**1.Write a C program to reverse a given string without using any additional library functions.**

#include <stdio.h>

#include <string.h>

int main() {

char str[] = "Kavya";

char res[10];

int len = strlen(str);

int j = 0;

for (int i = len - 1; i >= 0; --i) {

res[j] = str[i];

j++;

}

res[j] = '\0';

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

return 0;

}

**2.Explain the concept of structures in C and write a program to store student information (name, roll number, marks) using a structure.**

**In C, a structure is a user-defined data type that allows the combination of data items of different kinds. Structures are used to group together different types of variables under a single name. Each variable within a structure is called a member.**

#include <stdio.h>

int main() {

struct Student {

char name[20];

int rollNumber;

float marks;

};

struct Student student;

printf("Enter student name: ");

scanf(“%s”,student.name);

printf("Enter student roll number: ");

scanf("%d", &student.rollNumber);

printf("Enter student marks: ");

scanf("%f", &student.marks);

printf("\nStudent Information:\n");

printf("Name: %s\n", student.name);

printf("Roll Number: %d\n", student.rollNumber);

printf("Marks: %.2f\n", student.marks);

return 0;

}

**3.Differentiate between single-linked lists and doubly-linked lists in C. Write code snippets to create a node and perform a basic insertion operation in a singly-linked list.**

|  |  |
| --- | --- |
| **Single-Linked List** | **Double-Linked List** |
| Each node has data and a pointer to the next node. | Each node has data, a pointer to the next node, and a pointer to the previous node. |
| Traversal is possible in one direction only. | Traversal is possible in both directions (forward and backward). |
| Single linked list for the execution of stacks. | Double linked list to execute heaps and binary trees. |
| **A single list consumes less memory as compared to the double linked list.** | **The double linked list consumes more memory as compared to the single linked list** |
| **Complexity of insertion and deletion O(n).** | **Complexity of insertion and deletion is O(1).** |

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

printf("Memory allocation error\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

**4.Explain the concept of pointers in C and write a program to swap the values of two variables using pointers.**

A pointer is a variable that stores the memory address of another variable. Pointers provide a powerful and flexible way to manipulate data and memory, allowing for dynamic memory allocation, efficient array handling, and the creation of complex data structures like linked lists and trees.

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x, y;

printf("Enter value for x: ");

scanf("%d", &x);

printf("Enter value for y: ");

scanf("%d", &y);

printf("Before swapping: x = %d, y = %d\n", x, y);

swap(&x, &y);

printf("After swapping: x = %d, y = %d\n", x, y);

return 0;

}