Dr. SNS RAJALAKSHMI COLLEGE OF ARTS AND SCIENCE

(AUTONOMOUS)

Accredited by NAAC (Cycle- III) with 'A+' Grade

ESTD : 1999

DEPARTMENT OF B.SC CS (GCD)

23UCU401 – PROGRAMMING IN C

UNIT- IV

Subject : Programming in C
Department : GCD
Faculty : Mrs.K.Sangeetha

Unit 4 :

Features of C Preprocessor - Macro Expansion - File Inclusion - Conditional Compilation - #if and #elif Directives. Arrays: What are Arrays - More on Arrays - Pointers and Arrays - Two Dimensional Arrays - Array of Pointers - Three Dimensional Array.

1. Features of C Preprocessor

- The C preprocessor is a tool that processes your source code before compilation. It's used for tasks like including header files, macro expansion, and conditional compilation.

Some features of the C preprocessor include:

- Macro expansion: Allows you to define and use macros in your code.
- File inclusion: Provides the ability to include other source code files.
- Conditional compilation: Allows you to compile or exclude code based on conditions.
- Header files: Used to include declarations and function prototypes.

2. Macro Expansion

- Macros are a way to create symbolic constants or short functions in C. They are expanded during preprocessing.

Example:

```
#define PI 3.14159265
#define SQUARE(x) (x * x)

float area = PI * SQUARE(2);
```

Explanation:

- The `#define` directive creates macros. In the example, `PI` is a constant, and `SQUARE(x)` is a simple function-like macro.
- During preprocessing, `PI` is replaced with `3.14159265`, and `SQUARE(2)` becomes `(2 * 2)`, resulting in the calculation of the `area`.

3. File Inclusion

- File inclusion is a way to add external code files to your program using `#include` directives.

Example:

```
#include <stdio.h>
#include "myheader.h"

int main() {
    printf("Hello, world!\n");
    myFunction();
    return 0;
}
```

Explanation:

- `#include <stdio.h>` includes the standard input/output library.
- `#include "myheader.h"` includes a custom header file called `myheader.h`, which contains declarations and functions used in the program.

4. Conditional Compilation

- Conditional compilation allows you to include or exclude parts of your code based on defined conditions using `#if`, `#ifdef`, and `#ifndef` directives.

Example:

```
#define DEBUG
#ifdef DEBUG
    printf("Debug mode is
enabled.\n");
#endif
```

Explanation:

- If the `DEBUG` macro is defined, the code within the `#ifdef DEBUG` block is included in the compilation.

5. #if and #elif Directives

- The `#if` and `#elif` directives are used for conditional compilation with numerical expressions.

Example:

```
#define NUM 42
#if NUM > 50
    printf("NUM is greater than 50.\n");
#elif NUM < 50
    printf("NUM is less than 50.\n");
#else
    printf("NUM is equal to 50.\n");
#endif
```

Explanation:

- The `#if` directive allows you to evaluate numerical expressions for conditional compilation.

6. Arrays: What are Arrays

- Arrays are a fundamental data structure in C that allow you to store multiple values of the same data type in a single variable.

Example:

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

Explanation:

- `numbers` is an array that can store 5 integers.
- Array elements can be accessed using indexes, e.g., `numbers[0]` is the first element.

7. More on Arrays

- Arrays are indexed starting from 0.
- You can use loops to iterate through array elements.

Example:

```
for (int i = 0; i < 5; i++) {
    printf("Element %d: %d\n", i, numbers[i]);
}
```

Explanation:

- This loop iterates through the `numbers` array and prints each element along with its index.

8. Pointers and Arrays

- Pointers and arrays are closely related. An array name can be thought of as a constant pointer to the first element of the array.

Example:

```
int arr[3] = {10, 20, 30};
int* ptr = arr; // 'ptr' points to the first element of 'arr'
```

Explanation:

- `ptr` can be used to access elements of `arr`. `*ptr` is equivalent to `arr[0]`.

9. Two Dimensional Arrays

- Two-dimensional arrays allow you to represent tables or matrices with rows and columns.

Example:

```
int matrix[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

Explanation:

- `matrix` is a 3x3 two-dimensional array, and elements can be accessed using two indices, e.g., `matrix[0][1]` is 2.

In this example, we'll create a simple 3x3 matrix and perform some basic operations on it:

```
#include <stdio.h>

int main() {
    int matrix[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    // Display the matrix
    printf("Matrix:\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
```

```
// Calculate the sum of all elements
    int sum = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            sum += matrix[i][j];
        }
    }

    printf("Sum of all elements: %d\n", sum);

    return 0;
}
```

Explanation:

- We declare a 3x3 matrix named `matrix` and initialize it with values.
- We use nested loops to display the matrix and calculate the sum of its elements.
- The loops iterate through the rows and columns of the matrix to access and process each element.

10. Array of Pointers

- An array of pointers allows you to create arrays of data where each element is a pointer to data of a different type.

Example:

```
int* arr[3]; // An array of integer pointers
```

Explanation:

- `arr` is an array of integer pointers, and each element can point to an integer value.

11. Three Dimensional Array

- Three-dimensional arrays are used to represent data structures with three levels of indexing.

Example:

```
int cube[2][3][4];
```

Explanation:

- `cube` is a three-dimensional array with dimensions 2x3x4.
- Accessing elements requires three indices, e.g., `cube[1][2][3]` accesses a specific element.