Отличный план! Подготовлю вас на 100% к каждому вопросу. Вот полные и структурированные ответы:

📋 1. ФУНКЦИИ РАБОТЫ С ФАЙЛАМИ

Основные группы функций:

1.1 Открытие/закрытие файлов:

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

Режимы открытия:

- `"r"` - чтение (файл должен существовать)
- `"w"` - запись (создает новый или перезаписывает существующий)
- `"a"` - добавление в конец (создает если не существует)
- `"r+"` - чтение и запись (файл должен существовать)
- `"w+"` - чтение и запись (создает новый)
- `"a+"` - чтение и добавление (создает если не существует)

1.2 Построчное чтение/запись:

```
char *fgets(char *str, int n, FILE *stream);
int fputs(const char *str, FILE *stream);
```

- `fgets` читает до n-1 символов или до символа новой строки `\n`
- `fgets` сохраняет `\n` в строку и добавляет `\0`
- `fputs` записывает строку без добавления `\n`

1.3 Посимвольное чтение/запись:

```
int fgetc(FILE *stream);
int fputc(int c, FILE *stream);
int getc(FILE *stream);    // макрос (быстрее)
int putc(int c, FILE *stream); // макрос
```

- Возвращают `EOF` при достижении конца файла или ошибке

1.4 Форматированный ввод/вывод:

```
int fprintf(FILE *stream, const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

- Аналогичны `printf/scanf` но работают с файлами

1.5 Блочное чтение/запись:

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

- Для работы с бинарными файлами
- `size` - размер одного элемента
- `count` - количество элементов

1.6 Навигация по файлу:

```
int fseek(FILE *stream, long offset, int origin);
long ftell(FILE *stream);
void rewind(FILE *stream);
```

- `origin` : `SEEK_SET` (начало), `SEEK_CUR` (текущая), `SEEK_END` (конец)
- `rewind(stream)` = `fseek(stream, 0, SEEK_SET)`

1.7 Проверка состояния:

```
int feof(FILE *stream);    // конец файла?
int ferror(FILE *stream);  // ошибка?
void clearerr(FILE *stream); // сброс ошибок
```

🔧 2. ДИРЕКТИВЫ ПРЕПРОЦЕССОРА

2.1 #include - включение файлов

```
#include <stdio.h>     // Из системных каталогов
#include "myfile.h"    // Из текущего каталога
```

Как работает:

- Препроцессор находит файл и вставляет его содержимое
- Угловые скобки - поиск в системных путях
- Кавычки - поиск в текущей директории, потом в системных

2.2 #define - макроподстановка

Простая замена:

```
#define PI 3.14159
#define MAX_SIZE 100
```

- Препроцессор заменяет `PI` на `3.14159` во всем коде

Макросы с параметрами:

```
#define SQUARE(x) ((x) * (x))
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

Важно использовать скобки! Без скобок:

```
#define SQUARE(x) x * x
SQUARE(2 + 3) → 2 + 3 * 2 + 3 = 11 (а не 25!)
```

Многострочные макросы:

```
#define PRINT_ERROR(msg) \
    printf("Ошибка: %s\n", msg); \
    printf("Файл: %s, Строка: %d\n", __FILE__, __LINE__)
```

Предопределенные макросы:

```
__LINE__      // текущий номер строки
__FILE__      // имя текущего файла
__DATE__      // дата компиляции
__TIME__      // время компиляции
__STDC__      // соответствует стандарту?
```

2.3 Другие важные директивы:

```
#ifdef DEBUG          // если DEBUG определен
    // отладочный код
#endif

#ifndef HEADER_H      // если HEADER_H не определен
#define HEADER_H      // определяем чтобы не включать повторно
    // содержимое заголовочного файла
#endif

#pragma once          // современная альтернатива include guard
```

🔤 3. КОДИРОВКА СИМВОЛОВ В C

3.1 Основные понятия:

- **ASCII** - американская кодировка (0-127)
- **Расширенная ASCII** - 0-255 (зависит от локали)
- **UTF-8** - современная многобайтовая кодировка
- **Локаль** - настройки региона (язык, кодировка)

3.2 Символьные типы:

```
char c = 'A';          // обычный символ (1 байт)
wchar_t wc = L'Я';     // широкий символ (2-4 байта)
```

3.3 Функции для работы с локалями:

```c
#include <locale.h>

// Установка локали
char *setlocale(int category, const char *locale);

// Категории:
LC_ALL      // всё
LC_CTYPE    // классификация символов
LC_COLLATE  // сравнение строк
LC_NUMERIC  // формат чисел

// Примеры:
setlocale(LC_ALL, "");        // локаль по умолчанию системы
setlocale(LC_ALL, "ru_RU.UTF-8");  // русская UTF-8
setlocale(LC_ALL, "C");       // стандартная локаль
```

3.4 Работа с кодировками в Windows:

```c
#include <windows.h>

// Смена кодовой страницы консоли
system("chcp 65001");     // UTF-8
system("chcp 1251");      // Windows-1251 (кириллица)

// Или через WinAPI:
SetConsoleCP(1251);       // установка входной кодовой страницы
SetConsoleOutputCP(1251); // установка выходной кодовой страницы
```

3.5 Преобразование символов:

```c
#include <ctype.h>

int tolower(int c);      // в нижний регистр
int toupper(int c);      // в верхний регистр
int isalpha(int c);      // буква?
int isdigit(int c);      // цифра?
int isalnum(int c);      // буква или цифра?
```

🧮 4. МАССИВЫ В C

4.1 Объявление массивов:

```c
// Одномерные массивы:
int numbers[10];                    // 10 целых чисел
float temperatures[24];             // 24 дробных числа
char name[50];                      // 49 символов + '\0'

// Многомерные массивы:
int matrix[3][4];                   // 3 строки, 4 столбца
char strings[5][100];               // 5 строк по 99 символов каждая
```

4.2 Инициализация массивов:

```c
// Полная инициализация:
int primes[] = {2, 3, 5, 7, 11};    // размер вычисляется автоматически

// Частичная инициализация:
int arr[10] = {1, 2, 3};            // остальные элементы = 0

// Инициализация нулями:
int zeros[100] = {0};               // все элементы = 0

// Строковые массивы:
char greeting[] = "Hello";          // автоматически добавляет '\0'
char name[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; // эквивалентно
```

4.3 Обращение к элементам:

```c
int arr[5];
arr[0] = 10;           // первый элемент (индекс 0)
arr[4] = 50;           // последний элемент
int x = arr[2];        // чтение значения

// Многомерные массивы:
int matrix[2][3];
matrix[0][1] = 5;      // строка 0, столбец 1
```

4.4 Передача массивов в функции:

```c
// Способ 1: как указатель (рекомендуется)
void process_array(int *arr, int size) {
    for(int i = 0; i < size; i++) {
        arr[i] *= 2;
    }
}

// Способ 2: как массив с размером
void process_array(int arr[], int size) {
    // тот же код
}

// Способ 3: многомерный массив с фиксированным размером
void process_matrix(int matrix[][4], int rows) {
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < 4; j++) {
            matrix[i][j] = i + j;
        }
    }
}

// Вызов:
int my_array[10];
process_array(my_array, 10);
```

4.5 Особенности массивов в C:

Указатель на первый элемент:

```
int arr[5];
// arr эквивалентно &arr[0]
// arr + i эквивалентно &arr[i]
// *arr эквивалентно arr[0]
```

Размер массива:

```
int arr[10];
int size = sizeof(arr) / sizeof(arr[0]);  // правильный способ
```

Строки как массивы символов:

```
char str[20] = "Hello";
str[0] = 'h';                    // изменяем первый символ
int len = strlen(str);           // длина строки (без '\0')
```

Массивы и указатели:

```
int arr[5];
int *ptr = arr;                  // ptr указывает на arr[0]
ptr[2] = 10;                     // эквивалентно arr[2] = 10
*(ptr + 3) = 20;                 // эквивалентно arr[3] = 20
```

4.6 Распространенные ошибки:

```
// Выход за границы массива:
int arr[5];
arr[5] = 10;  // ОШИБКА! Индексы 0-4

// Путаница с размерами:
void wrong(int arr[]) {
    int size = sizeof(arr);  // РАЗМЕР УКАЗАТЕЛЯ, не массива!
}
```

🎯 КЛЮЧЕВЫЕ МОМЕНТЫ ДЛЯ ЗАПОМИНАНИЯ:

1. **Файлы:** Всегда проверяйте результат `fopen()` и закрывайте файлы `fclose()`
2. **Препроцессор:** `#include` вставляет код, `#define` делает текстовую замену
3. **Кодировки:** Используйте `setlocale()` для русского текста
4. **Массивы:** Индексы с 0, имя массива - указатель на первый элемент

Теперь вы готовы на 100% к любым вопросам! Удачи на защите! 🚀