1. Язык программирования Си. История создания. Область применения. Характеристики

1. История создания языка Си, стандарты

Язык программирования Си разрабатывался в период с 1969 по 1973 годы в лабораториях Bell Labs Керниганом и Ритчи

В 1978 году Брайан Керниган и Деннис Ритчи опубликовали первую редакцию книги «Язык программирования Си». Эта книга, известная среди программистов как «K&R». Были введены: структуры, длинное целое, беззнаковое целое, оператор '+=' и подобные

Появляется множество компилятор, поэтому в какой-то момент, из-за пробелов в стандарте "K&R", эти стандарты перестают быть совместимы
В 1983 году Американский национальный институт стандартов (ANSI) сформировал комитет для разработки стандартной спецификации Си. По окончании этого долгого и сложного процесса в 1989 году он был наконец утверждён как «Язык программирования Си» ANSI X3.159-1989. Эту версию языка принято называть ANSI C или C89. Были введены: прототипы функций, пропроцессы
В 1990 был принят Международной организацией по стандартизации (ISO). Он получил название C90. Стандарты C90 и C89 почти ничем не отличаются

В конце 1990-х годов стандарт подвергся пересмотру, что привело к публикации ISO 9899:1999 в 1999 году. Этот стандарт обычно называют «C99». В марте 2000 года он был принят и адаптирован ANSI. Этим стандартом мы пользуемся
Были введены: новые типы данных, встраиваемые функции, массивы переменной длины
Новые типы вводят, так как начинают распространятся 64-битные системы и появляется необходимость поддержать типы данных для этих систем

8 декабря 2011 опубликован новый стандарт для языка Си (ISO/IEC 9899:2011)
Были введены: многопоточность, улучшенная поддержка UNICODE

Черновой вариант стандарта был представлен как C17 (ISO/IEC 9899:2017) в 2017 году. В июне 2018 года стандарт был опубликован как C18 (ISO/IEC

9899:2018). Новый стандарт устраняет дефекты, замеченные в предыдущей версии, без добавления новых возможностей. Названия C17 и C18 обычно упоминаются как синонимы

C23 - это неофициальное название для следующей (после C17) основной редакции стандарта языка C. На протяжении большей части своего развития он был неофициально известен как "C2X". C23 был опубликован в начале 2024 года как ISO/IEC 9899:2024
Были введены: новые ключевые слова, удалены триграфы, новые директивы препроцессора, изменения в стандартной библиотеке

2. Области использования языка Си

Язык Си используется в следующих областях:
1. Разработка операционных систем: ядра (почти полностью/частично) - UNIX, Linux, Windows, MacOS, Android
2. Разработка встраиваемых систем (Встраиваемые системы - специализированные системы, предназначенные для работы в устройстве, которым они будут управлять)
3. Разработка высокопроизводительных систем: Oracle (с использованием Си), SQL Server(с использованием Си), MySQL(с использованием Си), PostgreSQL, Mathematica, MATLAB
4. Компиляторы, интерпретаторы: первые реализации - Python, Ruby, Perl, Java
5. ПО с открытым кодом

Одной из причин широкого распространения для программирования на низком уровне является возможность писать кроссплатформенный код, который может по-разному обрабатываться на разном оборудовании и на разных операционных системах
Также язык Си дает большую свободу программисту, что позволяет писать оптимизированный и быстрый код

3. Основные черты языка Си

1) Язык Си является компилируемым - преобразует исходный код на машинный язык без выполнение программы
2) Имеет статическую типизацию - типы известны на этапе компиляции
3) Типизация слабая, т.к. есть неявные приведения типов

4) Ручное освобождение памяти - при работе с ресурсом нужно сначала его инициализировать, потом обработать, а потом освободить память
5) Язык сравнительно низкого уровня - несмотря на то, что является высокоуровневым (введены абстракции для упрощения написания кода) не скрывает доступ к абстракциям машинного уровня (байты, адреса и т.д.)
6) Мало ключевых слов по сравнению с другими языками, много функций вынесено в стандартную библиотеку + компилятор однопроходный - компиляторы легко разрабатывать -> язык доступен на многих платформах
7) Программист знает, что делает - позволяет избежать лишних проверок и получить в результате трансляции оптимальную программу, но больше ответственности
8) Ориентирован на процедурное программирование - программирование, при котором последовательно выполняемые операторы можно собрать в подпрограммы
9) Передача параметров по значению

2. Сравните язык C и язык Python

4. Сравнение языка Си и языка Python

Python	C
интерпретируемый - построчный анализ, обработка и выполнение кода	компилируемый - преобразует исходный код на машинный язык без выполнения программы
динамическая типизация - все типы выясняются во время выполнения программы	статическая типизация - типы известны на этапе компиляции Статическая типизация позволяет избежать ошибок в программе уже на этапе компиляции
автоматическое управление памятью - язык сам распоряжается переменной	ручное управление памятью - при работе с ресурсом нужно сначала его инициализировать, потом обработать, а потом освободить память
строгая типизация	нестрогая типизация
является объектно-ориентированным - все является объектами	является процедурным - программирование, при котором последовательно выполняемые операторы можно собрать в подпрограммы
передача по ссылке для изменяемых типов данных передача по значению для неизменяемых типов данных	передача параметров по значению есть указатели имитирующие передачу по ссылке
начало запускаемого модуля - первая строка программы или конструкция `if __name__ == '__main__':`	имеет единую точку входа

Плюсы Python: код писать легче из-за легкого синтаксиса, какие-то программы можно быстро реализовать для MVP, прост для усвоения, поддерживается большое количество библиотек с различным функционалом

Плюсы C: более оптимизирован, быстрее работает, имеет лучшую переносимость, большая свобода позволяет создавать более низкоуровневое ПО, подходит для более крупных проектов

3. Синтаксис и семантика языка программирования. Состав языка на примере языка Си

5. Структура программы на языке Си

Программа на языке Си состоит из набора инструкций, директив препроцессора, функций

Инструкция - простейший строительный элемент в Си, она оканчивается “;”
Пример:

```
printf("Hello!");
```

Директивы препроцессора представляют собой инструкции, записанные в тексте программы на СИ, и выполняемые до трансляции программы
Начинаются со знака #, заканчиваются переводом на новую строку
Известные нам директивы:
#include - включает текст программы в содержание программы
#define - создание константы или макроса
#undef - отменяет действие директивы #define
#if, #ifdef, #elif, #else, #endif - условная компиляция

Функцией в языке С называется подпрограмма, тип возвращаемого значения которой не является void

В каждой программе должна присутствовать *точка входа* в программу. В Си ей является функция *main()*

6. Дайте определение алфавиту языка. Алфавит языка Си

Алфавитом называется конечное непустое множество символов языка, которые могут использоваться при написании программы

Алфавит языка С включает:

- латинские буквы [a-zA-Z]
- цифры [0-9]
- 29 графических символов ! " # % & ' ( ) * + и др.
- пробельные символы

7. Дайте определение лексеме. Правило выделения лексем в языке Си

Лексема - минимальная единица языка

К лексемам языка С относятся:
- идентификаторы
- ключевые слова
- константы
- строковой литерал
- пунктуаторы (знаки пунктуации)

Пунктуаторы (знаки пунктуации) - символ, имеющий независимое синтаксическое и семантическое значение. В зависимости от контекста он определяет операцию для выполнения

Литерал - последовательность символов, заключенная в двойные кавычки

Правило выделения лексем в языке Си:
Каждый непробельный символ добавляется к считаемому токену до тех пор, пока он не станет корректным

8. Что такое идентификатор? Правила формирования идентификаторов в языке Си

Идентификатор - слово, используемое для идентификации сущности

Идентификатор в языке С может содержать только:
- символы латиницы [a-zA-Z] , регистр имеет значение
- цифры не в начале слова [0-9]
- символ нижнего подчеркивания ‘_’

Идентификатор:
- чувствителен к регистру
- ограничений по количеству символов нет
- первый символ должен быть не число
- нельзя использовать зарезервированные слова для названия идентификатора и названия функций из библиотек или файла
- идентификатор должен отражать назначение сущности, которой он присваивается

9. Что такое ключевое слово?

Ключевые слова - зарезервированные слова, которые имеют специальное значение для интерпретатора языка программирования

К ключевым словам языка С относятся:
auto, continue, extern, int, signed, union, _Complex, break, default, float, long, sizeof, unsigned, _Imaginary, case, do, for, register, static, void, char, double, goto, restrict, struct, volatile, const, else, if, return, switch, while, continue, enum, inline, short, typedef, _Bool

Ключевое слово называется *зарезервированным*, если синтаксис запрещает его использование в качестве идентификатора, определяемого программистом

10. Что такое предопределенное имя?

Предопределенным именем называется идентификатор, не являющийся ключевым словом, использование которого в качестве идентификатора, определяемого пользователем, неопределено

Предопределенными считаются следующие имена:
- идентификаторы, начинающиеся с двойного подчеркивания
- идентификаторы, начинающиеся с подчеркивания и заглавной буквы
- в глобальном пространстве имен, идентификаторы, которые начинаются с подчеркивания
- все внешние идентификаторы, определенные стандартной библиотекой

11. Что такое константа?

Константой называется неизменяемая величина.

Константы в языке С
В языке С константой называется выражение, результат которого известен на момент-трансляции и неизменяемое во время выполнения

Могут быть представлены:
- в *целочисленном* типе в разных СС - 3 [10], 0100 [8], 0xFF [16], 1234u [unsinged], 0L [long], 10UL [unsinged long]
- *вещественном* типе - по умолчанию тип double, 123.321, .012, 1.2345E-20, 123.321f [float], 0.012L [long double]

- *символьном* типе - 's', '3', 'A', 'Z', '\a', '\v'
- *строковой* тип - последовательность символов, заключённая в кавычки: «Hello, World!!!»

Пример констант в коде программы на Си:

```
#define MAGIC_NUMBER 42

int main(void)
{
    printf("The Answer to the Ultimate Question of Life, the
Universe, and       Everything is %d", MAGIC_NUMBER);
    return 0;
}
```

В данном примере константами являются:
- число 42
- строка The Answer ... is %d (строковый константа);
- число 0

12. Что такое синтаксис языка программирования? Способы описания синтаксиса

Синтаксисом языка программирования называется набор правил, описывающий комбинации символов алфавита языка, считающиеся правильно структурированной программой (документов) или ее фрагментом

Способы описания синтаксиса языка:
- формальные грамматики (Керниган Б.У., Ритчи Д.М. «Язык программирования С» Приложение А)
- формы Бекуса-Наура (BNF)
- расширенные формы Бекуса-Наура (ISO 14977)
- диаграммы Вирта

13. Что такое семантика языка программирования? Способы описания семантики

Семантикой языка программирования называется набор правил придания смысла синтаксически правильным программам. В конечном счете определяет последовательность действий вычислительной машины

Способы описания семантики:

- *Операционная семантика* описывает, как программы выполняются на абстрактной машине. Это как если бы вы следили за каждым шагом вашего кода, как он исполняется в компьютере
- *Денотационная семантика* связывает каждую часть программы с математическим объектом, представляющим ее значение. Это как перевод вашего кода на язык математики
- *Аксиоматическая семантика* использует логические утверждения для описания, как состояние программы изменяется во время ее выполнения. Это похоже на использование правил для предсказания поведения вашего кода
- *Трансляционная семантика* — описание операционной семантики конструкций в терминах языков программирования высокого уровня. С помощью этого способа можно изучать язык, схожий с уже известным программисту

4. Переменные и простейшие операции

14. Что такое переменная?

Переменной в языке С называется именованный участок памяти, обладающий некоторым типом

Определение переменной:

```
тип  идентификатор;
```

15. Атрибуты переменной

Переменная характеризуется:

- именем (идентификатором)
- адресом
- типом
- значением
- область видимости
- время жизни
- связывание

Идентификатор (имя) - слово, используемое для идентификации сущности
Адрес переменной – это ячейка памяти, с которой связана данная переменная
Тип переменной определяет диапазон значений, которые может принимать переменная, и набор операций, предусмотренных для этого типа
Значение переменной – это содержимое ячейки или ячеек памяти, связанных с данной переменной
Область видимости - это часть текста программы, в пределах которой имя может быть использовано
Время жизни - интервал времени выполнения программы, в течение которого «программный объект» существует
Связывание определяет область программы (функция, файл, вся программа целиком), в которой «программный объект» может быть доступен другим функциям программы

16. Описание переменной на языке Си

Определение переменной – это оператор программы, перечисляющий имена переменных и устанавливающий, что они имеют определенный тип

Определение переменной:

```
тип  идентификатор;
```

Переменные в программе на языке Си *не инициализируются по умолчанию нулями*, как во многих других языках программирования (например, в Питоне или Паскале). После определения переменной в ней хранится «мусор» - случайное значение, которое осталось в той области памяти, которая была выделена под переменную. Это связано, в первую очередь, с оптимизацией работы программы: если нет необходимости в инициализации, то незачем тратить ресурсы для записи нулей

Если несколько переменных имеют один и тот же тип, их определения можно объединять

```
int width, length;
```

28. Операции присваивания в языке Си

Операция присваивания

```
v = e
```

Формально алгоритм работы этой операции можно описать следующим образом

1. Вычислить l-значение первого операнда (v) выражения
2. Вычислить r-значение второго операнда (e) выражения
3. Присвоить вычисленное r-значение вычисленному l-значению объекта данных
4. Возвратить вычисленное r-значение как результат выполнения операции

Если v и e имеют разный тип, значение e преобразуется к типу v

l-value - выражение с типом объекта или неполным типом, отличным от void, слева от оператора присваивания
r-value - объект, который не имеет идентифицируемого местоположения в памяти, справа от оператора присваивания

Назначение оператора присваивания:

- изменение значение объекта данных через l-value: `a = b`
- функция, возвращающая значение через r-value: `a = b = c = 5`

Операция присваивания правоассоциативна

30. Арифметические операции в языке Си

Арифметические операции

Операция	Нотация	Класс	Приоритет	Ассоциативность	Комментарии
++	x++	Постфиксные	16	правая	обладают побочным эффектом
–	x–				
++	++x	Префиксные	15	левая	
–	–x				
(унар) +	+x	Префиксные	15	правая	ничего не делает
(унар) -	-x				возвращает противоположный знак
*	x * y	Инфиксные	13	левая	
/	x / y				целые - целая часть вещественные - деление один с “-” - округление вверх
%	x % y				один с “-” - округление вверх
+	x + y		12		
-	x - y				

5. Функция printf, алгоритм работы, использование, частые ошибки при использовании

17. Функция printf. Строка форматирования. Ошибки при использовании функции printf

Для вывода значений переменных обычно используется функция printf, определенная в стандартной библиотеке stdio.h

```
#include <stdio.h>
int printf(const char * restrict format, ...);
```

Возвращаемое значение: количество прочитанных символов или отрицательное значение, если произошла ошибка

Функция printf предназначена для отображения содержимого строки, которая называется *строкой форматирования*

```
printf("My favorite numbers are %d and %f\n", i, f);
```

Спецификаторы начинаются с символа “%”. В строке форматирования спецификатор обозначает место, в которое будет выведено соответствующее значение во время отображения строки. Информация, которая указывает за символом “%”, показывает каким образом конвертируется значение из внутреннего представления в отображение

Примеры спецификаторов:

Спецификатор	**Как соответствующий аргумент отображается**
%d	как int
%f	как float
%e	как вещественное число в экспоненциальной форме
%g	как в формате f или в формате e (в зависимости от того, что короче)
%c	как char
%s	как строка
%u	как беззнаковое

%lf	как double
%p	как указатель
и т.д	

Escape-последовательность – набор символов, которые в сочетании с условным знаком служат для задания форматирования вывода и печати специальных символов. В языке Си в качестве условного знака esc-последовательности используется обратный слеш (\)

Esc-последовательность	Отображение
\a	Звуковой сигнал
\r	Возврат каретки “return”
\t	Горизонтальная табуляция
\n	Новая строка
\\	Обратный слеш
\”	Двойная кавычка
\v	Вертикальная табуляция

Основные ошибки при использовании функции printf

- несоответствие количества спецификаторов и количества переменных
- несоответствие спецификатора и типа переменной

Чтобы отслеживать такие ошибки, нужно использовать спецификатор “-Werror”

```
// Переменных меньше, чем спецификаторов
printf("%d %d\n", i);
// Переменных больше, чем спецификаторов (не критично)
printf("%d\n", i, i);
// Спецификаторы не соответствуют типам переменных
printf("%d %f\n", f, i);
```

6. Функция scanf, алгоритм работы, использование, частые ошибки при использовании

18. Функция scanf. Строка форматирования. Ошибки при использовании функции scanf

Для ввода значений переменных обычно используется функция scanf, определенная в стандартной библиотеке stdio.h

```
#include <stdio.h>
int scanf(const char * restrict format, ...);
```

Возвращаемое значение: значение макроса EOF, если перед любым преобразованием происходит сбой ввода, в противном случае количество полей, значения которых были действительно присвоены переменным, которое может быть меньше, чем предусмотрено, или даже нулем в случае раннего сбоя сопоставления

Работа функции scanf управляется строкой форматирования

```
scanf("%d%f", &i, &f);
```

Строка форматирования этой функции может содержать *как обычные символы, так и спецификаторы*

Спецификаторы начинаются с символа “%”. В строке форматирования спецификатор обозначает место, в которое будет выведено соответствующее значение во время отображения строки. Информация, которая указывает за символом “%”, показывает каким образом конвертируется значение из внутреннего представления в отображение

Примеры спецификаторов:

Спецификатор	**Как соответствующий аргумент отображается**
%d	как int
%f	как float
%e	как вещественное число в экспоненциальной форме
%g	как в формате f или в формате e (в зависимости от того, что короче)

%c	как char
%s	как строка
%u	как беззнаковое
%lf	как double
%p	как указатель
и т.д	

Escape-последовательность – набор символов, которые в сочетании с условным знаком служат для задания форматирования вывода и печати специальных символов. В языке Си в качестве условного знака esc-последовательности используется обратный слеш (\)

Esc-последовательность	Отображение
\a	Звуковой сигнал
\r	Возврат каретки “return”
\t	Горизонтальная табуляция
\n	Новая строка
\\	Обратный слеш
\”	Двойная кавычка
\v	Вертикальная табуляция

После вызова функция scanf начинает анализировать строку форматирования *слева направо*. Для каждого спецификатора scanf пытается выделить данные соответствующего типа во входных данных, *пропуская пробельные символы*. scanf читает выделенное значение, *останавливаясь на символе*, который не относится к очередному вводимому значению. Если значение было успешно прочитано, scanf *продолжает* обрабатывать оставшуюся часть строки. Если же значение не соответствует указанному типу, scanf немедленно *прекращает работу* без анализа оставшейся части строки

Когда функция scanf обнаруживает один или несколько пробельных символов в строке форматирования, она читает один или несколько символов из входных данных пока не будет обнаружен символ отличный от пробельного. *Один пробельный символ в строке форматирования соответствует 0 или более пробельным символам во входных данных.*

Если же scanf обнаруживает в строке форматирования символ отличный от пробельного, она сравнивает его с символом из входных данных. *Если они совпадают, она пропускает эти символы. Если символы не совпадают, она возвращает прочитанный символ обратно и прекращает свою работу.*

// пробельный символ в строке scanf("%d %d", &i, &j); 4SPACE5ENTER **// ОК** 4TAB5ENTER **// ОК** 4ENTER5ENTER **// ОК**	**// обычный символ в строке** scanf("%d,%d", &i, &j); 4,5ENTER **// ОК** 4,TAB5ENTER **// ОК** 4ENTER,5ENTER **// ОШИБКА** 4n5ENTER **// ОШИБКА**

Основные ошибки при использовании функции printf

- несоответствие количества спецификаторов и количества переменных
- несоответствие спецификатора и типа переменной
- аргументами функции scanf являются адреса переменных, а не сами переменные

Чтобы отслеживать такие ошибки, нужно использовать спецификатор “-Werror”. Ошибки использования функции scanf могут привести к *неожиданному* вводу значения переменной и к *аварийному завершению* программы

```
// Переменных меньше, чем спецификаторов
scanf("%d%d", &i);
// Переменных больше, чем спецификаторов (не критично)
scanf("%d", &i, &i);
// Спецификаторы не соответствуют типам переменных
scanf("%d%f", &f, &i);
// ОШИБКА: забыт символ & перед именем переменной!
scanf("%d", i);
```

7. Операции и выражения. Определения, основные понятия, особенности реализации в языке Си

19. Что такое выражение? Что такое операция?

Выражение - простейшее средством описания действий. Выражение задает действия, которые вычисляют единственное значение. Состоит из констант, переменных, а также знаков операций и скобок.

Выражение – это последовательность операций и операндов, которая:
- указывает как вычислить значение
- обозначает объект или функцию
- порождает побочный эффект
- выполняет комбинацию вышеперечисленного

Операция - конструкция в языках программирования для записи некоторых действий. Элементы данных, к которым применяется операция, называют *операндами*

20. Что такое побочный эффект операции?

Некоторые операции имеют *побочный эффект*. При выполнении этих операций, кроме основного эффекта - вычисления значения - происходят изменения объектов или файлов. Таковы, например, операции присваивания. Значением выражения a = b будет значение переменной b и в качестве побочного эффекта это значение будет присвоено переменной a

Также иногда может возникать *неопределенное поведение*, так как порядок вычисления подвыражений неопределен
Например

```
i * j + (j++) + (--i)
```

Выражение может принимать различные значения при обработке разными компиляторами

21. Классификация операций в языке Си по способу записи операции

В языке Си все операции подразделяются *по способу записи*

Вид	**Написание**	**Пример**
Постфиксные	<операнд><операция>	array[i]

Префиксные	<операция><операнд>	&speed
Инфиксные	<операнд><операция><операнд>	width * length

22. Классификация операций в языке Си по количеству операндов

В языке Си все операции подразделяются *по количеству операндов*

Вид	**Количество операндов**	**Пример**
Унарные	1	&speed
Бинарные	2	a + b
Тернарные	3	a > b ? a : b

23. Что такое приоритет операции?

Порядок выполнения операций определен приоритетом и ассоциативностью операции

Величина, определяющая преимущественное право на выполнение той или иной операции, называется *приоритетом*
Если операции имеют различный приоритет, сначала выполняется операции с *наибольшим приоритетом*

Эти правила во всех распространенных языках программирования практически идентичны, потому что основаны на соответствующих математических правилах

24. Что такое ассоциативность операции?

Ассоциативность операций - свойство операций, позволяющее восстановить последовательность их выполнения при отсутствии явных указаний на очерёдность при равном приоритете

Существуют операции:
- *Левоассоциативные* - выполняются слева направо
- *Правоассоциативные* - выполняются справа налево

Изменить приоритет и правила ассоциативности программист может, разместив в выражении скобки. Заключенная в скобки часть выражения имеет приоритет выше, чем свободная

25. Порядок вычисления подвыражений в языке Си.

Порядок вычисления операндов в общем случае не определен. Для логических операций определена ленивая логическая схема

```
f1() + f2() + f3() // f1(), f2(), f3() могут вычислены в любом порядке
```

При повторной обработке порядок вычисления подвыражений может *измениться*

Так как порядок вычисления операндов не определен, то в некоторых ситуациях возникает *неопределенное поведение* и ничего не гарантирует в таких случаях (программа может не компилироваться, работать неправильно, работать правильно, не работать и т.п.)
Примеры

```
a = 5;
c = (b = a + 2) - (a = 1);

i = 2;
j = i * i++;
```

26. Порядок вычисления логических выражений в языке Си.

Логические выражения вычисляются *слева направо* и применяется сокращенная логическая схема вычислений

27. Полная и сокращенная схемы вычисления логических выражений.

Полная схема вычислений - все выражения вычисляются полностью
Сокращенная схема вычисления логических операций - вычисления прекращаются, как только становится понятным значение всего выражения

Пример

```
expression_1 && expression_2
```

если подвыражение expression_1 ложно, то нет никакой необходимости вычислять значение подвыражения expression_2, потому что значение всего выражения ложно

8. Операция присваивания, арифметические операции, побочный эффект и неопределенное поведение в операциях

28. Операции присваивания в языке Си.

30. Арифметические операции в языке Си.

31. Операции инкремента и декремента в языке Си.

Инкремент “++” - увеличение операнда на 1
Декремент “--” - уменьшение операнда на 1

Обладают побочным эффектом - изменяют значения операндов

++i - возвращает i + 1, затем увеличивает i на 1
i++ - возвращает i, затем увеличивает i на 1
–i - возвращает i - 1, затем уменьшает i на 1
i– - возвращает i, затем уменьшает i на 1

32. Операции сравнения в языке Си.

Операция	Нотация	Класс	Приоритет	Ассоциативность
<	x < y	Инфиксные	10	левая
<=	x <= y			
>	x > y			
>=	x >= y			
==	x == y		9	
!=	x != y			

Возвращают 0, когда операция ложна, иначе 1
Типы операндов могут отличаться, тип неявно приводится

Не проверяет, что j лежит между i и k

```
i < j < k ⇔ (i < j) < k
```

20. Что такое побочный эффект операции?

34. Какие операции в языке Си обладают побочным эффектом?

Операции: присваивание, инкремент, декремент

9. Логические операции, побитовые операции

32. Операции сравнения в языке Си.

33. Логические операции в языке Си.

Значениями логических операция являются целые числа 0 («ложь») и 1 («истина»). Операнды этих операций часто имеют такие же значения, но это не обязательно: логические операции интерпретируют любое число, отличное от нуля, как «истину», а 0 – как «ложь».

Операция	Нотация	Класс	Приоритет	Ассоциативность	Комментарии
!	!x	Префиксные	15	правая	1, если x = 0
&&	x && y	Инфиксные	5	левая	1, если значения обоих выражений не 0
||	x || y		4		1, если значение хотя бы одного значения не 0

65. Логический тип в языке Си (до и после стандарта C99)

До стандарта c99 в c89 логический тип – int: 0 – ложь (false), не 0 – истина (true). Отдельного типа нет
Логический тип добавили в стандарте c99, он назывался _Bool

```
_Bool flag;
```

_Bool – это беззнаковый целочисленный тип, который может принимать только значения 0 и 1. Попытка присвоить отличное от нуля значение переменной типа _Bool приведет к тому, что значение этой переменной станет равным 1

Переменные типа _Bool могут использоваться в арифметических выражениях (но это не желательно). Стандарт c99 предоставляет заголовочный файл stdbool.h, который содержит макрос bool для обозначения _Bool, а также макросы true и false

```
#include <stdbool.h>
...
```

```
{
    bool flag;
    flag = true;
```

26. Порядок вычисления логических выражений в языке Си.

27. Полная и сокращенная схемы вычисления логических выражений.

Побитовые операции

Битовые операции позволяют оперировать отдельными битами или группами битов, из которых формируются байты

Битовые операции применимы только к целочисленным переменным. Обычно их выполняют над беззнаковыми целыми, чтобы:

- повысить переносимость программы
- не было путаницы с битом, который отвечает за знак

Операция	Нотация	Название	Класс	Приоритет	Ассоциативность
~	~x	дополнение	Префиксные	15	правая
<<	x << y	сдвиг влево	Инфиксные	11	левая
>>	x >> y	сдвиг вправо			
&	x & y	побитовое “и”		8	
^	x ^ y	побитовое исключающее “или”		7	
|	x | y	побитовое “или”		6	
<<=	x <<= y	присваивание со сдвигом влево		2	правая
>>=	x >>= y	присваивание со сдвигом вправо			

&=	x &= y	присваивание с побитовым “и”			
^=	x ^= y	присваивание с побитовым исключающим “или”			
|=	x |= y	присваивание с побитовым “или”			

Операции сдвига

Операции сдвига изменяют представление целого числа, сдвигая его биты вправо или влево

a << n - сдвига бит числа a влево на n позиций. На каждый бит, который «выдвигается» слева из числа a, бит со значением 0 «вдвигается» справа в число a

a >> n - сдвига бит числа a вправо на n позиций. Если a > 0 или unsigned, то на каждый «выдвигаемый» справа бит добавляется бит со значением 0 слева; если a < 0, то результат зависит от реализации

```
unsigned char a = 13; // a = 0000 1101
unsigned char n = 2, res;

res = a << n; // res = 0011 0100 (52)
res = a >> n; // res = 0000 0011 (3)
```

Побитовые операции

Операция «битовое дополнение» является унарной, остальные битовые операции – бинарные.

Операции ~, &, ^ и | выполняют соответствующие булевы операции над всеми битами своих операндов

```
unsigned char a, b, c;
a = 21; // 0001 0101
b = 56; // 0011 1000
```

```
c = ~a; // 1110 1010 (234)
c = a & b; // 0001 0000 (16)
c = a ^ b; // 0010 1101 (45)
c = a | b; // 0011 1101 (61)
```

Побитовые операции удобно использовать для манипуляции над битами числа: установка очистка, проверка значения

```
// Установка
unsigned char a;
a = 0x00; // 0000 0000
a |= 0x10; // 0001 0000

unsigned char a = 0x00, n = 4;
a |= (1 << n);

// Очистка
unsigned char a;
a = 0xFF; // 1111 1111
a &= ~0x10; // 1110 1111

unsigned char a = 0xFF, n = 4;
a &= ~(1 << n);

// Проверка значения
if (a & 0x10) …
if (a & (1 << n)) …
```

10. Оператор выражения, условный оператор, составной оператор, оператор выбора

35. Что такое оператор?

Оператором - синтаксическая единица языка, обозначающее действие, которое требуется выполнить

36. Пустой оператор.

Пустым оператором в языке С называется оператор, состоящие исключительно из символа “;” . Ничего не делает

Основная «специализация» пустого оператора – реализация циклов с пустым телом

37. Оператор-выражение.

Оператор-выражение - выражение с точкой запятой на конце

```
i++;
```

Возвращаемые значения выражения отбрасываются, после выполнения оператора управление переходит к следующему оператору

Если оператор не имеет никакого побочного эффекта, то он бесполезен и компилятор обычно указывает на ошибку: `statement with no effect [-Werror=unused-value]`

Точка с запятой является элементом оператора и его завершающей частью

38. Составной оператор.

Составной оператор (или блок) группирует несколько операторов и объявлений путем их размещения в фигурных скобках в единый оператор

```
{
    [список объявлений и операторов]
}
```

В отличие от оператора выражения составные операторы не завершаются точками с запятой

39. Условный оператор.

Условный оператор позволяет программе сделать выбор между двумя альтернативами, проверив значение выражения

```
// Неполный
if (выражение)
    оператор

// Полный
if (выражение)
    оператор_1
else
    оператор_2
```

Скобки вокруг выражения обязательны, они являются частью самого условного оператора. Часть else не является обязательной

В языке Си проблема решается благодаря тому, что else всегда связывается с ближайшим предыдущим оператором if без else. Например, тут else связан с - `if (выражение_1)`

```
if (выражение_1)
    if (выражение_2)
        оператор_1
else
    оператор_2
```

Для реализации выбора из нескольких альтернатив можно использовать *каскадный условный оператор*

```
if (выражение_1)
    оператор_1
else if (выражение_2)
    оператор_2
...
else if (выражение_n)
    оператор_n
else
    оператор
```

40. Условная операция.

Условная операция (или тернарная операция) - сокращенная форма if-else

```
выражение_1 ? выражение_2 : выражение_3
```

Сначала вычисляется выражение выражение_1. Если оно отлично от нуля, то вычисляется выражение выражение_2, и его значение становится значением условной операции. Если значение выражение выражение_1 равно нулю, то значением условной операции становится значение выражения выражение_3

41. Оператор выбора.

Оператор выбора - оператор, позволяющий принять многовариантное решение. В языке Си это оператор switch

В общей форме оператор switch может быть записан следующим образом

```
switch (выражение)
{
    case констнатное_выражение : операторы
    ...
    case констнатное_выражение : операторы
    default : операторы
}
```

Особенности оператора switch:

- Управляющее выражение, которое располагается за ключевым словом switch должно быть целочисленным
- Константное выражение – это обычное выражение, но оно не может содержать переменных и вызовов функций
- После каждой case-метки может располагаться любое число операторов. Никакие скобки не требуются. Последним оператором в группе таких операторов обычно бывает break. Выполнение оператора break «внутри» оператора switch передает управление за оператор switch. Если бы break отсутствовал, то стали бы выполняться операторы, расположенные в следующих case-метках
- case-метки не могут быть одинаковыми
- Порядок case-меток (даже метки default) не важен
- case-метка default не является обязательной

- Только одно константное выражение может располагаться в case-метке, но несколько case-меток могут предшествовать одной и той же группе операторов

```
switch (mark)
{
    case 1:
    case 2: printf("Passing\n");
        break;
}
```

46. Оператор break.

Оператор break может использоваться для принудительного выхода из циклов while, do-while и for. Выход выполняется из ближайшего цикла или оператора switch

```
#include <stdio.h>

int main(void)
{
    int d, n = 17;
    for (d = 2; d < n; d++)
        if (n % d == 0)
            break;

    if (d < n)
        printf("%d is divisible by %d\n", n, d);
    else
        printf("%d is prime\n", n);
    return 0;
}
```

47. Оператор continue.

Оператором continue в языке С называется оператор, выполняющий функцию передачи управления за конец тела ближайшего оператора while, оператора do while или оператора for

```
#include <stdio.h>

int main(void)
```

```
{
    int n, i, sum;
    i = 0;
    sum = 0;
    while (i < 10)
    {
        scanf("%d", &n);
        if (n <= 0)
            continue;
        sum += n;
        i++;
    }

    printf("sum is %d\n", sum);
    return 0;
}
```

11. Операторы цикла while, do-while. Операторы break, continue, goto

Цикл в программировании - оператор языка программирования, позволяющий многократно повторять одну и ту же последовательность команд

42. Оператор цикла while.

Оператор while реализует цикл с предусловием

```
while (выражение) оператор
```

Скобки вокруг выражения являются обязательными. Оператор после скобок представляет собой тело цикла

Выполнение оператора while начинается с вычисления значение выражения. Если оно отлично от нуля, выполняется тело цикла, после чего значение выражения вычисляется еще раз. Процесс продолжается в подобной манере до тех пор, пока значение выражения не станет равным 0
Пример

```
#include <stdio.h>

int main(void)
{
    int sum, i, n = 5;
    i = 1;
    sum = 0;
    while (i <= n)
    {
        sum += i;
        i++;
        // Можно обойтись одним оператором: sum += i++;
    }
    printf("Total of the first %d numbers is %d\n", n, sum);
    return 0;
}
```

45. Оператор цикла do-while.

Оператор do-while реализует цикл с постусловием

```
do оператор while (выражение);
```

Скобки вокруг выражения являются обязательными. Оператор после скобок представляет собой тело цикла

Выполнение оператора do-while начинается с выполнения тела цикла. После чего вычисляется значение выражения. Если это значение отлично от нуля, тело цикла выполняется опять и снова вычисляется значение выражения. Выполнение оператора do-while заканчивается, когда значение этого выражения станет равным нулю
Пример

```
#include <stdio.h>

int main(void)
{
    int digits = 0, n = 1024;
    do
    {
        digits++;
        n /= 10;
    }
    while (n != 0);
    printf("The number has %d digit(s).\n", digits);
    return 0;
}
```

Операторы while и do-while, выраженные друг через друга

Циклы взаимозаменяемы

```
#include <stdio.h>

int main() {
    int i = 1;

    while (i <= 10) {
        printf("The square of %d is %d\n", i, i * i);
        i++;
    }

    return 0;
}
```

```
#include <stdio.h>

int main() {
    int i = 1;

    if (i <= 10) { // Проверка условия перед входом в цикл
        do {
            printf("The square of %d is %d\n", i, i * i);
            i++;
        } while (i <= 10);
    }

    return 0;
}
```

46. Оператор break.

47. Оператор continue.

48. Оператор goto.

Оператором goto в языке С называется оператор, передающий управление в любую точку программы, помеченную меткой

Метка – это идентификатор, расположенный вначале оператора

```
идентификатор : оператор
```

Сам оператор goto записывается в форме

```
goto идентификатор;
```

while через goto

```
#include <stdio.h>
int main() {
    int i = 1;

    start: // Метка для начала цикла
    if (i > 5) {
        goto end; // Переходим к метке конца, если условие цикла не выполняется
```

```
    }
    printf("%d\n", i);
    i++;
    goto start; // Возвращаемся к началу цикла

    end: // Метка для конца цикла
    return 0;
}
```

do-while через goto

```
#include <stdio.h>

int main() {
    int i = 1;

    loop_start: // Метка для начала цикла
    printf("%d\n", i);
    i++;

    if (i <= 5) {
        goto loop_start; // Возвращаемся к началу цикла, если
условие истинно
    }

    return 0;
}
```

12. Операторы цикла for, while. Операторы break, continue, goto, пустой оператор

43. Оператор цикла for.

Оператор for реализует цикл со счетчиком

```
for (выражение_1; выражение_2; выражение_3) оператор
```

выражение_1 - выражение, которое выполнится единожды первым, в C99 может быть заменено определением
выражение_2 - выражение, истинность которого определяет конец цикла
выражение_3 - выражение, выполняемое после каждого выполнения оператора

Любое из трех выражений можно опустить, но точки с запятой должны остаться на своих местах. Если опустить все выражения, то условие продолжение цикла всегда истинно

```
for ( ; ; )
    printf("Infinity loop\n");
```

44. Операция запятая.

Если нужен оператор for с двумя или более выражениями инициализации или нужно изменить несколько переменных в конце цикла. Это можно сделать с помощью *операции запятая*

```
выражение_1, выражение_2
```

Сначала вычисляется выражение_1 и его значение отбрасывается, затем вычисляется выражение_2 - результат операции всей операции.
выражение_1 всегда должно содержать побочный эффект

```
for (sum = 0, i = 1, n = 5; i <= n; sum += i, i++)
    ;
```

42. Оператор цикла while.

Операторы while и for, выраженные друг через друга

Циклы взаимозаменяемы

```
#include <stdio.h>

int main() {
```

```
    int i = 1;

    while (i <= 5) {
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        printf("%d\n", i);
    }

    return 0;
}
```

46. Оператор break.

47. Оператор continue.

48. Оператор goto.

Оператором goto в языке С называется оператор, передающий управление в любую точку программы, помеченную меткой
Метка – это идентификатор, расположенный вначале оператора

```
идентификатор : оператор
```

Сам оператор goto записывается в форме

```
goto идентификатор;
```

while и for через goto

```
#include <stdio.h>
int main() {
    int i = 1;

```

```
    start: // Метка для начала цикла
    if (i > 5) {
        goto end; // Переходим к метке конца, если условие цикла
не выполняется
    }
    printf("%d\n", i);
    i++;
    goto start; // Возвращаемся к началу цикла

    end: // Метка для конца цикла
    return 0;
}
```

13. Операторы цикла for, do-while. Операторы break, continue, goto, пустой оператор

43. Оператор цикла for.

44. Операция запятая.

45. Оператор цикла do-while.

Операторы do-while и for, выраженные друг через друга

Циклы взаимозаменяемы

```
#include <stdio.h>

int main() {
    int i = 1;

    do {
        printf("%d\n", i);
        i++;
    } while (i <= 5);

    return 0;
}
```

```
#include <stdio.h>

int main() {
    for (int i = 1; ; i++) { // Начинаем бесконечный цикл
        printf("%d\n", i);

        if (i > 5) {
            break;
        }
    }

    return 0;
}
```

46. Оператор break.

47. Оператор continue.

48. Оператор goto.

Оператором goto в языке С называется оператор, передающий управление в любую точку программы, помеченную меткой
Метка – это идентификатор, расположенный вначале оператора

```
идентификатор : оператор
```

Сам оператор goto записывается в форме

```
goto идентификатор;
```

for через goto

```
#include <stdio.h>
int main() {
    int i = 1;

    start: // Метка для начала цикла
    if (i > 5) {
        goto end; // Переходим к метке конца, если условие цикла не выполняется
    }
    printf("%d\n", i);
    i++;
    goto start; // Возвращаемся к началу цикла

    end: // Метка для конца цикла
    return 0;
}
```

do-while через goto

```
#include <stdio.h>

int main() {
    int i = 1;

    loop_start: // Метка для начала цикла
```

```
    printf("%d\n", i);
    i++;

    if (i <= 5) {
        goto loop_start; // Возвращаемся к началу цикла, если
условие истинно
    }

    return 0;
}
```

14 Типы данных. Простые типы

49. Что такое тип данных?

Типом данных называется множество значений и операций над ними

50. Что такое статическая типизация?

Язык Си поддерживает *статическую типизацию*. Это значит, что типы всех используемых в программе переменных должны быть известны перед ее компиляцией.
Переменная, параметр подпрограммы, возвращаемое значение функции связывается с типом в момент объявления и тип не может быть изменен позже

51. Что такое динамическая типизация?

Динамическая типизация — приём, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов
Информация о типе переменной отсутствует на момент компиляции программы

52. Сравните статическую и динамическую типизации.

Динамическая типизация	**Статическая типизация**
Типы неизвестны на момент компиляции	Типы известны на момент компиляции
Типы определяются автоматически	Типы определяются заранее
Повышает безопасность и читаемость кода	Может возникнуть непредсказуемое поведение при выполнении
Меньшая гибкость при написании кода	Большая гибкость при работе с переменными
Ошибки обнаруживаются при компиляции	Ошибки обнаруживаются при выполнении

Строгая и нестрогая типизации

Строгая / нестрогая типизация. Строгая типизация выделяется тем, что язык не позволяет смешивать в выражениях различные типы и не выполняет автоматические неявные преобразования, например нельзя вычесть из строки множество. Языки с нестрогой типизацией выполняют множество неявных преобразований автоматически, даже если может произойти потеря точности или преобразование неоднозначно.

53. Классификация типов данных в языке Си.

Типы в языке Си делятся на простые и составные

Простые могут содержать только одно значение

- символьные (char)
- целые (int)
- вещественные (float, double и др.)
- перечисляемый тип
- void
- указатели

Составные могут содержать несколько однотипных или разнотипных значений

- массивы
- структуры
- объединения

54. Целый тип в языке Си.

Целочисленным типом данных называется простой тип данных, множество значений которого является подмножеством целых чисел Z

Язык Си позволяет описать несколько типов целых чисел. Они будут различаться между собой занимаемым объемом памяти, возможностью присваивания положительных и отрицательных значений. Это можно сделать с помощью модификаторов типа:

- signed
- unsigned
- short
- long

Порядок модификаторов не важен Язык Си позволяет сокращать имена целых типов, опуская слово int

Стандарт C99 добавляет еще два целых типа

- long long int
- unsigned long long int

Но в стандарте существуют два правила, которых должны придерживаться все компиляторы:

- пункт 5.2.4.2 определяет минимальный диапазон значений (limits.h)
- пункт 6.3.1.1 определяет относительные размеры типов
  sizeof(short int) <= sizeof(int) <= sizeof(long int) <= sizeof(long long int)

55. Целочисленные константы в языке Си.

Целочисленные константы могут записываться в десятичной, восьмеричной и шестнадцатеричной системах счисления: 8-ые константы начинаются с 0, 16-ые константы начинаются с 0x

Тип целочисленных констант обычно *int*. Если значение константы слишком велико, чтобы поместится в int, компилятор (который поддерживает стандарт C99) попытается определить наименьший подходящий тип, в который оно уместится
Чтобы компилятор интерпретировал константу как long int надо добавить суффикс L или l к значению (в случае типа long long int – LL или ll): 15L, 0377L, 0x7fffL
Чтобы показать, что константа беззнаковая, необходимо добавить суффикс U и u к значению: 15U, 0377U, 0x7fffU
Суффиксы L (или LL) и U могут использоваться вместе

56. Целочисленные типы заданного размера.

Для «обычных» типов стандарт определяет только диапазон допустимых значений, это позволяет этим типам быть максимально переносимыми.

Заголовочный файл stdint.h объявляет *целочисленные типы, которые имеют заданный размер*, имеют заданный минимальный размер и которые являются наиболее быстрыми при использовании. Они были введены в C99
На такие типы стандарт налагает гораздо больше требований, из-за этого они менее переносимы, так как машина может, к примеру, не поддерживать тип такого размера

57. Ввод/вывод целочисленных переменных.

Ввод/вывод типа int - %d
Ввод/вывод типа unsigned - %u (10 СС), %o (8 СС), %x (16 СС)
Ввод/вывод типа short - %h + %d, %u, %o, %x
Ввод/вывод типа long (long long) - %l (%ll) + %d, %u, %o, %x

```
unsigned int u;
scanf("%u", &u);
printf("%u", u);
scanf("%o", &u);
printf("%o", u);
scanf("%x", &u);
printf("%x", u);

short s;
scanf("%hd", &s);
printf("%hd", s);

long l;
scanf("%ld", &l);
printf("%ld", l);
```

58. Вещественный тип в языке Си. Особенности сравнения вещественных переменных.

Вещественным типом данных называется простой тип данных, множество значений которого является подмножеством вещественных чисел R, а также положительная бесконечность (+inf), отрицательная бесконечность (-inf) и NaN

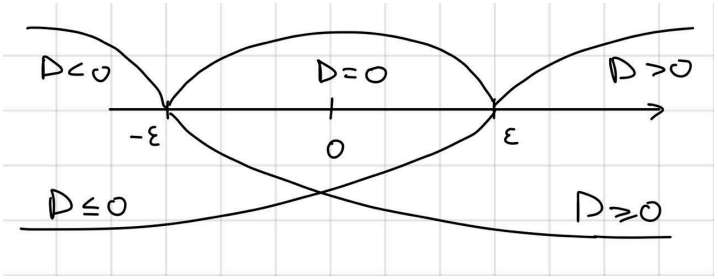
Язык Си предоставляет три вещественных типа:

- float: 6 - 8 значащих разрядов
- double: 13 - 16 значащих разрядов
- long double

Константы, определяющие характеристики вещественных типов, могут быть найдены в заголовочном файле float.h

Сравнивать вещественные числа нужно с заданной точностью ϵ, так как у вещественных чисел есть ограничения по хранению значащих чисел и

точности. Как следствие там может храниться мусор, который мешает сравнивать числа точно



59. Вещественные константы в языке Си.

Вещественная константа может быть записана разными способами: 57.0, 57., 5.7e1, 57E0
Вещественная константа должна содержать десятичную точку и/или экспоненту.
По умолчанию вещественные константы имеют тип double. Если нужен тип float - суффикс f или F (для типа long double – L или l)

60. Ввод/вывод вещественных переменных.

Ввод/вывод типа float - %e, %f, %g
Ввод/вывод типа double - %l + %e, %f, %g
Ввод/вывод типа long double - %L + %e, %f, %g

```
double d;
scanf("%lf", &d);

long double ld;
scanf("%Lf", &ld);
printf("%Lf", ld);
```

Используйте l только в функции scanf, но не в функции printf. В функции printf спецификаторы e, f и g могут быть использованы для вывода как значений типа float, так и значений типа double. Стандарт C99 позволяет использовать спецификаторы le, lf, lg в функции printf, но они не имеют значения

61. Символьный тип в языке Си.

Для работы с символами предназначен тип char

Переменной типа char может быть присвоено значение любого ASCII символа. Под значения типа char отводится 1 байт
Язык Си интерпретирует этот тип как «маленькое целое». В переменную типа char *помещается не сам символ, а его код из таблицы ASCII*
Стандарт не определяет «знаковость» этого типа.

62. Символьные константы в языке Си.

Символьная константа – это некоторый символ таблицы ASCII, заключенный в одиночные кавычки. Они не являются lvalue

63. Ввод/вывод символьных переменных. Функции для обработки отдельных символов.

Ввод/вывод типа char - %c
Также можно вывести ASCII код символа через спецификатор %d

```
char ch;
scanf("%c", &ch);
printf("%c %d", ch, ch);
```

Стандартные *функции для обработки отдельных символов* объявляются в заголовочном файлу ctype.h

int isalnum(int c); // c - буква либо цифра
int isalpha(int c); / c - буква
int isblank(int c); // c - пробел или горизонтальная табуляция
int iscntrl(int c); // c - управляющий символ, такой как <Ctrl+B>.
int isdigit(int c); // c - десятичная цифра
int isgraph(int c); // c - печатаемый символ, отличный от пробела
int islower(int c); // c - символ нижнего регистра
int isprint(int c); // c - печатаемый символ
int ispunct(int c); // c - знак препинания
int isspace(int c); // c — пробельный символ
int isupper(int c); // c - символ верхнего регистра
int isxdigit(int c); // c — шестнадцатеричная цифра
int toupper(int c); // переводит буквы нижнего регистра в верхний регистр
int tolower(int c); // переводит буквы верхнего регистра в нижний регистр

64. Перечисляемый тип в языке Си.

Перечисляемым типом данных или перечислением называется простой тип данных, каждому значению из конечного множества значений типа сопоставлен идентификатор

Перечисляемый тип данных в C является подтипом целочисленного типа данных, способного представить все значения типа int.

```
enum идентификатор { идентификатор, ..., идентификатор };
```

По умолчанию компилятор назначает значения 0, 1, 2, ... «константам» в каждом конкретном перечислении. Значение можно переназначить

```
enum season_e {
    WINTER = 5,
    SPRING = 4,
    SUMMER = 3,
    AUTUMN = 2,
};
```

```
enum season_e {
    WINTER, // 0
    SPRING, // 1
    SUMMER, // 2
    AUTUMN, // 3
};
```

Значения перечисляемого типа можно перемешивать с обычными целыми

```
int i;
enum {WINTER, SPRING, SUMMER, AUTUMN} season;

i = WINTER; // i === 0
season = 0; // season === WINTER
season++; // season === SPRING
i = season + 2; // i === 3
season = 4; // i === ?
```

65. Логический тип в языке Си (до и после стандарта C99)

66. Определение пользовательского типа (typedef). typedef vs define

Конструкция typedef позволяет задавать собственное имя для типа данных

```
typedef тип имя;
```

Здесь "тип" - любой разрешенный тип данных и "имя" - любой разрешенный идентификатор

Использование конструкции typedef позволяет

- улучшить читаемость программы
  Предположим, что переменные используются для хранения количества единиц товара. Использование специального имени типа более наглядно
- улучшить сопровождение программы
  Если позже мы решим использовать другой тип для представления количества нам нужно будет исправить только конструкцию typedef

typedef vs define

- typedef дает имена типам, как например называть int "Score";
  #define дает имена значениям, например, говоря, что 3.14 - это "PI"
- typedef обрабатывается компилятором, в то время как #define управляется препроцессором перед компиляцией
- #define просто меняет текст в вашем коде, в то время как typedef создает новые типы

Сравним определение типа через typedef и define

```
#include <stdio.h>
typedef char* ptr;
#define PTR char*
int main()
{
    ptr a, b;
    PTR x, y;
    printf("sizeof a:%zu\n", sizeof(a)); // 8
    printf("sizeof b:%zu\n", sizeof(b)); // 8
    printf("sizeof x:%zu\n", sizeof(x)); // 8
    printf("sizeof y:%zu\n", sizeof(y)); // 1
```

```
    return 0;
}
```

```
PTR x, y, z;
```

Выражение превращается в

```
char *x, y, z;
```

Поэтому для определения пользовательских типов нежелательно использовать #define, так как это может привести к неожиданным ошибкам

67. Операция sizeof. Когда выполняется эта операция?

Унарная *операция sizeof* позволяет программе определить какое количество памяти требуется для хранения величины определенного типа

```
sizeof(type_name)
```

Значением выражения является size_t, представляющее собой количество байт необходимых для хранения величины типа type_name

Когда операция sizeof применяется к выражению скобки могут быть опущены sizeof может применяться к *любому* типу данных

Эта операция *выполняется на стадии компиляции*

15. Явное и неявное преобразование типов

Зачем нужна операция преобразования в Си

Машинные команды, как правило, обрабатывают операнды одного размера и одного типа. Например, может существовать команда, которая складывает два 16-ти битных целых числа, но не 16-ти и 32-х битные числа. Т.е. компилятор вынужден преобразовывать типы операндов так, чтобы подобрать подходящую машинную команду.

68. Что такое неявное преобразование типа?

Неявным преобразованием типа в языке С называется преобразование типа, которые выполняются автоматически без участия программиста

69. Когда происходит неявное преобразование типа? Примеры.

Неявное преобразование выполняется в следующих ситуациях:

- когда операнды в арифметическом или логическом выражении имеют разные типы (*обычное арифметическое преобразование*)
- когда тип выражения с правой стороны операции присваивания не совпадает с типом выражения с левой стороны
- когда тип аргументов в вызове функции не соответствует типу параметра в ее заголовке
- когда тип выражения в операторе return не соответствует типу функции

1. Обычное арифметическое преобразование

Стратегия, которая лежит в основе арифметического преобразования – преобразовать операнды к самому «широкому» типу, который может содержать значения обоих операндов. Такое преобразование иногда называют *расширением типа*

- `float -> double -> long double`
- целое `->` вещественное
- `char/short -> int -> unsigned int -> long int -> unsigned long int`
- знаковое `->` беззнаковое (прибавление `UMAX + 1`)

2. Преобразования во время присваивания

Тип выражения с правой стороны преобразуется к типу переменной с левой стороны. Если тип переменной такой же «широкий», как и тип выражения, это не вызовет никаких затруднений
В противном случае произойдет «сужение» типа и результат может не соответствовать ожидаемому. Обычно компилятор выдает предупреждение

3. Преобразование типов аргументов при вызове функции
Правила, по которым выполняется такое преобразование, зависят от того, известен ли компилятору прототип функции

- Прототип известен. Значение каждого аргумента неявно преобразуется к типу соответствующего параметра как при присваивании
- Прототип неизвестен. Компилятор выполняет расширение по умолчанию: 1) float в double, 2) char/short в int

4. Преобразование типов при выходе из функции
Если тип выражения в операторе return не соответствует типу функции, выражения будет неявно преобразовано к типу функции (как при присваивании)

Проблемы неявного преобразования:

- потеря данных при сужающем преобразовании
- непредсказуемое поведение, так как компилятор автоматически выбирает способ преобразования

70. Явное преобразование типа. Операция приведения типа. Примеры.

Явным преобразованием типа в языке С называется преобразование типа при помощи операции преобразования типа

```
(имя типа) выражение
```

«имя типа» - тип, к которому будет преобразовано выражение

Явное преобразование используется:

- для «документирования» преобразования типа, которое и так имеет место: `i = (int) f;`
- выполнение необходимых преобразований при вычислениях

```
mean = sum / count; // целочисленное деление
mean = (double) sum / count; // обычное деление
```

Примеры использования

```
// Сравнение знаковых и беззнаковых
#include <stdio.h>
int main(void)
{
    int si;
    unsigned int ui;

    printf("Input si: ");
    scanf("%d", &si);
    printf("Input ui: ");
    scanf("%u", &ui);

    printf("si %d, ui %u, result of (si < ui) is %d\n", si, ui,
(si < 0 || (unsigned) si < ui));
    return 0;
}
```

```
//
#include <stdint.h>
int main(void)
{
    int64_t res;
    int32_t a = ...;
    // 1 (NOT OK) может случиться переполнение
    res = a * a;

    // 2 (OK)
    res = (int64_t) a * a;
    ...
```

Проблемы явного преобразования

- распространенность ошибок при неправильном использовании
- сложность чтения и поддержки кода

16. Подпрограмма. Определение, основные понятия. Продемонстрируйте эти понятия на примере подпрограммы на языке Си

71. Что такое подпрограмма?

Подпрограмма – именованная часть программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно вызвана из разных частей программы

Подпрограмма состоит из:

- заголовка
- тела

Формальная запись подпрограммы в языке С:

```
тип идентификатор(перечисление_формальных_параметров)
{
    оператор1
    ...
    операторN
}
```

Здесь:
`тип идентификатор(перечисление формальных параметров)` — сигнатура подпрограммы
`{ оператор1, ...,операторN }` — тело подпрограммы.

72. Преимущества использования подпрограмм.

Преимущества использования подпрограмм:

- Уменьшение сложности программирования за счет декомпозиции большой задачи на несколько подзадач меньшего размера
- Уменьшение дублированного кода
- Возможность повторного использования кода в других программах
- Сокрытие деталей реализации от пользователей подпрограммы

73. Виды подпрограмм.

Виды подпрограмм:

- функция
- процедура

Функция - это подпрограмма специального вида, которая всегда должна возвращать результат. Вызов функции является, с точки зрения языка программирования, выражением
Процедура - это независимая именованная часть программы, которую после однократного описания можно многократно вызвать по имени из последующих частей программы для выполнения определенных действий. Не возвращает результат

74. Что такое заголовок подпрограммы?

Сигнатурой подпрограммы или *заголовок подпрограммы* в языке С называется совокупность:

- имени (идентификатора) подпрограммы
- перечисление формальных параметров подпрограммы
- тип возвращаемого значения подпрограммы

```
тип  идентификатор(перечисление_формальных_параметров)
```

75. Что такое тело подпрограммы?

Тело подпрограммы – набор операторов, который будет выполнен всякий раз, когда подпрограмма будет вызвана

```
{
    оператор1
    ...
    операторN
}
```

76. Вызов подпрограммы. Формальные и фактические параметры.

Вызовом подпрограммы в языке С называется выражение, содержащее название подпрограммы и перечисление фактических параметров

```
подпрограмма(параметр_1, ...параметр_N)
```

Формальным параметром подпрограммы называется переменная, указанная в сигнатуре подпрограммы

Фактическим параметром подпрограммы или *аргумент* подпрограммы называется значение формального параметра при вызове подпрограммы

```
int sum(int a, int b) // a, b - формальные параметры
{
    return a + b;
}
...
int c = 5;
int d = 6;
sum(c, d); // c, d - фактические параметры
```

77. Способы передачи параметров в подпрограмму. Реализация этих способов применительно к языку Си.

Существуют две концептуальные модели передачи данных при передаче параметров:

- либо фактическое значение физически перемещается (передача по значению)
- либо передается путь доступа к нему (передача по ссылке)

Передача параметров по значению - значение фактического параметра используется для инициализации соответствующего формального параметра, который в дальнейшем действует как локальная переменная

Передача параметров по ссылке - передает путь доступа к данным (обычно – просто адрес) в вызываемую подпрограмму. Это открывает доступ к ячейке памяти, хранящий фактический параметр. Таким образом, вызываемая подпрограмма может получить доступ к фактическому параметру

В языке Си реализована передача параметров по значению. С помощью указателей можно имитировать передачу по ссылке

17. Описание функций на языке Си

78. Описание функции на языке Си.

Функцией в языке С называется подпрограмма, тип возвращаемого значения которой не является void

Функция состоит из:

- заголовка
- тела

Формальная запись функции в языке С:

```
тип идентификатор(перечисление формальных параметров)
{
    оператор1
    ...
    операторN
}
```

Здесь:
`тип идентификатор(перечисление формальных параметров)` — сигнатура функции
`{ оператор1, ...,операторN }` — тело функции

79. Заголовок функции языке Си.

Сигнатурой функции или *заголовком функции* в языке С называется совокупность:

- имени (идентификатора) функции
- перечисление формальных параметров функции
- тип возвращаемого значения функции

Пример:

```
int sum(int a, int b)
```

Здесь:
`int` — тип возвращаемого значения
`sum` — имя (идентификатор) подпрограммы
`int a , int b` — список формальных параметров

80. Тело функции на языке Си.

Тело функции – набор операторов, который будет выполнен всякий раз, когда подпрограмма будет вызвана

Тело функции не может содержать в себе определения других функций

Пример:

```
{
    int a = 1, b = 2;
    return a + b;
}
```

Так как функция должна обязательно возвращать значение не void, то ее тело не может быть пустым

81. Область видимости локальных переменных и параметров функции.

Переменные, описанные в теле функции, «принадлежат» только этой функции и не могут быть ни получены, ни изменены другой функцией

Область видимости в пределах прототипа функции применяется к именам переменных, которые используются в прототипах функций. Она простирается от точки, в которой объявлена переменная, до конца объявления прототипа.

82. Оператор return: назначение, использование.

Оператор return прерывает выполнение функции и возвращает вычисленное значение и управление в ту часть программы, из которой эта функция была вызвана

```
int max(int a, int b)
{

   if (a < b)
        return b;

return a;
}
```

Оператор return может использоваться в процедурах, которые ничего не возвращают. В этом случае он записывается без указания выражения.

```
void print_pos(int a)
{
    if (a <= 0)
        return;

    printf("%d\n", a);
}
```

83. Вызов функции на языке Си.

Для *вызова функции* необходимо указать ее имя, за которым в круглых скобках через запятую перечислить аргументы

```
float avg = average(2.0, 5.0);
```

Указывать скобки при вызове функции необходимо, даже если у этой функции нет параметров

Если функция возвращает значение, ее можно использовать в выражениях.

```
if (average(a, b) < 0.0)
```

Значение, возвращаемое функцией, может быть проигнорировано.

```
// явно указано, что возвращаемое значение не используется
(void) printf("Hello, world!\n");
```

84. Что такое объявление? Сколько объявлений «объекта» может быть в программе?

Объявление – это инструкция компилятору, как использовать указанное имя (описывает свойства переменной или функции). Объявлений одного и того же имени может быть сколько угодно, главное чтобы они все были одинаковы

85. Что такое определение? Сколько определений «объекта» может быть в программе?

Определение осуществляет привязку имени к сущности (к памяти для данных или к коду для функций), т.е. специфицирует код или данные, которые стоят за этим именем. Определение может быть только одно

86. Взаимное расположение объявлений, определения и использования «объекта».

Сначала идет объявление, затем определение, затем использование

87. Объявление функции на языке Си.

Объявление функции предоставляет компилятору всю информацию, необходимую для вызова функции: количество и типы параметров, их последовательность, тип возвращаемого значения.

```
тип-результата имя-функции(список формальных параметров с их типами);
```

Объявление функции должно соответствовать ее определению
Объявление функции может не содержать имен параметров. Однако их обычно оставляют для большей наглядности

88. Определение функции на языке Си.

Определение функции записывается после ее объявления
При определении описывается тело функции, т.е. ее операторы
Определение функции должно быть перед ее использованием

89. Особенности описания функций без параметров в языке Си.

```
void foo(void);
```

Это объявление означает, что у функции *нет ни одного параметра*

```
void foo();
```

Это объявление означает, что у функции могут быть, а могут и не быть параметры. Если параметры есть, мы не знаем ни их количество, ни их тип

90. Способ передачи параметров в функцию на языке Си.

В Си аргументы передаются в функцию *«по значению».*

Преимущества:

- безопасность значений, фактические параметры не изменяются
- простота использования и чтения кода

Недостатки:

- трата ресурсов при копировании элементов
- нельзя передавать явно массивы

- нельзя изменять значения аргументов, нужно использовать указатели

18. Функции. Передача параметров

90. Способ передачи параметров в функцию на языке Си.

91. Выполнение вызова функции.

```
int power(int base, int n)
{
    int res = 1;
    while (n > 0) {
            res = res * base;
            n = n - 1;
    }
    // n = 0
    return res;
}

a = power(2, 5);
```

1. Выделяется область памяти доступная только вызываемой функции (power)
2. В этой области памяти создаются переменные-параметры (base, n) и локальные переменные (res)
3. Переменным-параметрам присваиваются начальные значения, в качестве которых выступают аргументы из точки вызова (base = 2, n = 5)
4. Инициализируются локальные переменные (res = 1). В случае отсутствия такой инициализации локальные переменные содержат «мусор»
5. Выполняется тело функции
6. Вычисленное значение возвращается из функции (в нашем случае попадает в переменную a)
7. Выделенная область памяти разрушается

92. Что такое чистая функция?

Чистая функция - это функция, которая 1) является детерминированной, 2) не обладает побочными эффектами

Функция является детерминированной, если для одного и того же набора входных значений она возвращает одинаковый результат

Функции с побочными эффектами – это функции, которые в процессе выполнения своих вычислений могут модифицировать значения глобальных переменных, осуществлять операции ввода/вывода. Другим видом побочных эффектов является модификация переданных в функцию параметров.

93. Способы возврата значения из функции на языке Си.

- Прямой возврат значения с помощью оператора return
  Функция может возвращать значение, используя оператор return, за которым следует выражение, значение которого должно быть возвращено. Это значение затем может быть присвоено переменной в вызывающей функции.
- Возврат нескольких значений через указатели
  Если требуется вернуть из функции более одного значения, можно передать в функцию указатели на переменные, которые будут изменяться внутри функции

94. Возвращение нескольких значений из функции на языке Си.

Чтобы вернуть несколько значений или вернуть массив из функции нужно использовать *указатели*

Переменная-указатель – это переменная, которая содержит адрес
Переменная-указатель описывается как и обычная переменная. Перед ее именем необходимо указать символ «*»:

```
int *p;
```

В языке Си есть две операции, которые предназначены для использования именно с указателями.
Чтобы узнать адрес переменной- операция получения адреса «&». Если x – это переменная, то &x – адрес переменной x в памяти
Чтобы получить доступ к объекту, на который указывает указатель - операция разыменования «*». Если p – переменная-указатель, то *p представляет объект, на который сейчас указывает p

Использование указателей для передачи параметров

```
void decompose(float f, int *int_part, float *frac_part)
```

```
{
    *int_part  = f;
    *frac_part = f - *int_part;
}

int i = 0;
float f = 0.0;
decompose(3.14159, &i, &f); // i = 3, f = 0.14159
```

С помощью указателя мы передаем адрес объекта и потом с помощью операции разыменования изменяем значение по этому адресу

19. Рекурсия

95. Какая функция называется рекурсивной?

Функция называется *рекурсивной*, если она вызывает саму себя

96. Преимущества и недостатки рекурсивных функций.

Преимущества

- Рекурсивная форма может быть структурно проще и нагляднее, в особенности, когда сам реализуемый алгоритм рекурсивен

Недостатки

- Рекурсивный вызов использует больше памяти, поскольку создает свой набор переменных
- Рекурсия выполняется медленней, поскольку на каждый вызов функции требуется определенное время

97. Что такое «хвостовая рекурсия»? Особенности этого вида рекурсии.

Хвостовая рекурсия - частный случай рекурсии, при котором любой рекурсивный вызов является последней операцией перед возвратом из функции
Подобный вид рекурсии может быть легко заменён на итерацию путём формальной и гарантированно корректной перестройки кода функции. Оптимизация хвостовой рекурсии путём преобразования её в итерацию реализована во многих оптимизирующих компиляторах.

20. Одномерный статический массив

98. Что такое массив? Основные свойства массива.

Будем называть тип данных *агрегированным*, если из сущности этого типа можно выделить сущность определённого типа данных, простого или агрегированного

Будем называть *массивом* агрегированный тип данных, обладающий тремя свойствами:

- Линейность — все элементы массива в памяти расположены упорядоченно единым неразрывным блоком
- Однородность — все элементы одного типа и одного объёма
- Произвольный доступ — скорость доступа к любому элементу массива не зависит от его положения в массиве

99. Особенности описания статических массивов на языке Си.

Будем называть *статическим массив*, размер которого известен на момент трансляции

- Тип элемента может быть любым
- Количество элементов указывается целочисленным константным выражением
- Количество элементов не может быть изменено в ходе выполнения программы

100. Инициализация статических массивов на языке Си (в том числе и в стандарте C99).

```
int a[5] = {1, 2, 3, 4, 5};
// a[0] == 1, a[1] == 2, a[2] == 3, a[3] == 4, a[4] == 5

int b[5] = {1, 2, 3};
// b[0] == 1, b[1] == 2, b[2] == 3, b[3] == 0, b[4] == 0

int c[5] = {0};
// все элементы массива c – нули

int d[3] = {1, 2, 3, 4};
// warning: excess elements in array initializer
```

```
int f[] = {1, 2, 3, 4, 5};
// f[0] == 1, f[1] == 2, f[2] == 3, f[3] == 4, f[4] == 5
```

- Компилятор самостоятельно определит количество элементов в массиве f и выделит память
- В программе количество элементов в массиве f можно определить из выражения sizeof(f) / sizeof(f[0])

Выделенные инициализаторы (c99 designated initializers)

```
int e[10] = {0, 0, 5, 0, 0, 10, 0, 0, 15, 0};
// C99
int e[10] = {[2] = 5, [5] = 10, [8] = 15};
```

101. Операция индексации.

- Для доступа к элементу массива используется индекс
- Индексация выполняется с нуля
- В качестве индекса может выступать целочисленное выражение (например, a[i%2])
- Си не предусматривает никаких проверок на выход за пределы массива

```
for (size_t i = 0; i < N; i++)
    sum += a[i];
```

Адрес элемента

- &a[i]
- pa + i

Значение элемента

- a[i]
- *(pa + i)

114. Выражение из имени массива. Исключения из этого правила.

Результат выражения, состоящего из имени массива, представляет собой адрес области памяти, выделенной под этот массив

```
int a[10], *pa;
pa = a; // pa = &a[0]; но НЕ pa = &a;
pa[0] = = = a[0]
```

«Преобразование» массива в указатель

Исключение 1: операция sizeof для массива

```
int a[10];
int *pa = a;
printf("sizeof(a) = %d\n", sizeof(a)); // sizeof(a) = 40
printf("sizeof(pa) = %d\n", sizeof(pa)); // sizeof(pa) = 4
```

Исключение 2: массив – операнд операции получения адреса

```
int a[10];
int *pa;
printf("a = %p, &a = %p\n", (void*) a, (void*) &a);
// a = 0022fefff0, &a = 0022fefff0
```

&a возвращает указатель, НО тип этого указателя - указатель на массив (не указатель на целое)

```
pa = &a;
// error: initialization of 'int *' from incompatible pointer
// type 'int (*)[10]' [-Werror=incompatible-pointer-types]
```

Исключение 3: строковый литерал-инициализатор массива char[]

```
char a[] = "abcdef";
printf("sizeof(a) = %d\n", sizeof(a)); // sizeof(a) = 7
```

115. Можно ли отождествлять массивы и указатели?

Отличие массивов и указателей

- При определении массива и указателя под соответствующие переменные выделяется разное количество памяти
- Выражению из имени массива нельзя присвоить другое значение

102. Особенности передачи массива в функцию на языке Си.

Передача массива в функцию

Любое похожее на массив объявление параметра функции рассматривается компилятором как указатель

Аргументом может быть любой одномерный массив соответствующий типу массива в заголовке функции

Функция не может узнать размер массива, поэтому передается параметр, который содержит количество элементов в массиве

```
void f_2(int a[], size_t n);
void f_3(int *a, size_t n);
```

Функция может изменять значение элементов массива, используемого при вызове

Функция может обработать лишь часть массива

```
int a[] = {1, 2, 3, 4, 5};
print(&a[2], 3);
```

21. Указатели. Объявление, инициализация, базовые операции

103. Организация оперативной памяти с точки зрения прикладного программиста.

Память - последовательность ячеек для хранения информации

104. Дайте определение минимальной единице адресации?

Минимальную разницу между двумя разными адресами мы будем называть *минимальной единицей адресации*

105. Что такое машинное слово?

Будем считать, что составные части компьютера общаются сообщениями фиксированной длины, *машинными словами*

106. Что такое little endian/big endian?

https://habr.com/ru/articles/233245/
Машины с порядком хранения от старшего к младшему, *big endian* хранят старший байт первым. Если посмотреть на набор байтов, то первый байт (младший адрес) считается старшим
Машины с порядком хранения от младшего к старшему, *little endian* хранят младший байт первым. Если посмотреть на набор байт, то первый байт будет наименьшим

107. Что такое указатель?

Указатель - это объект, содержащий адрес объекта или функции, либо выражение, обозначающее адрес объекта или функции

108. Разновидности указателей в языке Си.

Разновидности указателей

- типизированный указатель на данные (тип* имя)
- бестиповой указатель (void* имя)
- указатель на функцию

109. Использование указателей в языке Си.

- Передача параметров в функцию
  - Изменяемые параметры
  - «Объемные» параметры

- Обработка областей памяти
  - Динамическое выделение памяти
  - Ссылочные структуры данных
  - и т.д.

110. Базовые операции для работы с указателями.

При определении переменной-указателя перед ее именем должен размещаться символ ‘*’
Для определения адреса данных используется операция ‘&’

```
int *p = &a;
```

Для доступа к данным, на которые указывает указатель, используется операция ‘*’

```
printf("%d %d\n", a, *p)
```

Операции ‘&’ и ‘*’ взаимно обратные

```
*&d = = = d
&*p = = = p
```

На одни и те же данные могут указывать несколько указателей

```
int a = 1;
int *p = &a; // p теперь указывает на a
int *q = p; // q теперь указывает туда же куда и p, т.е. на a
```

111. Инициализация указателей.

```
// Адресом переменной того же типа
int a = 1;
double d = 5.0;
int *p = &a;
int *p_err = &d; // error: initialization from incompatible
pointer type

// Значением другого указателя того же типа
int *q = p;
double *q_err = p; // error: initialization from incompatible
pointer type
```

```
// Значением NULL
int *r = NULL;
int *r_ok = 0;
int *r_err = 12345; // error: initialization makes pointer from integer without a cast
```

112. Константа NULL.

NULL - макрос, объявленный в заголовочном файле stddef.h (и других заголовочных файлах). Его значение - константа нулевого указателя

Константа нулевого указателя - целочисленное константное выражение со значением 0
Константа нулевого указателя, приведённая к любому типу указателей, является *нулевым указателем*

Свойства:

- не равен указателю на любой объект или функцию
- любые два нулевых указателя равны между собой
- разыменовывание нулевого указателя является операцией с неопределённым поведением

113. Модификатор const и указатели.

Указатель на константу, нельзя менять значение, хранящееся в памяти

```
int a = 5;
int b = 7;
const int *p = &a; // <-
printf("%d\n", *p);
*p = 4; // error: assignment of read-only location '*p'
p = &b;
```

Константный указатель, нельзя указывать на другую ячейку памяти

```
int a = 5;
int b = 7;
int * const p = &a; // <-
printf("%d\n", *p);
*p = 4;
p = &b; // error: assignment of read-only variable 'p'
```

Константный указатель на константу, можно только выводить значение переменной

```
int a = 5;
int b = 7;
const int *const p = &a; // <-
printf("%d\n", *p);
*p = 4; // error: assignment of read-only location '*p'
p = &b; // error: assignment of read-only variable 'p'
```

22. Указатели. Адресная арифметика

103. Организация оперативной памяти с точки зрения прикладного программиста.

104. Дайте определение минимальной единице адресации?

105. Что такое машинное слово?

106. Что такое little endian/big endian?

107. Что такое указатель?

116. Сложение указателя с целым числом.

```
тип *p = ...;

p += n;
```
новый адрес в p = старый адрес из p + n * sizeof(тип)

```
p -= m;
```
новый адрес в p = старый адрес из p - m * sizeof(тип)

117. Сравнение указателей.

Указатели допускается использовать в операциях сравнения. При сравнении указателей сравниваются адреса.

При этом можно

- сравнивать указатель с NULL
- сравнивать два однотипных указателя

118. Вычитание указателей.

```
тип a[10];
тип *pa = a, *pb = &a[2];
pb – pa = (адрес из pb – адрес из pa) / sizeof(тип)
```

ptrdiff_t – знаковый целый тип, создан для хранения разности между двумя указателями. Разрядность типа ptrdiff_t зависит от платформы и реализации компилятора. Тип определен в заголовочном файле stddef.h

23. Многомерный статический массив. Объявление, инициализация, особенность реализации

119. «Концепция» многомерного массива в языке Си.

Многомерный статический массив размерности N в Си рассматривается как одномерный массив, элементами которого являются массивы размерности N-1

120. Описание многомерного массива на языке Си. Особенности расположения в памяти.

Количество размерностей массива практически неограничено
Многомерный массив *описывается*, как
`тип a[N][M];`

Компилятор Си располагает строки матрицы a в *памяти* одну за другой вплотную друг к другу
`a[0][0] | a[0][1] | a[1][0] | a[1][1] | a[2][0] | a[2][1]`
Следует из определения многомерного массива и свойства линейности обычного массива

121. «Компоненты» многомерного массива в языке Си.

```
int a[2][3][5];

a – массив из двух элементов типа “int [3][5]”
int (*p)[3][5] = a;

a[i] - массив из трех элементов типа “int [5]” (i [0, 1])
int (*q)[5] = a[i];

a[i][j] - массив из пяти элементов типа “int” (i [0, 1],
j [0, 1, 2])
int *r = a[i][j];

a[i][j][k] - элемент типа “int” (i [0, 1], j [0, 1, 2],
k [0, 1, 2, 3, 4])
int s = a[i][j][k];
```

122. Инициализация многомерных массивов на языке Си.

```
int a[3][3] =
{
    {1, 2, 3},
    {4, 5}
};

int d[][2] = { {1, 2} };

int e[][] =
{
    {1, 2},
    {4, 5}
};
// error: array type has incomplete element type
Второе значение размерности обязательно

int b[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
// warning: missing braces around initializer [-Wmissing-braces]

int c[2][2] = {[0][0] = 1, [1][1] = 1};
// c99
```

123. Доступ к элементу многомерного массива в языке Си.

```
int a[3][2] = {{1, 2}, {3, 4}, {5, 6}};
int i = 1, j = 1;
a[i][j] === *(*(a + i) + j)
```

	0x100	0x104	0x108	0x10c	0x110	0x114	
…	1	2	3	4	5	6	…
	a[0][0]	a[0][1]	a[1][0]	a[1][1]	a[2][0]	a[2][1]	
	a[0]		a[1]		a[2]		
	a						

```
// адресная арифметика
*(*(a + i) + j)
(*(a + i))[j]
*(a[i] + j)
*(&a[0][0] + 4 * i + j)
```

24. Многомерный статический массив. Особенности использования

Операция sizeof и многомерный массив

```
int a[][3] = {{1, 2, 3}, {4, 5, 6}};
printf("%d\n", sizeof(a) / sizeof(a[0][0])); // кол-во элементов
printf("%d\n", sizeof(a) / sizeof(a[0])); // кол-во строк
printf("%d\n", sizeof(a[0]) / sizeof(a[0][0])); //кол-во столбцов
```

124. Обработка многомерных массивов с помощью указателей.

Обнуление всех элементов матрицы

```
int a[N][M];
int *p;

for (p = &a[0][0]; p < &a[0][0] + N*M; p++)
    *p = 0;
```

Обнуление i-ой строки матрицы

```
int a[N][M];
int *p;

for (p = a[i]; p < a[i] + M; p++)
    *p = 0;
```

Обнуление j-ой строки

```
int a[N][M];
int (*q)[M];

for (q = a; q < a + N; q++)
    (*q)[j] = (пустое множество);
```

125. Передача многомерного массива в функцию.

Пусть определена матрица a. Для ее обработки могут быть использованы функции со следующими прототипами:

```
int a[N][M];
```

```
void f(int a[N][M], int n, int m);
void f(int a[][M], int n, int m);
void f(int (*a)[M], int n, int m);

// Неверные прототипы
void f(int a[][], int n, int m);
// не указана вторая размерность

void f(int *a[M], int n, int m);
void f(int **a, int n, int m);
// неверный тип – массив указателей
```

126. Особенности использования const и многомерных массивов в языке Си.

Согласно C99 6.7.3 #8 и 6.3.2.3.2 выражение `T (*p)[N]` не преобразуется неявно в `T const (*p)[N]`

Способы борьбы

- не использовать const;
- использовать явное преобразование типа `print((const int (*)[M]) a, 2, 3);`

Такое неявное преобразование запретили, так как можно было обходными путями изменять константы

25. Строка. Объявление, инициализация, ввод, вывод

127. Что такое строка в языке Си? Преимущества и недостатки такого представления.

Строка – это последовательность символов, заканчивающаяся и включающая первый нулевой символ (англ., null character ‘\0’)

Преимущества подхода

- Простота

Недостатки подхода

- Отсутствие быстрого способа определения длины строки
- Тщательность при работе с нулевым символом

128. Описание строки на языке Си.

Определение переменной-строки, которая может содержать до 80 символов обычно выглядит следующим образом:

```
#define STR_LEN 80
...
char str[STR_LEN+1]; // !
```

Длина строки задается на 1 символ больше, так как 1 символ отводится на символ конца строки ‘\0’

129. Особенности передачи строк в функцию в языке Си.

Строки передаются также как массивы
Длину строки передавать не нужно, так как идентификатором окончания строки является символ ‘\0’
Длину строки передаются в случаях, если нужно безопасно обработать строку

130. Особенности инициализации строковых переменных.

Инициализация строковых переменных

```
char str_1[] = {‘J’,‘u’,‘n’, ‘e’, ‘\0’};
char str_2[] = "June"; // ‘\0’ добавляется сам в конце строки
char str_3[5] = "June";
char str_4[3] = "June";
// error: initializer-string for array of chars is too long
char str_5[4] = "June";
```

```
// str_5 не строка!
```

131. Что такое строковый литерал?

Строковый литерал – последовательность символов, заключенных в двойные кавычки

```
char str[] = "String for test"
```

Строковый литерал рассматривается компилятором как массив элементов типа char. Для строкового литерал из n символов, он выделяет n+1 байт памяти, которые заполняет символами строкового литерала и завершает ‘\0’

Массив, который содержит строковый литерал, существует в течение всего времени выполнения программы

В стандарте сказано, что поведение программы не определено при попытке изменить строковый литерал. Обычно строковые литералы хранятся в read only секции

132. Указатель на строковый литерал и на строку.

```
char str_1[] = “June”; // массива символов
char *str_2 = “June”; // указатель на строковой лиетрал

str_1[0] = ‘j’;
str_2[0] = ‘j’;
```

Т.к. str_2 - указатель на строковой литерал, а литерал является константой, то с помощью указателя нельзя изменить строковой литерал, она доступна только для чтения

133. Способы описания «массива строк» на языке Си.

Способ 1: двумерный массив строк

```
char arr_1[][9] = {"January", "February", "March"};
```

J	**a**	**n**	**u**	**a**	**r**	**y**	**\0**	**\0**
F	**e**	**b**	**r**	**u**	**a**	**r**	**y**	**\0**
M	**a**	**r**	**c**	**h**	**\0**	**\0**	**\0**	**\0**

Способ 2: массив указателей на строки (“ragged array”)

```
/*const*/ char *arr_2[] = {"January", "February", "March"};
```

[0]	==>	**J**	**a**	**n**	**u**	**a**	**r**	**y**	**\0**	
[1]	==>	**F**	**e**	**b**	**r**	**u**	**a**	**r**	**y**	**\0**
[2]	==>	**M**	**a**	**r**	**c**	**h**	**\0**			

134. Ввод/вывод строк

Вывод

```
#include <stdio.h>
...
char str[] = "Hello, world!";

printf("%s\n", str);
puts(str);
```

Ввод

```
#include <stdio.h>
...
char str[10];

scanf("%s", str); // Через scanf нельзя ввести строку с пробелами!, небезопасна
gets(str); // небезопасна
fgets(str, sizeof(str), stdin); // безопасна; прекращает ввод, если: прочитан ‘\n’, EOF, прочитано size-1 символ; введенная строка всегда заканчивается нулем
```

Также можно сделать собственную реализацию с помощью `int getchar(void)`

26 Строка. Функции стандартной библиотеки для обработки строк

127. Что такое строка в языке Си? Преимущества и недостатки такого представления.

128. Описание строки на языке Си.

135. Обработка строк (strcpy, strcat, strlen, strcmp, snprintf, strtok, перевод строки в число).

Функции для обработки строк описаны в string.h

```
char* strcpy(char *dst, const char *src); // копирование в dst их src
char* strncpy(char *dst, const char *src, size_t n); // безопасное копирование в dst из src
size_t strlen(const char *s); // длина строки
char* strcat(char *s1, const char *s2); // объединение 2-х строк
char* strncat(char *s1, const char *s2, size_t n); // безопасное объединение 2-х строк
int strcmp(const char *s1, const char *s2); // сравнение
int strncmp(const char *s1, const char *s2, size_t n); // безопасное сравнение
// перевод строки в число stdlib.h
long int atol(const char* str);
long int strtol(const char* string, char** endptr, int basis);

// форматированный вывод
int sprintf(char *s, const char *format, ...);
int snprintf(char *s, size_t n, const char *format, ...);
```

136. Что такое лексикографический порядок слов?

Строка s1 меньше строки s2, если выполнено любое из двух условий:

- первые i символов строк s1 и s2 одинаковы, а символ s1[i+1] меньше символа s2[i+1] (пример, “abc” < “abd” или “abc” < “bcd”)
- все символы строк s1 и s2 одинаковы, но строка s1 короче строки s2 (пример, “abc” < “abcd”)

27 Строки. Способы хранения слов, алгоритм выделения слов, функция strtok

127. Что такое строка в языке Си? Преимущества и недостатки такого представления.

128. Описание строки на языке Си.

Способы хранения слов, алгоритм выделения слов

Слово хранится, как массив элементов типа char. Символы располагаются друг за другом. Байты соотносятся с кодом буквы в таблице ASCII. Заканчивается строка терминальным нулем с кодом 0

strtok

```
char* strtok(char *string, const char *delim);
```

Функция strtok выполняет поиск лексем (слов) в строке string. Последовательные вызовы этой функции разбивают строку string на лексемы. Лексема представляет собой последовательность символов, разделенных символами-разделителями, которые указываются в строке delim.

При первом вызове функция принимает строку string в качестве аргумента. В последующие вызовы функция ожидает NULL. Функция возвращает указатель на найденную лексему. Если лексем не найдено, то возвращается NULL. При каждом вызове strtok можно варьировать набор символов-разделителей delim

strtok изменяет передаваемые данные, вставляя терминальные нули вместо символов разделения

28. Структуры

137. Что такое структура в языке Си?

Структура представляет собой одну или несколько переменных (возможно разного типа), которые объединены под одним именем.

138. Описание структуры в языке Си.

Формальное описание

```
struct <имя> // тег
{
    <тип 1> <имя_1>;
    <тип 2> <имя_2>;
    ...
    <тип N> <имя_N>;
};
```

139. Что такое тег структуры? Для чего он используется?

Имя, которое располагается за ключевым словом struct, называется *тегом* структуры:

- Используется для краткого обозначения той части объявления, которая заключена в фигурные скобки
- Тег может быть опущен (безымянный тип) (C11)

140. Что такое поле структуры? Типы полей структуры. Описание полей структуры.

Перечисленные в структуре переменные называются *полями* структуры

141. Особенности именования тегов и полей структуры.

Тег структуры не распознается без ключевого слова struct. Благодаря этому тег *не конфликтует* с другими именами в программе

«Тело» структуры представляет собой *самостоятельную область видимости*: имена в этой области не конфликтуют с именами из других областей

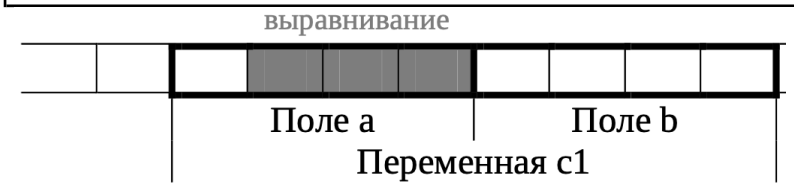
142. Расположение полей структуры в памяти. Выравнивание. Упаковка.

Поля структуры *располагаются в памяти* в порядке описания. Адрес первого поля совпадает с адресом переменной структурного типа

С целью оптимизации доступа компилятор может располагать поля в памяти не одно за другим, а по адресам кратным размеру поля - ***выравнивание***

```
struct s_1 {
    char a;
    int b; } c1;

sizeof(c1) == 8
(char*) &c1 == &c1.a
```



Можно использовать ***упаковку***, чтобы убрать выравнивание и поля располагались строго друг за другом

```
#pragma pack(push, 1)
struct s_2
{
    char a;
    int b; } c2;
#pragma pack(pop)

sizeof(c2) == 5
(char*)&c2 == &c2.a
```

143. Выравнивание данных.

Выравнивание нужно, чтобы обеспечить наиболее быстрый доступ к памяти
Существуют платформы, которые не позволяют обрабатывать невыровненные данные

Каждый тип (за исключением типа char) имеет требования к выравниванию: переменные располагаются по адресу, кратному их размеру

На разных платформах переменные могут по-разному располагаться в памяти компьютера

Требования к выравниванию переменной структурного типа

- На выравнивание переменной структурного типа влияют требования выравнивания и размеры полей этого структурного типа
- Обычно переменная структурного типа выравнивается по границе поля самого требовательного типа

Существует *завершающее выравнивание*
Если мы объявляем, например, массив структур, то чтобы каждый элемент массива располагался по кратному адресу, добавляется выравнивание в конец

144. Способы описания переменных структурного типа.

Раздельное определение типа и переменных

```
struct date {
    int day;
    int month;
    int year;
};

struct date birthday;
struct date exam;
```

Совместное определение типа и переменных

```
struct date {
    int day;
    int month;
    int year;
} birthday, exam;
```

145. Инициализация переменных структурного типа.

- Для инициализации переменной структурного типа необходимо указать список значений, заключенный в фигурные скобки
- Значения в списке должны появляться в том же порядке, что и имена полей структуры

- Если значений меньше, чем полей структуры, оставшиеся поля инициализируются нулями

```
struct date {
    int day;
    int month;
    int year;
};

struct person
{
    char name[NAME_LEN+1];
    struct date birth;
};

int main(void)
{
    struct date today = {17, 4, 2024};
    struct date day = {17};
    struct date year = {, , 2024}; // error: expected expression
before ',' token
    struct person rector = {"Gordin", {16, 8, 1969}};

    struct date holidays[] =
    {
        { 9, 5, 2024},
        {10, 5, 2024},
        {11, 5, 2024},
        {12, 5, 2024}
    };

    // Введено в C99
    struct date exam = {.day = 13, .month = 1, .year = 2019};
    // Легче читать и понимать
    // Значения могут идти в произвольном порядке
    // Отсутствующие поля получают нулевые значения
```

146. Операции над структурами.

Доступ к полю структуры осуществляется с помощью операции ".", а если доступ к самой структуре осуществляется *по указателю*, то с помощью операции "->"

```
struct date today, *some_date;

today.day   = 17;

(*some_date).day = 26;
some_date->month = 9;
```

Структурные переменные одного типа можно *присваивать* друг другу (замечание: у разных безымянных типов тип разный).
`some_date = today;`

Структуры *нельзя сравнивать* с помощью "==" и "!=".

Структуры могут *передаваться в функцию* как параметры (лучше через указатель) и *возвращаться из функции* в качестве ее значения.

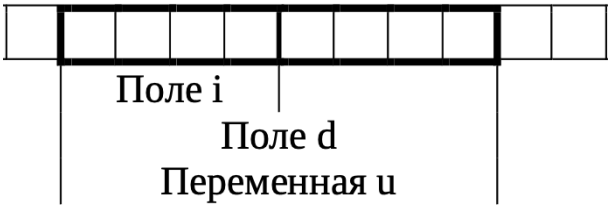
29. Объединения, перечисляемый тип

147. Что такое объединение?

Объединение, как и структура, содержит одно или несколько полей возможно разного типа под одним именем. Однако все поля объединения разделяют одну и ту же область памяти.

148. Расположение полей объединения в памяти.

```
union {
    int i;
    double d; } u;
```



Все поля объединения разделяют одну и ту же область памяти
Размер объединения равен размеру наибольшего поля объединения

149. Инициализация объединений.

Присвоение значения одному члену объединения обычно изменит значение других членов

```
printf("u.i %d, u.d %g\n", u.i, u.d); // u.i 2293664, u.d
1.7926e-307

u.i = 5;
printf("u.i %d, u.d %g\n", u.i, u.d); // u.i 5, u.d 1.79255e-307
```

```
union u_t {
    int i;
    double d;
};
...
union u_t  u_1 = {1};

// только c99
union u_t  u_2 = { .d = 5.25 };
```

150. Использование объединений.

Экономия места.

```
struct library_item {
    int number;
    int item_type;
    union {
        struct {
            char author[NAME_LEN + 1];
            char title[TITLE_LEN + 1];
            char publisher[PUBLISHER_LEN + 1];
            int year;
        } book;
        struct {
            char title[TITLE_LEN + 1];
            int year;
            int volume;
        } magazine;
    } item;
};
```

Создание структур данных из разных типов.

```
typedef enum { KIND_INT, KIND_DOUBLE } kind_num_t;

typedef struct
{
    kind_num_t kind;
    union {
        int i;
        double d; } u;
} number_t;

number_t arr[10];
```

Разный взгляд на одни и те же данные (машинно-зависимо)

```
union word {
    unsigned short word;
    struct word_parts
```

```
    {
        unsigned char lo;
        unsigned char hi;
    } parts;
} a;

a.word = 0xABCD;
printf("word 0x%4x, hi part 0x%2x, lo part 0x%2x", a.word,
a.parts.hi, a.parts.lo);
```

151. Сравните структуру и объединение.

Структура	**Объединение**
Описание схоже	Описание схоже
Каждое поле занимает отдельное место в памяти	Все поля разделяют одну и ту же область памяти
Каждому полю можно присваивать свое значение	Присвоение значения одному типу изменит все другие
Инициализировать можно разное кол-во поле	Инициализируют только одно поле

30. Текстовые файлы

156. Что такое файл?

Файл – именованная область данных на носителе информации.

157. Перечислите основные свойства файла.

Файл обладает набором различных *свойств*

- имя файла (файлы идентифицируются именами)
- тип файла
- размер файла
- права доступа
- время

158. Что такое файловая система?

Файловая система – специальный компонент операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю интерфейс для работы с данными, хранящимися на диске

Файловая система включает в себя:

- совокупность всех файлов на диске
- структуры данных, используемые для управления файлами
- интерфейс для взаимодействия с файлами

159. Назначение файловой системы?

Назначение файловой системы состоит в том, чтобы обеспечить пользователю интерфейс для работы с данными, хранящимися на диске

160. Классификация файлов.

Файлы бывают разных типов:

- обычные файлы
  - текстовые файлы
  - двоичные файлы
- специальные файлы
- файлы-каталоги.

161. Текстовые файлы.

Текстовые файлы содержат, главным образом, только печатные символы. Они организованы в виде последовательности строк, каждая из которых

заканчивается символом новой строки ‘\n’. В конце последней строки этот символ не является обязательным

163. Сравните текстовые и двоичные файлы.

Текстовые	**Двоичные**
переносимый	непереносим (информация хранится в представлении, в котором она хранятся в оперативной памяти)
последовательный доступ	произвольный доступ

164. Последовательность действий при работе с файлами (fopen, fclose, feof, ferror)

Стандартная библиотека ввода/вывода Си – это библиотека буферизированного обмена с файлами и устройствами, которые поддерживает операционная система. К таким устройствам относятся консоль, клавиатура, принтеры и многое другое

Для взаимодействия программы с файлом или устройством библиотека использует тип *FILE*, который описывается в заголовочном файле stdio.h. При открытии файла или устройства возвращается указатель на объект этого типа (*файловый указатель*)

```
FILE* fopen(const char *filename, const char *mode);
```

Функция *fopen* открывает файл. Возвращает файловый указатель, с помощью которого далее можно осуществлять доступ к файлу. Если вызов функции fopen прошел неудачно, то она возвратит NULL.

Режим (mode)	Описание
"r"	Чтение (Read). Файл должен существовать.
"w"	Запись (Write). Если файл с таким именем не существует, он будет создан, в противном случае его содержимое будет потеряно
"a"	Запись в конец файла (Append). Файл создаётся, если не существовал

Функция fopen может открывать файл в текстовом или бинарном режиме. По умолчанию используется *текстовый режим*. Если необходимо открыть файл в бинарном режиме, то в конец строки добавляется буква b, например "rb", "wb", "ab".

```
int fclose(FILE *f);
```
Функция *fclose* закрывает файл:

- выполняет запись буферизированных, но еще незаписанных данных
- уничтожает непрочитанные буферизированные входные данные
- освобождает все автоматически выделенные буфера
- после чего закрывает файл или устройство. Возвращает EOF в случае ошибки и нуль в противном случае

Почему нужно использовать fclose?

- У программы может быть ограничение на количество одновременно открытых файлов
- В Windows, когда программа открывает файл, другие программы не могут его открыть или удалить
- Вывод в файл совершается не сразу, а через вспомогательный буфер. При отсутствии вызова fclose информация из него может не попасть в файл

```
int feof(FILE *f);
```
Функция *feof* проверяет наличие установленного признака конца файла. Она возвращает ненулевое значение, если обнаружен установленный признак конца файла, иначе ноль. Функция feof возвращает отличное от нуля значение, если последняя файловая операция не была выполнена из-за достижения конца файла.

```
int ferror(FILE *f);
```
Функция *ferror* проверяет наличие ошибок. Возврат нуля означает отсутствие ошибок, а ненулевая величина указывает на наличие ошибки

165. Основные функции для обработки текстовых файлов (fprintf, fscanf, fgets, rewind)

```
int fprintf(FILE *f, const char *format, ...); // запись в файл

int fscanf(FILE *f, const char *format, ...); // чтение файла
```

```
char* fgets(char *s, int n, FILE *f); // безопасное чтение n символов файла

void rewind(FILE *f); // позволяет начать обработку файла с начала
```

167. Предопределенные файловые переменные.

При инициализации программы библиотека ввода/вывода заводит три файловые переменные *stdin, stdout и stderr* для:

- *Стандартного потока ввода* (сокращение от standard input). Обычно этот поток связан с клавиатурой. Программа может читать из него данные
- *Стандартного потока вывода* (сокращение от standard output). Обычно этот поток связан с дисплеем. Программа может выводить в него данные
- *Стандартного потока ошибок* (сокращение от standard error). Обычно этот поток так же связан с дисплеем. Программа может выводить в него сообщения об ошибках

168. Организация работы с ресурсами.

Подход типичного студента: выполнение функции, проверка, если ошибка - возврат кода ошибки
Проблема: много точек выхода - можно забыть освободить ресурсы

Структурный подход: много ветвлений
Проблема: сложно читать и поддерживать код

Организация единственной точки выхода с помощью оператора goto
Проблема: сложность использования, не рекомендуется использовать

169. Коды ошибок в стандартной библиотеке.

Большинство функций стандартной библиотеки при возникновении ошибки возвращают отрицательное число или NULL и записывают в глобальную переменную errno код ошибки. Эта переменная определена в заголовочном файле errno.h. В случае успешного выполнения функции эта переменная просто не изменяется и может содержать любой мусор, поэтому проверять ее имеет смысл лишь в случае, когда ошибка действительно произошла.

```
void perror(const char *message);
```

Функция perror преобразует значение глобальной переменной errno в строку и записывает эту строку в stderr.

```
char* strerror(int errnum);
```

Функция strerror возвращает указатель на строку, содержащую системное сообщение об ошибке, связанной со значением errnum. Эту строку не следует менять ни при каких обстоятельствах.

31. Двоичные файлы

156. Что такое файл?

157. Перечислите основные свойства файла.

158. Что такое файловая система?

159. Назначение файловой системы?

160. Классификация файлов.

162. Двоичные файлы.

Двоичные файлы – последовательность произвольных байтов, они часто имеют сложную внутреннюю структуру.

163. Сравните текстовые и двоичные файлы.

164. Последовательность действий при работе с файлами (fopen, fclose, feof, ferror)

166. Основные функции для обработки двоичных файлов (fread, fwrite, fseek, ftell).

```
size_t fwrite(const void *ptr, size_t size, size_t count, FILE *f);
```

Функция *fwrite* записывает в файл, связанный с файловой переменной f, данные из буфера ptr. Количество элементов в буфере ptr указывается в переменной count, а размер каждого элемента – в переменной size. Функция fwrite возвращает количество удачно записанных элементов.

```
size_t fread(void *ptr, size_t size, size_t count, FILE *f);
```

Функция *fread* считывает из файла, связанного с файловой переменной f, данные и помещает их в буфер ptr. Количество считываемых элементов буфера указывается в переменной count, а размер каждого элемента – в переменной size.
Функция возвращает число удачно прочитанных элементов. Если возвращаемое значение отличается от количества элементов, значит произошла ошибка или был достигнут конец файла.

```
int fseek(FILE *f, long int offset, int origin);
```
Функция fseek устанавливает внутренний указатель положения в файле, связанном с файловой переменной f, в новую позицию offsetотносительно origin.
origin может принимать три значения:

- SEEK_SET начало файла
- SEEK_CUR текущее положение файла
- SEEK_END конец файла

```
long int ftell(FILE *f);
```
Функция ftell возвращает текущее положение внутреннего указателя в файле, связанном с файловой переменной f. Функция ftell возвращает -1L в случае возникновения ошибки. Для двоичных файлов возвращается значение, соответствующее количеству байт от начала файла. Для текстовых файлов это значение может не соответствовать точному количеству байт от начала файла.

167. Предопределенные файловые переменные.

168. Организация работы с ресурсами.

169. Коды ошибок в стандартной библиотеке.

32 Организация интерфейсов файла регулярной структуры над двоичным файлом (типизированный файл)

156. Что такое файл?

157. Перечислите основные свойства файла.

158. Что такое файловая система?

159. Назначение файловой системы?

160. Классификация файлов.

Дайте определение типизированному файлу.

Это структура данных, в которой:

- Все элементы «якобы» одного типа
- Существует «якобы» произвольный доступ (сдвиг каретки «якобы» за const время)
- «Якобы» лежит на диске одним куском подряд
  Типизированный файл можно считать массивом на диске

163. Сравните текстовые и двоичные файлы.

Минимальный набор функций для работы с типизированным файлом

```
Чтение записи с текущей позиции
int rf_read(FILE *f, record_t *rec);

Запись записи в текущую позицию
int rf_write(FILE *f, const record_t *rec);

Установка "курсора" на позицию pos относительно начала файла
int rf_set_pos(FILE *f, size_t pos);

Установка "курсора" на позицию pos относительно origin
int rf_set_pos_ex(FILE *f, size_t pos, int origin);

Возвращает позицию курсора
```

```
int rf_get_pos(FILE *f, size_t *pos);

Возвращает размер файла в записях
int rf_file_size(FILE *f, size_t *n_recs);

Обрезает файл до текущей позиции
int rf_truncate(FILE *f);
```

167. Предопределенные файловые переменные.

168. Организация работы с ресурсами.

169. Коды ошибок в стандартной библиотеке.

33. Этапы получения исполняемого файла

170. Что такое транслятор? Какие функции выполняет транслятор?

Будем называть транслятором программу-переводчик с языка высокого уровня на язык машины

Команда

```
gcc -std=c99 -Wall -Werror –S main.i
```

Трансляция программы сначала на язык ассемблера позволяет:

- упростить реализацию и отладку транслятора
- повысить его переносимость с одной платформы на другую

171. Что такое интерпретатор? Какие функции выполняет интерпретатор?

Интерпретатор - программа, выполняющая интерпретацию. Интерпретация - построчный анализ, обработка и выполнение исходного кода программы или запроса

172. Что такое препроцессор? Какие функции выполняет препроцессор?

Будем называть препроцессором программу, осуществляющую препроцессинг — комплекс действий над текстом на языке высокого уровня перед основной трансляцией

Препроцессор выполняет следующие действия:

- удаление комментариев
- вставку файлов (директива include)
- текстовые замены (по-другому говорят - раскрытие макросов, директива define)
- условную компиляцию (директива if)

Команда

```
gcc –E main.c > main.i
```

173. Что такое компоновщик? Какие функции выполняет компоновщик?

Будем называть компоновщиком программу, преобразующую набор оттранслированных модулей в единую программу.

В процессе получения исполняемого файла компоновщик решает несколько задач

- объединяет несколько объектных файлов в единый исполняемый файл
- выполняет связывание переменных и функций, которые требуются очередному объектному файлу, но находятся где-то в другом месте
- добавляет специальный код, который подготавливает окружение для вызова функции main, а после ее завершения выполняет обратные действия

Команда

```
gcc –o main.exe main.o
```

174. Основные шаги получения исполняемого файла.

- Обработка препроцессором
- Трансляция на язык ассемблера
- Ассемблирование в объектный файл: `gcc –c main.s`
- Компоновка

175. Что такое единица трансляции?

Файл, получаемый в результате работы препроцессора, называется единицей трансляции

176. Что такое объектный файл?

Объектный файл - двоичный файл получаемый после ассемблирования

Объектный файл представляет собой блоки машинного кода и данных с неопределенными адресами ссылок на данные и подпрограммы в других объектных модулях, а также список своих подпрограмм и данных

Объектный файл из заголовков и секций.
Заголовки содержат служебную информацию, описывающую различные свойства объектного файла и его структуру.
Секции содержат данные в широком смысле этого слова

- код (.text)
- данные (.data, .rodata, .bss)
- служебная информация

177. Что такое исполняемый файл?

Исполняемый файл - файл, содержащий программу в виде, в котором она может быть (после загрузки в память и настройки по месту) исполнена компьютером.
Получается после этапа компоновки
Обычно исполняемый файл (как и объектный файл) состоит из заголовков и секций.

178. Чем исполняемый файл отличается от объектного файла?

Исполняемый файл представляет собой готовый к выполнению продукт, тогда как объектный файл является частью процесса разработки и требует дополнительной обработки для создания исполняемого файла.

У каждого модуля программы свой объектный файл
А исполняемый файл один на всю программу

34. Многофайловая организация проекта

190. Сравните однофайловую и многофайловую организации проекта.

Недостатки однофайловых проектов

- Ориентирование в тексте программы становится сложным
- Одновременная работа над программой нескольких программистов становится неэффективной
- Даже при локальном изменении перекомпилируется весь проект

Преимущества многофайловой организации проекта

- Код программы более удобочитаем
- Позволяет распределить работу над проектом между несколькими программистами
- Сокращает время повторной компиляции

191. Для чего нужны заголовочные файлы?

Чтобы связать файлы проекта нужно использовать *заголовочные файлы*, где будут описаны функции файлов, новые типы данных и прочие компоненты
Заголовочный файл нужно включать во все файлы проекта
`#include “header.h”`

Компиляция многофайлового проекта

- Создать объектные файлы для каждого файла проекта с помощью ключа -c
- Создание исполняемого файла проекта из всех объектных файлов

192. Что такое include guard? Для чего нужна эта конструкция? Как она работает?

include guard - это особая конструкция, применяемая для избежания проблем с «двойным подключением» при использовании директивы компилятора #include

```
// Файл grandfather.h
#ifndef H_GRANDFATHER
#define H_GRANDFATHER
...
#endif
```

193. Ошибки компиляции.

Ошибки компиляции — это проблемы, обнаруженные компилятором во время процесса компиляции исходного кода, которые приводят к невозможности успешного создания исполняемого файла

Может возникнуть, если заголовочный файл с определением функций не был включен в программу

194. Ошибки компоновки.

Ошибки компоновки возникают в процессе компоновки (linking) объектных файлов и библиотек в один исполняемый файл.

Может возникнуть, если при компоновке были указаны не все объектные файлы программы

195. Предупреждения компилятора.

Предупреждения компилятора — это сообщения, которые компилятор выводит во время процесса компиляции, предупреждая разработчика о потенциальных проблемах в коде, которые не приводят к ошибкам компиляции, но могут вызвать ошибки во время выполнения программы или ухудшить её производительность

35. Битовые поля и битовые операции

152. Что такое битовое поле?

Битовое поле - особый тип структуры, определяющей, какую длину имеет каждый член в битах.

Стандартный вид объявления битовых полей следующий:

```
struct имя_структуры
{
    тип имя1: длина;
    тип имя2: длина;
    …
    тип имяN: длина;
};
```

Битовые поля должны объявляться как целые, unsigned или signed.

153. Описание битовых полей.

```
#define HIDE 0
#define SHOW 1
#define BORDER 1
#define CAPTION 2
#define RED 1
#define GREEN 2
#define BLUE 4

struct wnd_flags
{
    unsigned char show : 1;
    unsigned char style : 2;
    unsigned char color : 3;
};

struct window
{
    struct wnd_flags flags;
};
...
```

```
w.flags.show = SHOW;
w.flags.style = BORDER;
w.flags.color = RED | BLUE;
```

154. Особенности использования битовых полей.

Нельзя работать, как с отдельным объектом

```
w.flags = 5; // ошибка компиляции

unsigned char f;
f = w.flags; // ошибка компиляции
```

Битовые поля можно описать и в обычной структуре

155. Сравните использование битовых полей и битовых операций. Приведите примеры.

Битовые поля плохо переносят с одной платформы на другую, в отличие от битовых операций
Битовые легче описывать и с ними легче взаимодействовать, чем с битовыми операциями

36. Область видимости

196. Что такое область видимости?

Область видимости имени – это часть текста программы, в пределах которой имя может быть использовано.

197. Какие виды областей видимости есть в языке Си?

В языке Си выделяют следующие области видимости

- блок
- файл
- функция
- прототип функции

Область видимости «блок»

В языке Си блоком считается последовательность объявлений, определений и операторов, заключенная в фигурные скобки.
Существуют два вида блоков:

- составной оператор
- определение функции

Блоки могут включать в себя составные операторы, но не определения функций

- Переменная, определенная внутри блока, имеет область видимости в пределах блока
- Формальные параметры функции имеют в качестве области видимости блок, составляющий тело функции

Область видимости «файл»

- Область видимости в пределах файла имеют имена, описанные за пределами какой бы то ни было функции
- Переменная с областью видимости в пределах файла видна на протяжении от точки ее описания и до конца файла, содержащего это определение
- Имя функции всегда имеет файловую область видимости

Область видимости «функция»

- Метки (goto) - это единственные идентификаторы, область действия которых - функция.
- Метки видны из любого места функции, в которой они описаны.
- В пределах функции имена меток должны быть уникальными

Область видимости «прототип функции»
Область видимости в пределах прототипа функции применяется к именам переменных, которые используются в прототипах функций. Область видимости в пределах прототипа функции простирается от точки, в которой объявлена переменная, до конца объявления прототипа.

198. Правила перекрытия областей видимости.

Переменные, определенные внутри некоторого блока, будут доступны из всех блоков, вложенных в данный
Возможно определить в одном из вложенных блоков переменную с именем, совпадающим с именем одной из "внешних" переменных

```
#include <stdio.h>

int x = 10;

void func() {
    int x = 20;
    printf("Inside func: x = %d\n", x);
}

int main() {
    printf("In main before func: x = %d\n", x);
    func();
    printf("In main after func: x = %d\n", x);
    return 0;
}
// In main before func: x = 10
// Inside func: x = 20
// In main after func: x = 10
```

37. Время жизни

199. Что такое время жизни?

Время жизни – это интервал времени выполнения программы, в течение которого «программный объект» существует

200. Какие виды времени жизни есть в языке Си?

В языке Си время жизни «программного объекта» делится на три категории

- глобальное (по стандарту - статическое (англ. static))
- локальное (по стандарту - автоматическое (англ. automatic))
- динамическое (по стандарту - выделенное (англ. allocated)).

Глобальное время жизни

Если «программный объект» имеет глобальное время жизни, он существует на протяжении выполнения всей программы

Примерами таких «программных объектов» могут быть функции и переменные, определенные вне каких либо функций

Локальное время жизни

Локальным временем жизни обладают «программные объекты», область видимости которых ограничена блоком.

Такие объекты создаются при каждом входе в блок, где они определяются.

Они уничтожаются при выходе из этого «родительского» блока.

Примерами таких переменных являются локальные переменные и параметры функции

Для хранения локальных переменных используется так называемая *автоматическая память*

Плюсы

- Память под локальные переменные выделяет и освобождает компилятор.

Минусы

- Время жизни локальной переменной "ограничено" блоком, в котором она определена
- Размер размещаемых в автоматической памяти объектов должен быть известен на этапе компиляции
- Размер автоматической памяти в большинстве случаев ограничен

Динамическое время жизни

Время жизни «выделенных» объектов длится с момента выделения памяти и заканчивается в момент ее освобождения.
В Си нет переменных, обладающих динамическим временем жизни. Динамическое выделение выполняется программистом «вручную» с помощью соответствующих функций. Единственные способ «добраться» до выделенной динамической памяти – использование указателей.

38. Библиотеки

179. Что такое библиотека? Как распространяются библиотеки?

Библиотека - набор специальным образом оформленных объектных файлов

Библиотека включает в себя

- заголовочный файл
- откомпилированный файл самой библиотеки
  - библиотеки меняются редко – нет причин перекомпилировать каждый раз
  - двоичный код предотвращает доступ к исходному коду

180. Какие виды библиотек существуют?

- статические
- динамические

181. Проанализируйте преимущества и недостатки статических и динамических библиотек.

Статические библиотеки связываются с программой в момент компоновки. Код библиотеки помещается в исполняемый файл

Плюсы

- Исполняемый файл включает в себя все необходимое
- Не возникает проблем с использованием не той версии библиотеки

Минусы

- «Размер»
- При обновлении библиотеки программу нужно пересобрать

Динамические библиотеки. Подпрограммы из библиотеки загружаются в приложение во время выполнения. Код библиотеки не помещается в исполняемый файл

Плюсы

- Несколько программ могут «разделять» одну библиотеку
- Меньший размер приложения (по сравнению с приложением со статической библиотекой)
- Модернизация библиотеки не требует перекомпиляции программы
- Могут использовать программы на разных языках

Минусы

- Требуется наличие библиотеки на компьютере

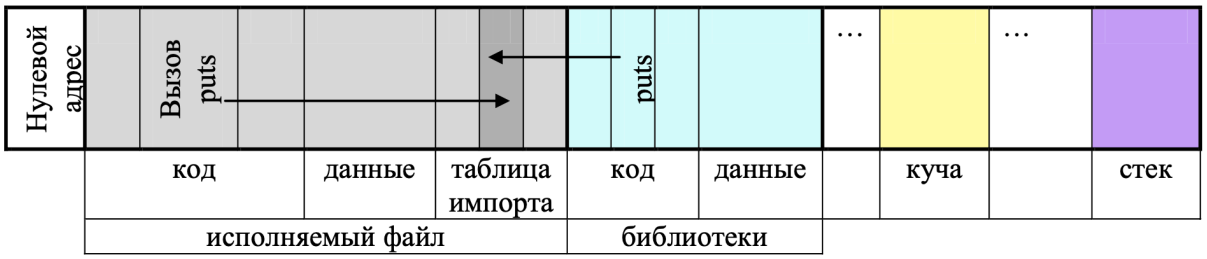
- Версионность библиотек

39. Запуск исполняемого файла в операционной системе. Абстрактное адресное пространство программы

182. Основные шаги «превращения» исполняемого файла в процесс операционной системы.

- считывание программы в озу
- загрузка библиотек
- настройка ссылок (поправляем ссылки на функции динамических библиотек через таблицу импорта)
- планирование процесса

183. Абстрактная память процесс и ее организация.



Когда ОС загружает программу в оперативную память, программа попадает в абстрактную память
Приложение кажется, что в памяти присутствует только оно и необходимые библиотеки
В нулевом адресе не может быть ничего расположено, позволяет

- ввести нулевой указатель
- используется для обнаружения обращений по невалидным адресам

Данные - глобальные переменные
Таблица импорта - вызов функций динамической библиотеки - ассоциативный массив
Куча - используется для динамического выделения памяти

40. Аппаратный стек и его использование в программе на Си

184. Использование аппаратного стека в программе на Си.

- *вызова функции*
  call name
  - поместить в стек адрес команды, следующей за командой call
  - передать управление по адресу метки name

- *возврата из функции*
  ret
  - извлечь из стека адрес возврата address
  - передать управление на адрес address

- *передачи параметров в функцию*
  соглашение о вызове:
  - расположение входных данных
  - порядок передачи параметров
  - какая из сторон очищает стек
  - etc

  cdecl
  - аргументы передаются через стек, справа налево
  - очистку стека производит вызывающая сторона
  - результат функции возвращается через регистр EAX, кроме слишком больших объектов

- *выделения и освобождения памяти под локальные переменные*

185. Что такое кадр стека?

Стековый кадр (фрейм) - механизм передачи аргументов и выделения временной памяти с использованием аппаратного стека

186. Для чего используется кадр стека в программе на Си?

В стековом кадре размещаются:

- значения фактических аргументов функции
- адрес возврата
- локальные переменные
- иные данные, связанные с вызовом функции

187. Преимущества и недостатки использования кадра стека.

Плюсы

- Удобство и простота использования.

Минусы

- Производительность. Передача данных через память без необходимости замедляет выполнение программы
- Безопасность. Стековый кадр смешивает данные приложения с критическими данными - указателями, значениями регистров и адресами возврата.

188. Что такое соглашение о вызове?

соглашение о вызове устанавливает:

- расположение входных данных
- порядок передачи параметров
- какая из сторон очищает стек

189. Какое соглашение о вызове используется в языке Си? Какие соглашения о вызове Вы знаете?

cdecl

- аргументы передаются через стек, справа налево
- очистку стека производит вызывающая сторона
- результат функции возвращается через регистр EAX, кроме слишком больших объектов

41. Неопределенное поведение

226. Какие виды «неопределенного» поведения есть в языке Си?

- *Unspecified behavior* (неспецифицированное поведение) - Стандарт предлагает несколько вариантов на выбор. Компилятор может реализовать любой вариант. При этом на вход компилятора подается корректная программа
- *Implementation-defined behavior* (поведение, зависящее от реализации) - Похоже на неспецифицированное (unspescified) поведение, но в документации к компилятору должно быть указано, какое именно поведение реализовано
- *Undefined behavior* (неопределенное поведение) - Такое поведение возникает как следствие неправильно написанной программы или некорректных данных. Стандарт ничего не гарантирует, может случиться все что угодно

227. Почему «неопределенное» поведение присутствует в языке Си?

- освободить разработчиков компиляторов от необходимости обнаруживать ошибки, которые трудно диагностировать
- избежать предпочтения одной стратегии реализации другой
- отметить области языка для расширения языка

228. Какой из видов «неопределенного» поведения является самым опасным? Чем он опасен?

Самое опасное “неопределенного поведения” - undefined behavior
Оно опасно тем, что мы не можем предсказать, как себя поведет программа, и может случиться все, что угодно

229. Как бороться с неопределенным поведением?

- Включайте все предупреждения компилятора, внимательно читайте их
- Используйте возможности компилятора (-ftrapv) (встраивает специальные ассемблерные инструкции в Ваш код, которые проверяют числовые типы данных после арифметических операций, проверяет переполнение типов)
- Используйте несколько компиляторов
- Используйте статические анализаторы кода (например, clang)

- Используйте инструменты такие как valgrind (отладка памяти, обнаружения утечек памяти, а также профилирования), Doctor Memory и др.
- Используйте утверждения

230. Приведите примеры неопределенного поведения.

- Использование неинициализированных переменных
- Переполнение знаковых целых типов
- Выход за границы массива
- Использование «диких» указателей

231. Приведите примеры поведения, зависящего от реализации.

результат x % y, где x и y целые, а y отрицательное, может быть как положительным, так и отрицательным

232. Приведите примеры неспецифицированного поведения.

Все аргументы функции должны быть вычислены до вызова функции, но они могут быть вычислены в любом порядке

219. Что такое побочный эффект?

Побочный эффект - это все выражения и операции, которые изменяют состояние среды выполнения

220. Какие выражения стандарт c99 относит к выражениям с побочным эффектом?

- Модификация данных
- Обращение к переменным, объявленным как volatile
- Вызов системной функции, которая производит побочные эффекты (например, файловый ввод или вывод).
- Вызов функций, выполняющих любое из вышеперечисленных действий

221. Почему порядок вычисления подвыражений в языке Си не определен?

Он не определен, чтобы увеличить эффективность программы. Компилятор сам решает, в каком порядке будет быстрее вычислить выражения

222. Порядок вычисления каких выражения в языке Си определен?

Всех выражений, включающих операнды

223. Что такое точка следования?

Точка следования – это точка в программе, в которой программист знает какие выражения (или подвыражения) уже вычислены, а какие выражения (или подвыражения) еще нет

224. Какие точки следования выделяет стандарт c99?

Определены следующие точки следования:

- Между вычислением левого и правого операндов в операциях &&, || и ","

```
*p++ != 0 && *q++ != 0
```

- Между вычислением первого и второго или третьего операндов в тернарной операции

```
a = (*p++) ? (*p++) : 0;
```

- В конце полного выражения

```
a = b;
if ()
switch ()
while ()
do{} while()
for ( x; y; z)
return x
```

- Перед входом в вызываемую функцию

  Порядок, в котором вычисляются аргументы не определен, но эта точка следования гарантирует, что все ее побочные эффекты проявятся на момент входа в функцию

- В объявлении с инициализацией на момент завершения вычисления инициализирующего значения

```
int a = (1 + i++);
```

225. Почему необходимо избегать выражений, которые дают разный результат в зависимости от порядка их вычисления?

Их нужно избегать, так как это приводит к непредсказуемому поведению программы, которое может меняться при смене компилятора

42. Связывание

201. Что такое связывание?

Связывание определяет область программы (функция, файл, вся программа целиком), в которой «программный объект» может быть доступен другим функциям программы

202. Какие виды связывания есть в языке Си?

Стандарт языка Си определяет три формы связывания

- внешнее (external)
- внутреннее (internal)
- никакое (none)

203. Как связывание влияет на «свойства» объектного/исполняемого файла? Что это за «свойства»?

Имена с *внешним связыванием* доступны во всей программе. Подобные имена «экспортируются» из объектного файла, создаваемого компилятором.

Имена с *внутренним связыванием* доступны только в пределах файла, в котором они определены, но могут «разделяться» между всеми функциями этого файла.

Имена *без связывания* принадлежат одной функции и не могут разделяться вообще.

43. Классы памяти. Общие понятия, классы памяти auto и static для переменных

204. Какими характеристиками (область видимости, время жизни, связывание) обладает переменная в зависимости от места своего определения?

Время жизни, область видимости и связывание переменной зависят от места ее определения. По умолчанию

```
int i; // глобальная переменная
- Глобальное время жизни
- Файловая область видимости
- Внешнее связывание

{
    int i; // локальная переменная
    ...
- Локальное время жизни
- Видимость в блоке
- Отсутствие связывания
```

206. Какие классы памяти есть в языке Си?

В языке Си существует четыре класса памяти

- auto
- static
- extern
- register

207. Для чего нужны классы памяти?

Управлять временем жизни, областью видимости и связыванием переменной (до определенной степени) можно с помощью так называемых *классов памяти*

208. Какие классы памяти можно использовать с переменными? С функциями?

Переменные: auto, static, extern, register
Функции: static, extern

209. Сколько классов памяти может быть у переменной? У функции?

Переменная - 1
Функция - 1

210. Какие классы памяти по умолчанию есть у переменной? У функции?

Переменная: В блоке или заголовке функции: auto
Функция: extern

211. Расскажите о классе памяти auto.

Применим только к переменным, определенным в блоке.

```
int main(void)
{
    auto int i;
```

Переменная, принадлежащая к классу auto, имеет локальное время жизни, видимость в пределах блока, не имеет связывания.

По умолчанию любая переменная, объявленная в блоке или в заголовке функции, относится к классу автоматической памяти

212. Расскажите о классе памяти static.

- Для переменной вне какого-либо блока: изменяет связывание этой переменной на внутреннее, глобальное время жизни, область видимости в пределах файла - скрывает переменную в файле

- Для переменной в блоке: изменяет время жизни на глобальное, область видимости в пределах блока, отсутствие связывания
    - Такая переменная сохраняет свое значение после выхода из блока.
    - Инициализируется только один раз.
    - Если функция вызывается рекурсивно, это порождает новый набор локальных переменных, в то время как статическая переменная разделяется между всеми вызовами

216. Особенности совместного использования ключевых слов static и extern.

Если компилятор встречает объявление переменной с extern и до этого он еще не видел объявление переменной, то он предполагает, что у переменной внешнее связывание
Если потом оказывается, что у этой переменной связывание внутреннее, то возникает ошибка компиляции

44. Классы памяти. Общие понятия, классы памяти extern и register для переменных

204. Какими характеристиками (область видимости, время жизни, связывание) обладает переменная в зависимости от места своего определения?

206. Какие классы памяти есть в языке Си?

207. Для чего нужны классы памяти?

208. Какие классы памяти можно использовать с переменными? С функциями?

209. Сколько классов памяти может быть у переменной? У функции?

210. Какие классы памяти по умолчанию есть у переменной? У функции?

213. Расскажите о классе памяти extern.

- Объявлений (extern int number;) может быть сколько угодно
- Определение (int number;) должно быть только одно
- Объявления и определение должны быть одинакового типа

Замечание

```
extern int number = 5; // определение
```

Используется для переменных определенных как в блоке, так и вне блока.

```
extern int i; // Глобальная переменная
```

- глобальное время жизни
- файловая область видимости
- связывание непонятное

```
{
    extern int i; // Локальная переменная
    ...
```

- глобальное время жизни
- видимость в блоке

- связывание непонятное, определяется по определению переменной

214. Расскажите о классе памяти register.

Использование класса памяти register – просьба (!) к компилятору разместить переменную не в памяти, а в регистре процессора

- Используется только для переменных, определенных в блоке
- Задает локальное время жизни, видимость в блоке и отсутствие связывания
- Обычно не используется (сейчас машины мощнее, поэтому компилятор сам размещает переменные)

К переменным с классом памяти register нельзя применять операцию получения адреса &

215. Для чего используется ключевое слово extern?

Помогает разделить переменную между несколькими файлами

216. Особенности совместного использования ключевых слов static и extern.

45. Классы памяти. Общие понятия, классы памяти для функций

204. Какими характеристиками (область видимости, время жизни, связывание) обладает переменная в зависимости от места своего определения?

206. Какие классы памяти есть в языке Си?

207. Для чего нужны классы памяти?

208. Какие классы памяти можно использовать с переменными? С функциями?

209. Сколько классов памяти может быть у переменной? У функции?

210. Какие классы памяти по умолчанию есть у переменной? У функции?

205. Какими характеристиками (область видимости, время жизни, связывание) обладает функция в зависимости от места своего определения?

К функциям могут применяться классы памяти static и extern

```
extern int f(int i);
```
f имеет внешнее связывание. Может вызываться из других файлов.

```
static int g(int i);
```
g имеет внутреннее связывание. Из других файлов вызываться не может

```
int h(int i);
```
h имеет внешнее связывание (по умолчанию). Может вызываться из других файлов.

Использование класса памяти static для функций полезно потому что:

- Функции, определенные со static, не видны в других файлах и могут безболезненно изменяться (инкапсуляция).
- Так как функция имеет внутреннее связывание, ее имя может использоваться в других файлах

Аргументы функций по умолчанию имеют класс памяти auto. Единственный другой класс памяти, который может использоваться с параметрами функций - register

46. Глобальные переменные. Журналирование

217. Какими недостатки есть у использования глобальных переменных?

Недостатки использования функций, которые используют глобальные переменные:

- Если глобальная переменная получает неверное значение, трудно понять какая функция работает неправильно
- Изменение глобальной переменной требует проверки правильности работы всех функций, которые ее используют
- Функции, которые используют глобальные переменные, трудно использовать в других программах

218. Журналирование, подходы к реализации.

Журналирование - один из подходов к обработке ошибочных ситуаций

```
// log.c
#include "log.h"

static FILE *flog;

int log_init(const char *name)
{
    flog = fopen(name, "w");
    if(!flog)
        return 1;
    return 0;
}

FILE* log_get(void)
{
    return flog;
}

void log_close(void)
{
    fclose(flog);
}
```

```
// log.h
```

```
#ifndef __LOG__H__
#define __LOG__H__

#include <stdio.h>

int log_init(const char *name);
FILE* log_get(void);
void log_close(void);

#endif  //  __LOG__H__
```

```
// main.c
#include <stdio.h>
#include "log.h"

void func_1(void)
{
    fprintf(log_get(), "func_1\n");
}

void func_2(int a)
{
    fprintf(log_get(), "func_2(%d)\n", a);
}

int main(void)
{
    if(log_init("test.log"))
    {
        fprintf(stderr, "Could not create log file\n");
        return -1;
    }

    func_1();
    func_2(5);
    log_close();
    return 0;
}
```