SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107

AN AUTONOMOUS INSTITUTION

Approved by AICTE, New Delhi and Affiliated to Anna University, Chennai

PROBLEM SOLVNG & C PROGRAMMNG

Puzzles

1. The Fibonacci Sequence

C

```c
#include <stdio.h>

int main() {
    int n, first = 0, second = 1, next, c;

    printf("Enter the number of terms: ");
    scanf("%d", &n);

    printf("First %d terms of Fibonacci series:\n", n);

    for (c = 0; c < n; c++) {
        if (c <= 1)
            next = c;
        else {
            next = first + second;
            first = second;
            second = next;
        }
        printf("%d ", next);
    }

    return 0;
}
```

Explanation:

- This code generates the Fibonacci sequence, where each number is the sum of the two preceding ones (e.g., 0, 1, 1, 2, 3, 5, 8, ...).
- The user inputs the desired number of terms.
- The code iterates through the loop, calculating and printing each term of the sequence.

2. Factorial of a Number

C

```c
#include <stdio.h>

int main() {int n, i, factorial = 1;
```

```
    printf("Enter an integer: ");
    scanf("%d", &n);

    // Show error if the user enters a negative integer
    if (n < 0)
        printf("Error! Factorial of a negative number doesn't exist.");

    else {
        for (i = 1; i <= n; ++i) {
            factorial *= i;
        }
        printf("Factorial of %d = %llu", n, factorial);
    }

    return 0;
}
```

Explanation:

- This code calculates the factorial of a given number.
- The factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or[1] equal to n.
- The code includes a check for negative input and calculates the factorial using a loop.

3. Prime Number Check

C

```
    int n, i, isPrime = 1;

    printf("Enter an integer: ");
    scanf("%d", &n);

    // 0 and 1 are not prime numbers
    if (n <= 1) {
        isPrime = 0;
    } else {
        for (i = 2; i <= n / 2; ++i) {
            if (n % i == 0) {
                isPrime = 0;
                break;
            }
        }
    }

    if (isPrime)
        printf("%d is a prime number.", n);
    else
        printf("%d is not a prime number.", n);
```

```
    return 0;
}
```
Explanation:

- This code checks if a given number is a prime number (a natural number greater than 1 that has no positive divisors other than 1 and itself).
- It iterates from 2 to half of the given number, checking if any number divides evenly.
- If any divisor is found, the number is not prime.

4. String Reversal

C

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

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%s", str);

    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = temp;
    }

    printf("Reversed string: %s\n", str);

    return 0;
}
```

Explanation:

- This code reverses a given string.
- It iterates through half of the string's length, swapping characters from the beginning and int data[100], n, i, j, swap;

  printf("Enter number of elements: "); scanf("%d", &n);

  printf("Enter elements: "); for (i = 0; i < n; i++) { scanf("%d", &data[i]); }
  for (i = 0; i < n - 1; i++) { for (j = 0; j < n - i - 1; j++) { if (data[j] > data[j + 1]) { swap = data[j]; data[j] = data[j + 1]; data[j + 1] = swap; } } }
  printf("Sorted[2] Array: "); for (i = 0; i < n; i++) { printf("%d ", data[i]); }

  return 0; }

Explanation:

* This code implements the bubble sort algorithm, which repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.
* This process continues until no swaps are needed in an entire pass.

Certainly, let's explore some C pointer puzzles!

1. The Double Pointer Puzzle

```c
#include <stdio.h>

int main() {
    int x = 10;
    int *ptr1 = &x;
    int **ptr2 = &ptr1;

    printf("Value of x: %d\n", x);
    printf("Value of *ptr1: %d\n", *ptr1);
    printf("Value of **ptr2: %d\n", **ptr2);

    **ptr2 = 20; // What happens here?**

    printf("Value of x after modification: %d\n", x);

    return 0;
}
```

Explanation:

* This code demonstrates the concept of double pointers.

* `ptr1` is a pointer to the integer `x`.

* `ptr2` is a pointer to the pointer `ptr1`.

* The line `**ptr2 = 20;` **does not** change the value of `x`. It actually assigns the integer value 20 to the memory location pointed to by `ptr1`.

2. The Array of Pointers Puzzle

```c
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30, 40};
    int *ptr[4];

    for (int i = 0; i < 4; i++) {
        ptr[i] = &arr[i];
    }

    for (int i = 0; i < 4; i++) {
        printf("Value at ptr[%d]: %d\n", i, *ptr[i]);
    }

    return 0;
}
```

Explanation:

* This code creates an array of pointers (`ptr`).

* Each element of the `ptr` array is assigned the address of the corresponding element in the `arr` array.

* The code then iterates through the `ptr` array and prints the values pointed to by each pointer.

3. The Function Pointer Puzzle

```c
#include <stdio.h>

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

int subtract(int a, int b) {
    return a - b;
}

int main() {
    int (*operation)(int, int); // Function pointer declaration

    operation = add;
    printf("Result of addition: %d\n", operation(5, 3));

    operation = subtract;
    printf("Result of subtraction: %d\n", operation(5, 3));

    return 0;
}
```

Explanation:

* This code demonstrates the use of function pointers.
* `operation` is a pointer that can hold the address of a function that takes two integers as arguments and returns an integer.
* The code assigns the addresses of the `add` and `subtract` functions to the `operation` pointer and then calls the functions through the pointer.

Key Pointer Concepts:

* **Pointers store memory addresses.**
* **Pointers can be used to access and modify values indirectly.**
* **Pointers can point to any data type, including other pointers.**
* **Function pointers allow you to pass functions as arguments to other functions.**