perplexity

This is for module 4, read each pdf (name corresponds to the module number), read through these pdfs and compile a list of all programming questions module wise and make a table of it module wise. And after list of programs, include code for complex programs and question and answers for theory questions, Avoid repetition of questions, and if one follow up question seems redundant because we have discussed same topic before in an older question, ignore that.

Here's a structured breakdown of **Module 4: Pointers, Structures, and File Handling** programming questions and theory Q&A, compiled from the provided PDFs:

Module 4: Programming Questions Table

Q.No	Program Description	Key Concepts Tested	Complexity Level
1	Compare two integer arrays for equality using pointers	Pointer arithmetic, array traversal	Intermediate
2	Swap values using pointers (call by reference)	Pointer parameters, dereferencing	Beginner
3	Reverse a string using pointers	Pointer arithmetic, string manipulation	Intermediate
4	Add two matrices using pointers	2D array handling, pointer arithmetic	Advanced
5	Print array elements in reverse order using pointers	Pointer decrement, array traversal	Intermediate
6	Compute factorial and write result to a file	File I/O (`fprintf`, `fclose`)	Intermediate
7	Copy contents of one file to another	File handling (`fgetc`, `fputc`)	Intermediate
8	Replace vowels with 'x' in a text file	File handling (`fseek`, `fputc`)	Advanced
9	Count lines in a text file	File handling (`fgetc`, EOF handling)	Intermediate
10	Dynamically allocate memory for an integer array	`malloc`, `free`	Intermediate

Q.No	Program Description	Key Concepts Tested	Complexity Level
11	Implement matrix transpose using pointers	2D array manipulation, pointer arithmetic	Advanced
12	Manage inventory data (structs) and write/read from file	Structures, file I/O (`fprintf`, `fscanf`)	Advanced

Code for Complex Programs

1. Matrix Transpose (Q11)

```
#include <stdio.h>
#define SIZE 3

void transpose(int mat[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = i + 1; j < SIZE; j++) {
            int temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
        }
    }
}

int main() {
    int mat[SIZE][SIZE] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    transpose(mat);
    printf("Transposed matrix:\n");
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
    return 0;
}
```

Key Features:

- Uses nested loops and pointer-like indexing for matrix manipulation.
- Swaps elements across the diagonal.

2. Dynamic Memory Allocation (Q10)

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *arr, n;
```

```
    printf("Enter number of elements: ");
    scanf("%d", &n);
    arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }
    printf("Enter elements: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    printf("Array elements: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    free(arr);
    return 0;
}
```

Key Features:

- Uses `malloc` for dynamic allocation.
- Includes error handling for failed allocation.

3. File Copy (Q7)

```
#include <stdio.h>

int main() {
    FILE *src = fopen("source.txt", "r");
    FILE *dest = fopen("dest.txt", "w");
    char ch;
    while ((ch = fgetc(src)) != EOF)
        fputc(ch, dest);
    fclose(src);
    fclose(dest);
    printf("File copied successfully.\n");
    return 0;
}
```

Key Features:

- Uses `fgetc` and `fputc` for byte-by-byte copying.
- Handles EOF correctly.

Theory Q&A

1. Difference between `malloc()` and `calloc()`

Feature	`malloc()`	`calloc()`
Initialization	Uninitialized (garbage values)	Zero-initialized
Syntax	`malloc(size)`	`calloc(num, size)`
Use Case	Single-block allocation	Array allocation

2. Pointer vs Array

Feature	Array	Pointer
Memory	Static allocation	Dynamic or static allocation
Size	Fixed at declaration	Can be resized (`realloc`)
Assignment	Cannot be reassigned	Can point to different addresses

3. `fseek()` and `ftell()`

- `fseek(FILE *ptr, offset, whence)`: Moves the file pointer to a specific position.
  - `whence`: `SEEK_SET` (start), `SEEK_CUR` (current), `SEEK_END` (end).
- `ftell(FILE *ptr)`: Returns the current position of the file pointer.

4. Structure vs Union

Feature	Structure	Union
Memory	Allocates space for all members	Shares memory for members
Access	All members accessible	Only one member at a time
Size	Sum of member sizes	Size of largest member

5. File Modes

Mode	Description
`"r"`	Read mode (file must exist)
`"w"`	Write mode (creates/overwrites file)
`"a"`	Append mode (writes at end of file)
`"r+"`	Read/write mode (file must exist)
`"w+"`	Read/write mode (creates/overwrites file)

Key Notes

- **Pointer Arithmetic**: Incrementing a pointer moves it by the size of its data type (e.g., `int *ptr++` moves by 4 bytes).
- **Dynamic Memory**: Always check for `NULL` after `malloc/calloc` and use `free()` to prevent leaks.
- **File Handling**: Use `fclose()` after operations to ensure data integrity.

Let me know if you need further clarification or additional programs!

⁂