **Основные конструкции языков программирования. Операторы управления. Примеры.**

*(P.S. операторы управления, по лекциям Ивановой, «позволяют прервать выполнение операторов в зависимости от некоторых условий». Предполагаю, что речь идёт про операторы перехода. Про них и поговорим)*



Оператор **goto** используется крайне редко, т.к. нарушает логическую последовательность программы. Позволяет перейти на *метку* в любой части программы. В оператор передаётся название метки.

Оператор **return** служит для возврата управления на место вызова функции. Любая функция кроме типа **void** по стандартам языка должна иметь оператор **return**. Тип возвращаемого значения должен совпадать с типом функции.

Оператор **прерывания циклов break** используется при прерывании цикла, вне зависимости от логического условия. В случае вложенных циклов, прерывается только тот, который содержал в себе оператор прерывания.

Оператор прерывания **continue** прерывает только текущую итерацию цикла.

# **Составные типы данных. Указатели. Примеры**

При объявлении переменной под неё выделяется память. **Указатель** – специальный тип переменной, хранящий адрес ячейки памяти переменной. Чтобы получить адрес переменной – нужно написать символ **«&»** перед ней. Чтобы получить значение из переменной указателя (*операция* *разыменования указателя*) нужно написать символ **«\*»** перед указателем.

Как видно из примера, объявление указателя имеет следующий вид:

Указатели могут указывать на типы данных самого разного размера, при этом сами всегда занимают лишь 2-4 байта.

Способов инициализации несколько: указав конкретную ячейку памяти, использовав null-указатель, указав на уже инициализированную переменную, использовав операцию выделения памяти **new**. Последний имеет вид:



Последний вариант позволяет выделять память не на этапе компиляции, а на этапе выполнения программы. Самостоятельно выделив память, обычно, после всех проделанных операций, эта память так же вручную удаляется через операцию **delete**.

# **Составные типы данных. Одномерные массивы. Примеры.**

**Массив** – структура данных, содержащая множество значений, относящихся к одному и тому же типу. Тип может быть любым. Все элементы хранятся последовательно. Массивы и способы их задать делятся на:

В статических массивах память выделяется под конкретное число ячеек (то есть размер известен на этапе компиляции).

В динамических память выделяется на этапе работы программы. Указатель при этом указывает на нулевой элемент массива. К элементам динамического массива можно обращаться как по индексу, так и через арифметику указателей (прибавляя к указателю на нулевой элемент столько байт, сколько занимает тип данных в массиве)

Примеры:

