Вопросы к экзамену - 1 курс 1 семестр

1. Sizeof(), диапазон значений типа (Пример: unsigned char - от 0 до 255)

`sizeof()` - оператор, возвращающий размер типа данных или переменной в байтах.

unsigned char: занимает 1 байт (8 бит), диапазон значений: 0 до 255 (2^8 вариантов).

Примеры использования:

```c
printf("Размер int: %lu байт\n", sizeof(int));
printf("Размер массива: %lu байт\n", sizeof(arr));

int *ptr = (int*)malloc(10 * sizeof(int));  // выделение памяти
```

Диапазоны основных типов:

- `char`: -128 до 127
- `unsigned char`: 0 до 255
- `short`: -32,768 до 32,767
- `unsigned short`: 0 до 65,535
- `int`: -2,147,483,648 до 2,147,483,647 (обычно 4 байта)
- `unsigned int`: 0 до 4,294,967,295

---

2. Выражения (a+b*c)

Порядок выполнения согласно приоритету операторов:

1. Сначала умножение: `b*c`
2. Затем сложение: `a + результат`

Пример:

```c
```

```c
int a = 2, b = 3, c = 4;
int result = a + b * c;  // result = 2 + 12 = 14 (не 20!)
```

---

3. Графические схемы алгоритмов = блок-схемы алгоритмов (Линейный алгоритм, разветвление, итд.)

Основные элементы блок-схем:

- **Овал** - начало/конец алгоритма
- **Параллелограмм** - ввод/вывод данных
- **Прямоугольник** - процесс (операции)
- **Ромб** - условие (ветвление)
- **Стрелки** - направление выполнения

Виды алгоритмов:

1. Линейный алгоритм - последовательное выполнение операций

```
[Начало] → [Операция 1] → [Операция 2] → [Операция 3] → [Конец]
```

2. Разветвляющийся алгоритм - выбор ветви по условию

```
[Начало] → [Условие?] → Да → [Действие A] → [Конец]
                      ↓ Нет
                [Действие B] → [Конец]
```

3. Циклический алгоритм - повторение действий

```
[Начало] → [Условие?] → Нет → [Конец]
              ↓ Да
         [Операция] ──┘
```

---

4. Динамическая память. Выделение и освобождение динамической памяти (malloc, free)

malloc(size) - выделяет блок памяти заданного размера в куче, возвращает указатель (void*).

free(ptr) - освобождает ранее выделенную память.

Примеры:

```c
// Выделение памяти для одного int
int *ptr = (int*)malloc(sizeof(int));
if (ptr != NULL) {
    *ptr = 42;
    free(ptr);  // освобождение памяти
}

// Выделение памяти для массива
int *arr = (int*)malloc(10 * sizeof(int));
if (arr != NULL) {
    for (int i = 0; i < 10; i++) {
        arr[i] = i * 2;
    }
    free(arr);
}

// calloc - выделение с обнулением
int *zeros = (int*)calloc(10, sizeof(int));

// realloc - изменение размера
arr = (int*)realloc(arr, 20 * sizeof(int));
```

Важно:

- Всегда проверяйте результат malloc на NULL
- Всегда освобождайте выделенную память во избежание утечек
- Не используйте указатель после free

---

5. Идентификаторы (DlinnoeMnemonicheskoeImya1)

Правила именования идентификаторов в C:

- Начинаются с буквы (a-z, A-Z) или подчеркивания (_)
- Могут содержать буквы, цифры (0-9), подчеркивания
- Регистрозависимые (`count` и `Count` - разные переменные)
- Не могут быть ключевыми словами (int, if, while, и т.д.)

Примеры:

```c
// Правильные идентификаторы
int count;
float _temperature;
char firstName123;
double DlinnoeMnemonicheskoeImya1;

// Неправильные идентификаторы
int 2name;       // начинается с цифры
float my-var;    // содержит дефис
char int;        // ключевое слово
```

---

6. Массивы. Работа с одномерными массивами. Работа с двумерными массивами (int a[10];)

Одномерные массивы:

```c
// Объявление
int a[10];               // массив из 10 элементов
int b[5] = {1, 2, 3, 4, 5};  // с инициализацией
int c[] = {10, 20, 30}; // размер определяется автоматически

// Доступ к элементам
a[0] = 5;                // первый элемент (индекс 0)
a[9] = 100;              // последний элемент (индекс 9)

// Заполнение массива
for (int i = 0; i < 10; i++) {
    a[i] = i * 2;
}

// Вывод массива
for (int i = 0; i < 10; i++) {
    printf("%d ", a[i]);
}
```

Двумерные массивы:

```c
```

```c
// Объявление
int matrix[3][4];       // 3 строки, 4 столбца

// Инициализация
int m[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};

// Доступ к элементам
matrix[0][0] = 10;      // первый элемент
matrix[1][2] = 7;       // 2-я строка, 3-й столбец

// Заполнение двумерного массива
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        matrix[i][j] = i * 4 + j;
    }
}

// Вывод двумерного массива
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        printf("%3d ", matrix[i][j]);
    }
    printf("\n");
}
```

---

7. Область видимости переменных. Локальные и глобальные переменные (int x; void f() { int x; })

Глобальные переменные:

```c
c
```

```c
int globalVar = 100;  // видна во всех функциях

void function1() {
    printf("%d\n", globalVar);  // доступна здесь
}

void function2() {
    globalVar = 200;  // можно изменить
}

int main() {
    printf("%d\n", globalVar);  // доступна здесь
    return 0;
}
```

Локальные переменные:

```c
void f() {
    int x = 10;  // локальная переменная, видна только внутри f()
    printf("%d\n", x);
}  // x уничтожается здесь

int main() {
    int x = 20;  // другая переменная с тем же именем
    f();         // выведет 10
    printf("%d\n", x);  // выведет 20
    return 0;
}
```

Вложенные блоки:

```c
```

```c
int main() {
    int x = 1;
    printf("Внешний x: %d\n", x);  // 1

    {
        int x = 2;  // новая локальная переменная
        printf("Внутренний x: %d\n", x);  // 2
    }

    printf("Снова внешний x: %d\n", x);  // 1
    return 0;
}
```

Правило: Локальные переменные имеют приоритет над глобальными при совпадении имен.

---

8. Объявление переменных (int a;)

Синтаксис объявления:

```c
тип имя_переменной;
тип имя_переменной = значение;  // с инициализацией
```

Примеры:

```c
```

```c
// Простое объявление
int a;
float x;
char c;

// С инициализацией
int b = 10;
float y = 3.14;
char ch = 'A';

// Несколько переменных одного типа
int x, y, z;
int a = 1, b = 2, c = 3;

// Константы
const int MAX = 100;
const float PI = 3.14159;

// Модификаторы
unsigned int positive;
long long bigNumber;
short smallNumber;
```

---

9. Операторы. Приоритеты операторов (-b+sqrt(d)/2*a)

Таблица приоритетов (от высшего к низшему):

Приоритет	Операторы	Описание	Ассоциативность
1	`()` `[]` `.` `->`	Скобки, индексация, доступ	Слева направо
2	`++` `--` `!` `~` `-` `+` `*` `&` `sizeof` `(type)`	Унарные операторы	Справа налево
3	`*` `/` `%`	Умножение, деление, остаток	Слева направо
4	`+` `-`	Сложение, вычитание	Слева направо
5	`<<` `>>`	Побитовые сдвиги	Слева направо
6	`<` `<=` `>` `>=`	Сравнение	Слева направо

Приоритет	Операторы	Описание	Ассоциативность
7	`==` `!=`	Равенство	Слева направо
8	`&`	Побитовое И	Слева направо
9	`^`	Побитовое XOR	Слева направо
10	`|`	Побитовое ИЛИ	Слева направо
11	`&&`	Логическое И	Слева направо
12	`||`	Логическое ИЛИ	Слева направо
13	`?:`	Тернарный оператор	Справа налево
14	`=` `+=` `-=` `*=` `/=`	Присваивание	Справа налево
15	`,`	Запятая	Слева направо

Разбор примера: `-b+sqrt(d)/2*a`

Пошаговое выполнение:

1. `sqrt(d)` — вызов функции (приоритет 1)
2. `-b` — унарный минус (приоритет 2)
3. `sqrt(d)/2` — деление (приоритет 3)
4. `результат*a` — умножение (приоритет 3)
5. `-b+результат` — сложение (приоритет 4)

Результат: `-b + ((sqrt(d)/2)*a)`

Практические примеры:

```c
int x = 5 + 3 * 2;          // x = 11 (не 16!)
int y = 10 / 2 * 3;         // y = 15
int z = 2 + 3 * 4 - 1;      // z = 13
int a = 1 < 2 && 3 < 4;     // a = 1 (true)
```

---

10. Разветвление. Простая, усеченная, вложенная (if (a<b) min = a;)

Простое условие (полное):

```c
if (условие) {
    // выполняется если условие истинно
} else {
    // выполняется если условие ложно
}
```

Усеченное условие (без else):

```c
if (a < b) {
    min = a;
}
// если условие ложно, ничего не выполняется
```

Вложенные условия:

```c
if (a < b) {
    if (a < c) {
        min = a;
    } else {
        min = c;
    }
} else {
    if (b < c) {
        min = b;
    } else {
        min = c;
    }
}
```

Цепочка if-else if-else:

```c
```

```c
if (grade >= 90) {
    printf("Отлично\n");
} else if (grade >= 80) {
    printf("Хорошо\n");
} else if (grade >= 70) {
    printf("Удовлетворительно\n");
} else {
    printf("Неудовлетворительно\n");
}
```

Тернарный оператор (сокращенная форма):

```c
min = (a < b) ? a : b;  // если a < b, то min = a, иначе min = b
```

---

11. Оператор выбора (switch)

Синтаксис:

```c
switch (выражение) {
    case значение1:
        // код
        break;
    case значение2:
        // код
        break;
    default:
        // код по умолчанию
}
```

Примеры:

```c
```

```c
int day = 3;
switch (day) {
    case 1:
        printf("Понедельник\n");
        break;
    case 2:
        printf("Вторник\n");
        break;
    case 3:
        printf("Среда\n");
        break;
    case 4:
        printf("Четверг\n");
        break;
    case 5:
        printf("Пятница\n");
        break;
    case 6:
    case 7:
        printf("Выходной\n");
        break;
    default:
        printf("Неверный день\n");
}
```

Без break (проваливание):

```c
switch (grade) {
    case 'A':
    case 'B':
        printf("Хорошо\n");
        break;
    case 'C':
        printf("Удовлетворительно\n");
        break;
    case 'D':
    case 'F':
        printf("Плохо\n");
        break;
}
```

Важно:

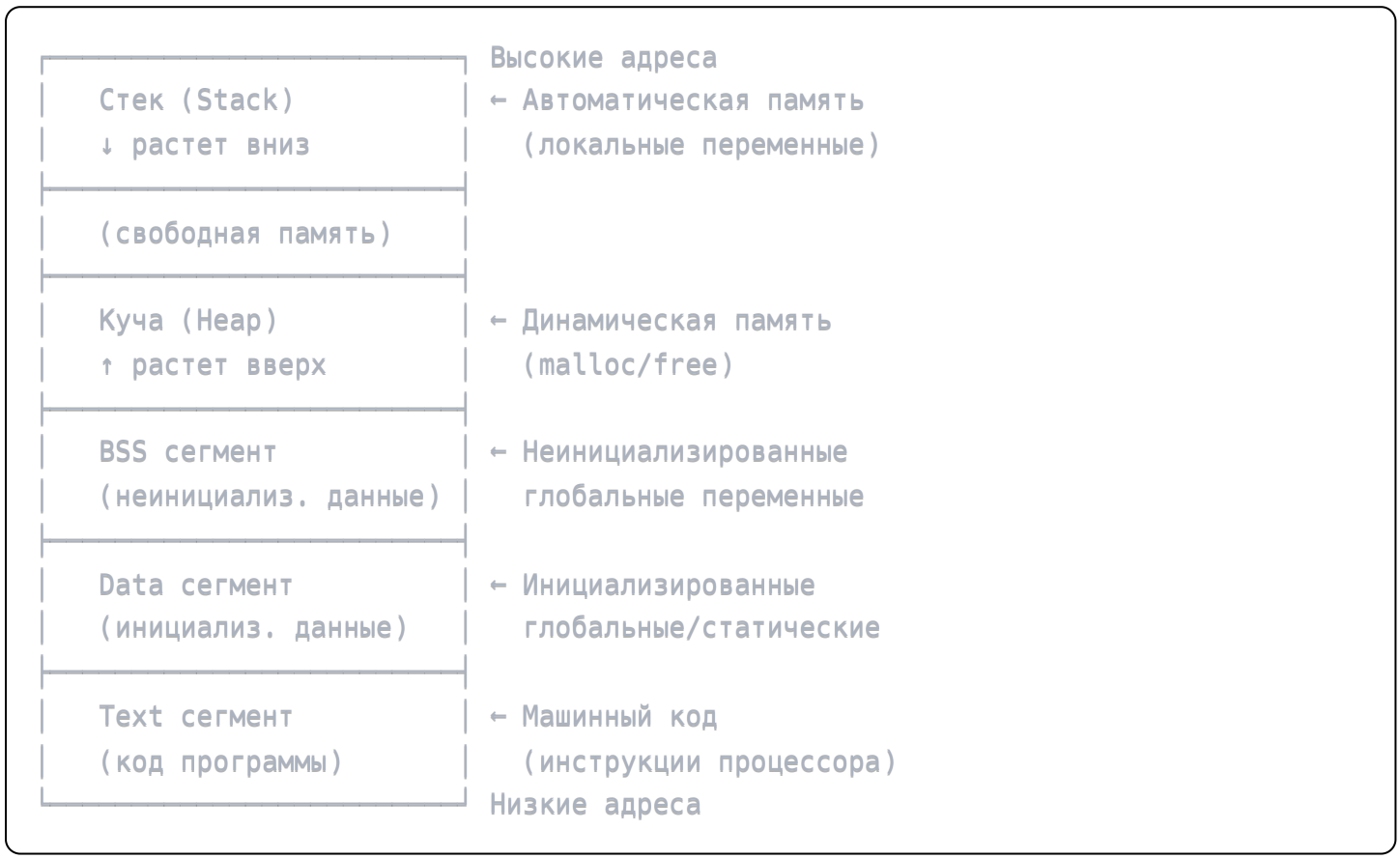
- `break` прерывает выполнение switch

- Без `break` выполнение "проваливается" в следующий case
- `default` выполняется, если ни один case не подошел

---

12. Разделы памяти во время выполнения программы: статическая, автоматическая, динамическая, машинный код

Структура памяти программы:



1. Машинный код (Text сегмент):

- Содержит исполняемые инструкции программы
- Только для чтения
- Загружается при запуске программы

2. Статическая память (Data/BSS сегменты):

```c

```

```c
int globalVar = 100;         // Data сегмент (инициализирована)
int globalUninitialized;     // BSS сегмент (не инициализирована)
static int staticVar = 50;   // Data сегмент

void func() {
    static int count = 0;    // Data сегмент, сохраняет значение
    count++;
}
```

- Выделяется при запуске программы
- Существует всё время работы программы
- Глобальные и статические переменные

3. Автоматическая память (Стек):

```c
void function() {
    int localVar = 10;       // на стеке
    char buffer[100];        // на стеке
    // освобождается автоматически при выходе из функции
}
```

- Выделяется при входе в функцию
- Освобождается при выходе из функции
- Локальные переменные и параметры функций

4. Динамическая память (Куча):

```c
int *ptr = (int*)malloc(sizeof(int) * 100);  // на куче
// используем память
free(ptr);  // освобождаем вручную
```

- Выделяется вручную (malloc, calloc)
- Освобождается вручную (free)
- Размер определяется во время выполнения

---

13. Рекурсия. Прямая и косвенная. Область применения (void f() { f(); })

Прямая рекурсия:

Функция вызывает саму себя напрямую.

```c
// Факториал
int factorial(int n) {
    if (n <= 1) return 1;       // базовый случай
    return n * factorial(n-1);  // рекурсивный вызов
}

// Числа Фибоначчи
int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n-1) + fibonacci(n-2);
}

// Сумма цифр числа
int sumDigits(int n) {
    if (n == 0) return 0;
    return n % 10 + sumDigits(n / 10);
}
```

Косвенная рекурсия:

Функция A вызывает функцию B, которая вызывает функцию A.

```c
```

```c
void functionA(int n);
void functionB(int n);

void functionA(int n) {
    if (n > 0) {
        printf("A: %d\n", n);
        functionB(n - 1);
    }
}

void functionB(int n) {
    if (n > 0) {
        printf("B: %d\n", n);
        functionA(n - 1);
    }
}
```

Область применения:

1. **Математические вычисления**: факториал, Фибоначчи, степень
2. **Обход структур данных**: деревья, графы
3. **Алгоритмы**: быстрая сортировка, бинарный поиск
4. **Задачи**: Ханойские башни, перебор вариантов

Важные элементы рекурсии:

```c
int recursiveFunction(int n) {
    // 1. Базовый случай (условие остановки)
    if (n <= 0) {
        return 0;
    }

    // 2. Рекурсивный случай
    return recursiveFunction(n - 1);
}
```

Преимущества: элегантный код, естественное решение для некоторых задач

Недостатки: расход памяти (стек), может быть медленнее итерации

---

14. Символы. ASCII. Функции обработки символов (if (isdigit(ch)) {digit = ch - '0';})

Таблица ASCII:

- Коды 0-31: управляющие символы
- Коды 32-126: печатные символы
- '0'-'9': коды 48-57
- 'A'-'Z': коды 65-90
- 'a'-'z': коды 97-122

Работа с символами:

```c
char ch = 'A';
printf("Код символа: %d\n", ch);  // 65

// Преобразование цифры-символа в число
char digit_char = '7';
int digit = digit_char - '0';  // 7 (число)

// Преобразование числа в цифру-символ
int num = 5;
char ch_digit = num + '0';  // '5' (символ)
```

Функции обработки символов (ctype.h):

Проверка типа символа:

```c
#include <ctype.h>

isdigit(ch)   // цифра? ('0'-'9')
isalpha(ch)   // буква? ('a'-'z', 'A'-'Z')
isalnum(ch)   // буква или цифра?
isspace(ch)   // пробельный символ? (пробел, табуляция, \n)
isupper(ch)   // заглавная буква?
islower(ch)   // строчная буква?
ispunct(ch)   // знак пунктуации?
```

Преобразование символов:

```c
```

```
toupper(ch)   // преобразовать в заглавную
tolower(ch)   // преобразовать в строчную
```

Примеры использования:

c

```c
#include <stdio.h>
#include <ctype.h>

int main() {
    char ch = '7';

    if (isdigit(ch)) {
        int digit = ch - '0';
        printf("Это цифра: %d\n", digit);
    }

    ch = 'a';
    if (islower(ch)) {
        ch = toupper(ch);  // 'a' → 'A'
        printf("Заглавная: %c\n", ch);
    }

    // Подсчет типов символов
    char str[] = "Hello123!";
    int letters = 0, digits = 0, others = 0;

    for (int i = 0; str[i] != '\0'; i++) {
        if (isalpha(str[i])) letters++;
        else if (isdigit(str[i])) digits++;
        else others++;
    }

    printf("Букв: %d, Цифр: %d, Других: %d\n", letters, digits, others);

    return 0;
}
```

---

15. Стандартные типы данных (int)

Целочисленные типы:

Тип	Размер	Диапазон
`char`	1 байт	-128 до 127
`unsigned char`	1 байт	0 до 255
`short`	2 байта	-32,768 до 32,767
`unsigned short`	2 байта	0 до 65,535
`int`	4 байта	-2,147,483,648 до 2,147,483,647
`unsigned int`	4 байта	0 до 4,294,967,295
`long`	4/8 байт	зависит от системы
`long long`	8 байт	-9,223,372,036,854,775,808 до 9,223,372,036,854,775,807

Типы с плавающей точкой:

Тип	Размер	Точность
`float`	4 байта	~7 знаков
`double`	8 байт	~15 знаков
`long double`	10-16 байт	~19 знаков

Другие типы:

Тип	Описание
`void`	Отсутствие типа
`_Bool` (или `bool` в C99)	Логический тип (0 или 1)

Примеры использования:

```c
char letter = 'A';
unsigned char byte = 255;
short smallNum = 1000;
int number = 42;
unsigned int positive = 100;
long bigNum = 1000000L;
long long veryBig = 9223372036854775807LL;

float pi = 3.14f;
double precise = 3.14159265359;

// Узнать размер типа
printf("Размер int: %lu байт\n", sizeof(int));
```

Модификаторы:

- `unsigned` - только положительные значения
- `signed` - положительные и отрицательные (по умолчанию)
- `short` - уменьшенный размер
- `long` - увеличенный размер

---

16. Статические переменные (void f() { static int x = 0; })

Особенности статических переменных:

1. Инициализируются **один раз** при первом вызове функции
2. Сохраняют значение между вызовами функции
3. Имеют область видимости функции, но время жизни программы
4. Хранятся в статической памяти (не на стеке)

Примеры:

Счетчик вызовов функции:

```c
```

```c
void countCalls() {
    static int count = 0;  // инициализируется один раз
    count++;
    printf("Функция вызвана %d раз\n", count);
}

int main() {
    countCalls();  // выведет: 1
    countCalls();  // выведет: 2
    countCalls();  // выведет: 3
    return 0;
}
```

Генератор уникальных ID:

```c
int getUniqueID() {
    static int id = 0;
    return ++id;
}

int main() {
    printf("ID: %d\n", getUniqueID());  // 1
    printf("ID: %d\n", getUniqueID());  // 2
    printf("ID: %d\n", getUniqueID());  // 3
    return 0;
}
```

Кэширование результата:

```c
int fibonacci(int n) {
    static int cache[100] = {0};

    if (n <= 1) return n;
    if (cache[n] != 0) return cache[n];  // возврат из кэша

    cache[n] = fibonacci(n-1) + fibonacci(n-2);
    return cache[n];
}
```

Сравнение: обычная vs статическая переменная:

```c
// Обычная локальная переменная
void normalVar() {
    int x = 0;  // инициализируется каждый раз
    x++;
    printf("%d ", x);  // всегда выведет 1
}

// Статическая переменная
void staticVar() {
    static int x = 0;  // инициализируется один раз
    x++;
    printf("%d ", x);  // выведет 1, 2, 3, ...
}

int main() {
    normalVar();  // 1
    normalVar();  // 1
    normalVar();  // 1

    staticVar();  // 1
    staticVar();  // 2
    staticVar();  // 3

    return 0;
}
```

Глобальные статические переменные:

```c
// В файле file1.c
static int privateVar = 10;  // видна только в этом файле

// В файле file2.c
static int privateVar = 20;  // другая переменная, нет конфликта
```

---

17. Функции (int f(int x) {})

Объявление и определение функции:

```c
// Прототип (объявление)
int add(int a, int b);

// Определение
int add(int a, int b) {
    return a + b;
}

// Вызов
int result = add(5, 3
```