DSA 2003

Use C/C++

1. Implement a program to reverse the given string using Stack.
2. Implement a stack using linked list to keep student's details such as name, reg.no., age, etc.
3. Implement a program to convert the given infix expression in to postfix expression.
4. Implement a program to evaluate infix expressions.
5. Implement a data structure to keep student's details in the same order of their admission, with provision to add and remove in the same order of their entry in the list.
6. Implement a data structure to keep sales details in the order of their entry. The number of entries is unknown and dynamically changing.
7. Implement Josephus problem using circular linked list, to identify and print the id of the winner. User can get the number of players and id of starter.
8. Implement Tower of Hanoi problem using recursion. User can give the number of disks, print the each steps of disk movements.
9. Implement polynomial addition, use linked list to represent polynomials.

Sample Implementation Codes

Tower of Hanoi

```
/*
 * C program for Tower of Hanoi using Recursion
 */
#include <stdio.h>

void towers(int, char, char, char);

int main()
{
    int num;

    printf("Enter the number of disks : ");
    scanf("%d", &num);
    printf("The sequence of moves involved in the Tower of Hanoi are :\n");
    towers(num, 'A', 'C', 'B');
    return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
    if (num == 1)
    {
        printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
        return;
    }
    towers(num - 1, frompeg, auxpeg, topeg);
    printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
    towers(num - 1, auxpeg, topeg, frompeg);
}
```

Polynomial Addition

```
// C++ program for addition of two polynomials
// using Linked Lists
#include <bits/stdc++.h>
using namespace std;

// Node structure containing power and coefficient of
// variable
struct Node {
    int coeff;
    int pow;
    struct Node* next;
};

// Function to create new node
void create_node(int x, int y, struct Node** temp)
{
    struct Node *r, *z;
    z = *temp;
    if (z == NULL) {
            r = (struct Node*)malloc(sizeof(struct Node));
            r->coeff = x;
            r->pow = y;
            *temp = r;
            r->next = (struct Node*)malloc(sizeof(struct Node));
            r = r->next;
            r->next = NULL;
    }
    else {
            r->coeff = x;
            r->pow = y;
            r->next = (struct Node*)malloc(sizeof(struct Node));
            r = r->next;
            r->next = NULL;
    }
}

// Function Adding two polynomial numbers
void polyadd(struct Node* poly1, struct Node* poly2,
                    struct Node* poly)
{
    while (poly1->next && poly2->next) {
            // If power of 1st polynomial is greater then 2nd,
            // then store 1st as it is and move its pointer
            if (poly1->pow > poly2->pow) {
                    poly->pow = poly1->pow;
                    poly->coeff = poly1->coeff;
                    poly1 = poly1->next;
            }

            // If power of 2nd polynomial is greater then 1st,
            // then store 2nd as it is and move its pointer
            else if (poly1->pow < poly2->pow) {
                    poly->pow = poly2->pow;
                    poly->coeff = poly2->coeff;
                    poly2 = poly2->next;
            }

            // If power of both polynomial numbers is same then
            // add their coefficients
            else {
                    poly->pow = poly1->pow;
                    poly->coeff = poly1->coeff + poly2->coeff;
                    poly1 = poly1->next;
                    poly2 = poly2->next;
            }
```

```
        // Dynamically create new node
        poly->next
            = (struct Node*)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
    while (poly1->next ‖ poly2->next) {
        if (poly1->next) {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        if (poly2->next) {
            poly->pow = poly2->pow;
            poly->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        poly->next
            = (struct Node*)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
}

// Display Linked list
void show(struct Node* node)
{
    while (node->next != NULL) {
        printf("%dx^%d", node->coeff, node->pow);
        node = node->next;
        if (node->coeff >= 0) {
            if (node->next != NULL)
                printf("+");
        }
    }
}
```

```
// Driver code
int main()
{
    struct Node *poly1 = NULL, *poly2 = NULL, *poly = NULL;

    // Create first list of 5x^2 + 4x^1 + 2x^0
    create_node(5, 2, &poly1);
    create_node(-4, 1, &poly1);
    create_node(2, 0, &poly1);

    // Create second list of -5x^1 - 5x^0
    create_node(-5, 1, &poly2);
    create_node(5, 0, &poly2);

    printf("1st Number: ");
    show(poly1);

    printf("\n2nd Number: ");
    show(poly2);

    poly = (struct Node*)malloc(sizeof(struct Node));

    // Function add two polynomial numbers
    polyadd(poly1, poly2, poly);

    // Display resultant List
    printf("\nAdded polynomial: ");
    show(poly);

    return 0;
}
```

Josephus Problem

```
#include <stdio.h>

int josephus(int n, int k)
{
  if (n == 1)
    return 1;
  else
    /* The position returned by josephus(n - 1, k) is adjusted because the
       recursive call josephus(n - 1, k) considers the original position
       k%n + 1 as position 1 */
    return (josephus(n - 1, k) + k-1) % n + 1;
}

// Driver Program to test above function
int main()
{
  int n = 14;
  int k = 2;
  printf("The chosen place is %d", josephus(n, k));
  return 0;
}
```