**1. Основные этапы решения задач на ЭВМ**

1. Постановка задачи.
2. Построение модели задачи.
3. Разработка алгоритма.
4. Кодирование (программирование).
5. Тестирование и отладка.
6. Анализ и интерпретация результатов.

**2. Последовательность этапов решения любой прикладной задачи с использованием ЭВМ**

1. Анализ задачи: выявление входных и выходных данных, ограничений и целей.
2. Построение математической или логической модели.
3. Разработка алгоритма решения задачи.
4. Реализация алгоритма в виде программы.
5. Тестирование программы на примерах.
6. Исполнение программы на реальных данных.
7. Анализ результатов и их интерпретация.

**3. Прежде, чем приступить к заключению договора по разработке программной системы, нужно:**

1. Определить требования заказчика.
2. Составить техническое задание.
3. Оценить объем работ, сроки и ресурсы.
4. Согласовать архитектуру системы и основные подходы к реализации.
5. Подготовить проект договора с учетом всех условий.

**4. На этапе постановки задачи необходимо ответить на следующие вопросы:**

1. Каковы цели и задачи проекта?
2. Какие входные данные используются?
3. Какие выходные данные ожидаются?
4. Какие ограничения существуют?
5. Какими методами будет решаться задача?

**5. Соответствие свойств модели исходному объекту характеризуется:**

* **Адекаватностью** — насколько модель точно отражает свойства реального объекта.
* **Надежностью** — способность модели давать правильные результаты в различных ситуациях.
* **Обобщаемостью** — возможность использования модели для широкого класса задач.

**6. При проектировании программного обеспечения могут использоваться следующие типы моделей:**

1. Концептуальные модели (описание задач на естественном языке).
2. Диаграммы (например, UML).
3. Математические модели.
4. Логические и структурные схемы.
5. Симуляционные модели.

**7. Модели решения задач при проектировании программного обеспечения и их особенности:**

1. **Математические модели** — используют формулы и уравнения.
2. **Алгоритмические модели** — определяют шаги решения задачи.
3. **Имитационные модели** — симуляция работы системы.
4. **Функциональные модели** — выделение функций и их взаимодействий.

**8. Точный набор инструкций (команд), описывающих последовательность действий для достижения результата — это:**

**Алгоритм.**

**9. Основные свойства алгоритма:**

1. Дискретность.
2. Определенность.
3. Конечность.
4. Результативность.
5. Массовость.

**10. Основные виды алгоритмов:**

1. Линейные.
2. Разветвляющиеся.
3. Циклические.
4. Рекурсивные.
5. Итеративные.

**11. Составление блок-схем алгоритмов регламентируется стандартом:**

**ГОСТ 19.701-90.**

**12. Базовые структуры алгоритма:**

1. Последовательность.
2. Разветвление.
3. Цикл.

**13. Название символа блок-схемы и его назначение:**

* **Овал** — начало и конец алгоритма.
* **Параллелограмм** — ввод/вывод данных.
* **Прямоугольник** — выполнение операции.
* **Ромб** — условие.

**14. Естественное направление линий передачи управления в блок-схемах:**

Слева направо или сверху вниз.

**15. Основные блоки структурного программирования:**

1. Последовательность.
2. Условие.
3. Цикл.

**16. Теорема о структурных алгоритмах:**

Любой алгоритм может быть реализован с использованием трех структур: последовательность, ветвление и цикл.

**17. Программа, которая проверяет код и выдаёт сообщения об ошибках — это:**

**Компилятор или интерпретатор.**

**18. Переход от алгоритма к программе на языке программирования называется:**

**Кодирование.**

**19. Элементарные конструкции языка программирования называются:**

**Операторы.**

**20–22. Лексемы, выражения, операторы:**

* **Лексемы** — минимальные единицы языка (ключевые слова, идентификаторы, литералы).
* **Выражения** — комбинации лексем с операциями.
* **Операторы** — инструкции для выполнения действий.

**23. Основные лексемы языка программирования:**

Ключевые слова, идентификаторы, литералы, операторы.

**24. Явно указанное в коде значение называется:**

**Константа.**

**25. Имя любого программного объекта**

**Идентификатор** — это имя, которое используется для обозначения переменных, функций, классов и других объектов программы.

**26. Спецификаторы типа данных, уточняющие внутреннее представление и диапазон значений стандартных типов**

Примеры спецификаторов:

* **short, long** — для уточнения размера целых чисел.
* **signed, unsigned** — для управления знаковостью чисел.
* **float, double** — для указания точности чисел с плавающей запятой.

**27. Простые типы данных**

1. **Целые числа**: int, short, long.
2. **Вещественные числа**: float, double.
3. **Символы**: char.
4. **Логические значения**: bool.

**28. Тип данных, который используется для определения функций, которые не возвращают значения**

**void**.

**29. Приоритеты операций в выражениях**

Приоритет определяет порядок выполнения операций. Пример (от более высокого к низкому):

1. Сначала выполняются скобки: ( ).
2. Унарные операции: -, ++, --.
3. Арифметические операции: \*, /, %, затем +, -.
4. Операции сравнения: <, >, <=, >=.
5. Логические операции: &&, ||.
6. Присваивание: =.

**30. Команды (операторы) ввода-вывода**

1. **C++:** cin, cout.
2. **C#:** Console.ReadLine(), Console.WriteLine().
3. **Python:** input(), print().

**31. Оператор разрешения области видимости**

:: — указывает, из какого пространства имен брать идентификатор. Пример (C++):

std::cout << "Hello, World!";

**32. Группа идентификаторов, внутри которой все идентификаторы уникальны**

**Пространство имен** (namespace).

**33. Преобразование типов, которое происходит автоматически**

**Неявное преобразование** (implicit conversion).  
Пример:

int a = 5;

float b = a; // Неявное преобразование int в float

**34. Пример явного/неявного преобразования типов**

* **Явное:**

float a = 5.5;

int b = (int)a; // Преобразование float в int

* **Неявное:**

int a = 5;

float b = a; // int автоматически преобразуется в float

**35. Назначение некоторых функций библиотеки math**

1. **sqrt(x)** — вычисление квадратного корня.
2. **pow(x, y)** — возведение в степень.
3. **sin(x)**, **cos(x)**, **tan(x)** — тригонометрические функции.
4. **abs(x)** — модуль числа.

**36. Основные операторы языка C#:**

1. Арифметические: +, -, \*, /, %.
2. Логические: &&, ||, !.
3. Операторы сравнения: ==, !=, <, >, <=, >=.
4. Присваивания: =, +=, -=, \*=, /=.
5. Тернарный оператор: ?:.

**37. Назначение и характеристики массива**

**Массив** — это структура данных для хранения элементов одного типа.  
**Характеристики:**

* Доступ к элементам по индексу.
* Фиксированный размер.
* Однородность (все элементы одного типа).

**38. Структура данных, позволяющая хранить под одним именем совокупность данных одного типа**

**Массив.**

**39. Массив характеризуется:**

1. Размерностью (одномерный, двумерный и т. д.).
2. Типом данных элементов.
3. Количеством элементов.

**40. Количество индексов, которое необходимо для однозначной идентификации любого элемента массива**

**Размерность массива.**

**41. Общее количество элементов в массиве**

**Размер массива.**

**42. Одномерный массив иногда называют:**

**Вектор.**

**43. Двумерный массив иногда называют:**

**Матрица.**

**44. Отличия статических и динамических массивов**

* **Статические массивы**: создаются с фиксированным размером на этапе компиляции.
* **Динамические массивы**: размер задается во время выполнения программы.

**45. Переменные, которые созданы в процессе выполнения программы, называются:**

**Динамическими переменными.**

**46. Для создания динамических переменных используется оператор:**

* **C++:** new.
* **C#:** new.

**47. Для удаления динамических переменных используется оператор:**

* **C++:** delete.
* **C#:** автоматическое управление сборщиком мусора (GC).

**48. Пример конструкции для создания двумерного динамического массива**

**C++:**

int\*\* arr = new int\*[rows];

for (int i = 0; i < rows; i++) {

arr[i] = new int[cols];

}

**49. Пример конструкции для создания одномерного динамического массива**

**C++:**

int\* arr = new int[size];

**50. Именованная последовательность описаний и операторов, выполняющая законченное действие – это:**

**Функция.**

**51. Задает имя функции, тип возвращаемого значения и список передаваемых параметров**

**Прототип функции.**

Пример (C++):

int Add(int a, int b); // Прототип функции

**52. Тело функции представляет собой:**

Набор инструкций, которые выполняются при вызове функции.

Пример (C++):

int Add(int a, int b) {

return a + b; // Тело функции

}

**53. Если функция имеет в конце тела оператор без параметров return;, то функция имеет тип:**

**void** (не возвращает значения).

**54. Тип возвращаемого значения функции может быть:**

1. **Простые типы**: int, float, char, bool и т. д.
2. **Сложные типы**: указатели, структуры, классы.
3. **void** (если функция ничего не возвращает).

**55. В прототипе функции находятся:**

1. Имя функции.
2. Тип возвращаемого значения.
3. Типы и порядок параметров.

Пример:

float Multiply(float x, float y);

**56. Глобальные и локальные переменные**

* **Глобальные переменные**: объявлены вне всех функций и доступны во всей программе.
* **Локальные переменные**: объявлены внутри функции или блока кода и доступны только в этом контексте.

**57. Область видимости переменных**

Область программы, где переменная доступна для использования.

* **Глобальная**: переменные видны во всей программе.
* **Локальная**: переменные видны только внутри функции или блока.

**58. Переменные, описанные вне функций, — это:**

**Глобальные переменные.**

**59. Переменные, описанные внутри функций, — это:**

**Локальные переменные.**

**60. При передаче в функцию параметра по значению выполняются следующие действия:**

1. Создается копия передаваемого значения.
2. Изменения внутри функции не влияют на оригинальное значение.

**61. Переменная, которая сохраняет свое значение даже после выхода из блока, в котором она определена, называется:**

**static-переменная**.

Пример (C++):

void Example() {

static int counter = 0;

counter++;

std::cout << counter << std::endl;

}

**62. Массив в качестве параметра всегда передается в функцию по:**

**Ссылке** (фактически передается указатель на массив).

Пример (C++):

void PrintArray(int arr[], int size) {

for (int i = 0; i < size; i++) {

std::cout << arr[i] << " ";

}

}

**63. Размер массива передается в функцию как:**

Отдельный параметр.  
Пример (C++):

void PrintArray(int arr[], int size) {

for (int i = 0; i < size; i++) {

std::cout << arr[i] << " ";

}

}

**64. Определение функции с переменным числом параметров**

Используется специальный синтаксис.

Пример (C++):

#include <cstdarg>

int Sum(int count, ...) {

va\_list args;

va\_start(args, count);

int sum = 0;

for (int i = 0; i < count; i++) {

sum += va\_arg(args, int);

}

va\_end(args);

return sum;

}

**65. Для функции с переменным числом параметров используется синтаксическая конструкция:**

**...** (три точки) — обозначение переменного числа параметров.

**66. Написание функции с одним и тем же именем, но с различными параметрами называется:**

**Перегрузка функции** (overloading).

Пример (C++):

int Multiply(int a, int b) {

return a \* b;

}

float Multiply(float a, float b) {

return a \* b;

}

**67. Для автоматизации создания функций, обрабатывающих разнотипные данные, используются:**

**Шаблоны (templates).**

Пример (C++):

template <typename T>

T Add(T a, T b) {

return a + b;

}

**68. Основная главная функция программы (функция main) может иметь следующие параметры:**

1. **int main()** — без параметров.
2. **int main(int argc, char\* argv[])**:
   * argc — количество аргументов командной строки.
   * argv — массив строк, содержащий аргументы.

**69. Структура — это:**

Тип данных, объединяющий несколько переменных разных типов под одним именем.

Пример (C++):

struct Point {

int x;

int y;

};

**70. Перечисления — это:**

Набор именованных целочисленных констант.

Пример (C++):

enum Color { Red, Green, Blue };

**71. Перечисления используются для:**

Упрощения работы с набором связанных значений. Например, для цветов, дней недели.

**72. Ключевое слово определения перечисления:**

**enum.**

**73. Ключевое слово определения структуры:**

**struct.**

**74. Отличия структур от классов:**

1. **Структуры**: все члены по умолчанию имеют публичный доступ.
2. **Классы**: члены по умолчанию приватные.
3. **Структуры** чаще используются для простых данных, **классы** — для сложной логики.

**75. Синтаксис определения структуры**

struct Point {

int x;

int y;

};

**76. Когда использовать структуры, а когда классы:**

* **Структуры**: для хранения простых данных.
* **Классы**: для объектов с методами и логикой.

**77. Именованная область данных на носителе информации — это:**

**Файл**.

Файлы используются для хранения данных, которые могут быть записаны или считаны программой.

**78. Режимы открытия файлов:**

1. **Чтение (read):** файл открывается только для чтения.
2. **Запись (write):** файл открывается для записи (содержимое файла перезаписывается).
3. **Добавление (append):** файл открывается для добавления данных в конец.
4. **Чтение и запись (read-write):** файл открывается для чтения и записи.

**79. Режимы доступа к файлам:**

1. **Текстовый (text):** данные читаются или записываются в текстовом формате.
2. **Бинарный (binary):** данные читаются или записываются в их двоичном представлении.

**80. Типы файлов бывают:**

1. **Текстовые файлы:** данные хранятся в виде символов.
2. **Бинарные файлы:** данные хранятся в машинно-ориентированном двоичном формате.

**81. Файл, в котором каждый символ из используемого набора символов хранится в виде одного байта (кода, соответствующего символу), — это:**

**Текстовый файл.**

**82. Файл, в котором символы и числа записываются в виде последовательности байт (в своем внутреннем двоичном представлении в памяти компьютера), — это:**

**Бинарный файл.**

**83. Для работы с файлами необходимо подключить библиотеки:**

* **C++:** <fstream> для работы с файлами.
* **C#:** System.IO для работы с файлами.
* **Python:** встроенная библиотека open().

**84. Функции работы с файлами:**

1. **C++:**
   * ifstream — для чтения.
   * ofstream — для записи.
   * fstream — для чтения и записи.
2. **Python:**
   * open(file, mode) — открытие файла.
   * read(), write() — чтение и запись.
3. **C#:**
   * File.ReadAllText(), File.WriteAllText().

**85. Функция записи байтов в поток:**

* **C++:** write() (для бинарного файла).  
  Пример:

ofstream file("data.bin", ios::binary);

int num = 42;

file.write(reinterpret\_cast<char\*>(&num), sizeof(num));

**86. Функция чтения байтов из потока:**

* **C++:** read() (для бинарного файла).  
  Пример:

ifstream file("data.bin", ios::binary);

int num;

file.read(reinterpret\_cast<char\*>(&num), sizeof(num));

**87. Для определения числа байт в переданном аргументе используется оператор:**

**sizeof.**

Пример (C++):

int a = 42;

std::cout << sizeof(a); // Вернет размер переменной a в байтах

Явно указанное в исходном коде программы значение определенного типа называется -Общее количество элементов в массиве – это: