Rajalakshmi Engineering College

Name: magesh v
Email: 240801187@rajalakshmi.edu.in
Roll no: 240801187
Phone: 6381572897
Branch: REC
Department: I ECE FB
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

In a coding competition, you are assigned a task to create a program that simulates a stack using a linked list.

The program should feature a menu-driven interface for pushing an integer to stack, popping, and displaying stack elements, with robust error handling for stack underflow situations. This challenge tests your data structure skills.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the integer value onto the stack. If the choice is 1, the following input is a space-separated integer, representing the element to be pushed onto the stack.

Choice 2: Pop the integer from the stack.

Choice 3: Display the elements in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

If the choice is 1, push the given integer to the stack and display the following: "Pushed element: " followed by the value pushed.

If the choice is 2, pop the integer from the stack and display the following: "Popped element: " followed by the value popped.

If the choice is 2, and if the stack is empty without any elements, print "Stack is empty. Cannot pop."

If the choice is 3, print the elements in the stack: "Stack elements (top to bottom): " followed by the space-separated values.

If the choice is 3, and there are no elements in the stack, print "Stack is empty".

If the choice is 4, exit the program and display the following: "Exiting program".

If any other choice is entered, print "Invalid choice".

Refer to the sample input and output for the exact format.

Sample Test Case

```
Input: 1 3
1 4
3
2
3
4
Output: Pushed element: 3
Pushed element: 4
Stack elements (top to bottom): 4 3
Popped element: 4
Stack elements (top to bottom): 3
Exiting program
```

Answer

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

struct Node {
   int data;
   struct Node* next;
};

struct Node* top = NULL;

// You are using GCC
// Function to push an element onto the stack
void push(int value) {
   // Create a new node
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (newNode == NULL) {
      printf("Memory allocation failed. Cannot push element.\n");
      return;
   }
```

```
    // Assign data to the new node
    newNode->data = value;

    // Link the new node to the current top
    newNode->next = top;

    // Update the top of the stack
    top = newNode;

    // Print the push confirmation message
    printf("Pushed element: %d\n", value);
}

// Function to pop an element from the stack
void pop() {
    // Check if the stack is empty (underflow condition)
    if (top == NULL) {
        printf("Stack is Empty. Cannot pop.\n");
        return;
    }

    // Store the top node to free it later
    struct Node* temp = top;

    // Get the data before popping
    int poppedValue = temp->data;

    // Update the top to the next node
    top = top->next;

    // Free the memory of the popped node
    free(temp);

    // Print the pop confirmation message
    printf("Popped element: %d\n", poppedValue);
}

// Function to display the elements of the stack
void displayStack() {
    // Check if the stack is empty
    if (top == NULL) {
        printf("Stack is empty\n");
        return;
    }

    // Traverse the stack from top to bottom and print elements
    struct Node* current = top;
    printf("Stack elements (top to bottom): ");
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    int choice, value;
    do {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                printf("Exiting program\n");
                return 0;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 4);

    return 0;
}
```

Status : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: magesh v
Email: 240801187@rajalakshmi.edu.in
Roll no: 240801187
Phone: 6381572897
Branch: REC
Department: I ECE FB
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Sanjeev is in charge of managing a library's book storage, and he wants to create a program that simplifies this task. His goal is to implement a program that simulates a stack using an array.

Help him in writing a program that provides the following functionality:

Add Book ID to the Stack (Push): You can add a book ID to the top of the book stack. Remove Book ID from the Stack (Pop): You can remove the top book ID from the stack and display its details. If the stack is empty, you cannot remove any more book IDs.Display Books ID in the Stack (Display): You can view the books ID currently on the stack.Exit the Library: You can choose to exit the program.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the book onto the stack. If the choice is 1, the following input is a space-separated integer, representing the ID of the book to be pushed onto the stack.

Choice 2: Pop the book ID from the stack.

Choice 3: Display the book ID in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

1. If the choice is 1, push the given book ID to the stack and display the corresponding message.
2. If the choice is 2, pop the book ID from the stack and display the corresponding message.
3. If the choice is 2, and if the stack is empty without any book ID, print "Stack Underflow"
4. If the choice is 3, print the book IDs in the stack.
5. If the choice is 3, and there are book IDs in the stack, print "Stack is empty"
6. If the choice is 4, exit the program and display the corresponding message.
7. If any other choice is entered, print "Invalid choice"

Refer to the sample output for the exact text and format.

Sample Test Case

Input: 1 19
1 28
2
3
2
4
Output: Book ID 19 is pushed onto the stack
Book ID 28 is pushed onto the stack

```
Book ID 28 is popped from the stack
Book ID in the stack: 19
Book ID 19 is popped from the stack
Exiting the program
```

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

// Define the structure for a stack node (representing a book ID)
struct Node {
   int data;
   struct Node* next;
};

// Global variable for the top of the stack
struct Node* top = NULL;

// Function to push a book ID onto the stack
void push(int value) {
   // Create a new node
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (newNode == NULL) {
      // Handle memory allocation failure (overflow in practical terms for this
scenario)
      // The sample output doesn't show an overflow case, so we'll just print a
message.
      printf("Memory allocation failed. Cannot push element.\n");
      return;
   }

   // Assign book ID to the new node
   newNode->data = value;

   // Link the new node to the current top
   newNode->next = top;

   // Update the top of the stack
   top = newNode;

   // Print the push confirmation message as per the sample output format
```

```
    printf("Book ID %d is pushed onto the stack\n", value);
}

// Function to pop a book ID from the stack
void pop() {
    // Check if the stack is empty (underflow condition)
    if (top == NULL) {
        printf("Stack Underflow\n"); // Output format for empty pop (matches
sample)
        return;
    }

    // Store the top node to free it later
    struct Node* temp = top;

    // Get the book ID before popping
    int poppedValue = temp->data;

    // Update the top to the next node
    top = top->next;

    // Free the memory of the popped node
    free(temp);

    // Print the pop confirmation message as per the sample output format
    printf("Book ID %d is popped from the stack\n", poppedValue);
}

// Function to display the book IDs in the stack
void displayStack() {
    // Check if the stack is empty
    if (top == NULL) {
        printf("Stack is empty\n"); // Output format for empty display (matches
sample)
        return;
    }

    // Traverse the stack from top to bottom and print elements
    struct Node* current = top;
    printf("Book ID in the stack: "); // Output format for non-empty display
(matches sample)
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n"); // Newline after displaying elements
}

int main() {
    int choice;
    int bookID;

    // Loop to keep the program running until the user exits
    while (1) {
        // No menu printing here to match the sample input/output format which
doesn't show prompts
        // You would typically print a menu in a real interactive program.

        // Read the user's choice
        if (scanf("%d", &choice) != 1) {
             // Handle potential non-integer input gracefully, though not strictly
required by sample
             printf("Invalid input. Please enter a number.\n");
             while (getchar() != '\n'); // Consume the invalid input
             continue; // Go to the next loop iteration
        }


        // Process the user's choice
        switch (choice) {
            case 1:
                // For choice 1, we also need to read the book ID
                if (scanf("%d", &bookID) != 1) {
                    printf("Invalid input for Book ID.\n");
                    while (getchar() != '\n'); // Consume the invalid input
                    // Decide whether to continue or break based on desired error
handling
                    continue; // Skip the push operation for this invalid input
                }
                push(bookID);
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                printf("Exiting the program\n"); // Output format for exiting (matches
sample)
                // Before exiting, it's good practice to free all allocated memory
                while (top != NULL) {
                    struct Node *temp = top;
                    top = top->next;
                    free(temp);
                }
                return 0; // Exit the program
            default:
                printf("Invalid choice\n"); // Output format for invalid choice (matches
sample)
                // Consume the rest of the line to prevent issues with future inputs
                // Note: For single number invalid choice like '9' followed by '4', this is
sufficient.
                // If the invalid input was '9 abc', the initial scanf might leave ' abc' in the
buffer.
                // For competitive programming like scenarios where input format is
guaranteed, simpler handling might be enough.
                // Let's assume the sample input format implies valid integers for
choices and book IDs when expected.
                break; // Just print invalid choice and wait for the next input
        }
    }

    return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: magesh v
Email: 240801187@rajalakshmi.edu.in
Roll no: 240801187
Phone: 6381572897
Branch: REC
Department: I ECE FB
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Sharon is developing a programming challenge for a coding competition. The challenge revolves around implementing a character-based stack data structure using an array.

Sharon's project involves a stack that can perform the following operations:

Push a Character: Users can push a character onto the stack.Pop a Character: Users can pop a character from the stack, removing and displaying the top character.Display Stack: Users can view the current elements in the stack.Exit: Users can exit the stack operations application.

Write a program to help Sharon to implement a program that performs the given operations.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the character onto the stack. If the choice is 1, the following input is a space-separated character, representing the character to be pushed onto the stack.

Choice 2: Pop the character from the stack.

Choice 3: Display the characters in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

1. If the choice is 1, push the given character to the stack and display the pushed character having the prefix "Pushed: ".
2. If the choice is 2, undo the character from the stack and display the character that is popped having the prefix "Popped: ".
3. If the choice is 2, and if the stack is empty without any characters, print "Stack is empty. Nothing to pop."
4. If the choice is 3, print the elements in the stack having the prefix "Stack elements: ".
5. If the choice is 3, and there are no characters in the stack, print "Stack is empty."
6. If the choice is 4, exit the program.
7. If any other choice is entered, print "Invalid choice"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 2
4
Output: Stack is empty. Nothing to pop.

Answer

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

```
#include <stdbool.h>

#define MAX_SIZE 100

char items[MAX_SIZE];
int top = -1;

void initialize() {
   top = -1;
}
bool isFull() {
   return top == MAX_SIZE - 1;
}

bool isEmpty() {
   return top == -1;
}

// Function to push a character onto the stack
void push(char value) {
   // Check for stack overflow
   if (top >= MAX_SIZE - 1) {
      printf("Stack Overflow\n"); // Message for array overflow
      return;
   }

   // Increment top and add the element
   top++;
   items[top] = value;

   // Print the push confirmation message as per the sample output format
   printf("Pushed: %c\n", value);
}

// Function to pop a character from the stack
void pop() {
   // Check for stack underflow
   if (top == -1) {
      printf("Stack is empty. Nothing to pop.\n"); // Output format for empty pop
      return;
   }

   // Get the character from the top
```

```
    char poppedValue = items[top];

    // Decrement top
    top--;

    // Print the pop confirmation message as per the sample output format
    printf("Popped: %c\n", poppedValue);
}

// Function to display the characters in the stack
void display() {
    // Check if the stack is empty
    if (top == -1) {
        printf("Stack is empty.\n"); // Output format for empty display
        return;
    }

    // Traverse the array from top down to the first element (index 0)
    printf("Stack elements: "); // Output format for non-empty display
    for (int i = top; i >= 0; i--) {
        printf("%c ", items[i]); // Print character followed by a space
    }
    printf("\n"); // Newline after displaying elements
}

int main() {
    initialize();
    int choice;
    char value;

    while (true) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf(" %c", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: magesh v
Email: 240801187@rajalakshmi.edu.in
Roll no: 240801187
Phone: 6381572897
Branch: REC
Department: I ECE FB
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

You are a software developer tasked with building a module for a scientific calculator application. The primary function of this module is to convert infix mathematical expressions, which are easier for users to read and write, into postfix notation (also known as Reverse Polish Notation). Postfix notation is more straightforward for the application to evaluate because it removes the need for parentheses and operator precedence rules.

The scientific calculator needs to handle various mathematical expressions with different operators and ensure the conversion is correct. Your task is to implement this infix-to-postfix conversion algorithm using a stack-based approach.

Example

Input:

a+b

Output:

ab+

Explanation:

The postfix representation of (a+b) is ab+.

Input Format

The input is a string, representing the infix expression.

Output Format

The output displays the postfix representation of the given infix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: a+(b*e)
Output: abe*+

Answer

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

struct Stack {
   int top;
   unsigned capacity;
   char* array;
};

struct Stack* createStack(unsigned capacity) {
   struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));

   if (!stack)
```

```
        return NULL;

    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (char*)malloc(stack->capacity * sizeof(char));

    return stack;
}

int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

char peek(struct Stack* stack) {
    return stack->array[stack->top];
}

char pop(struct Stack* stack) {
    if (!isEmpty(stack))
        return stack->array[stack->top--];
    return '$';
}

void push(struct Stack* stack, char op) {
    stack->array[++stack->top] = op;
}
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define MAX 100

// Stack for characters (operators and parentheses)
char stack[MAX];
int top = -1;

// Push to stack
void push(char c) {
    if (top < MAX - 1)
        stack[++top] = c;
```

```
}

// Pop from stack
char pop() {
    if (top >= 0)
        return stack[top--];
    return '\0';
}

// Peek top of stack
char peek() {
    if (top >= 0)
        return stack[top];
    return '\0';
}

// Check if character is an operator
int is_operator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}

// Get precedence of operators
int precedence(char op) {
    switch (op) {
        case '+':
        case '-': return 1;
        case '*':
        case '/': return 2;
        case '^': return 3;
        default : return 0;
    }
}

// Check associativity: returns 1 if right associative
int is_right_associative(char op) {
    return op == '^';
}

// Infix to Postfix function
void infix_to_postfix(char* infix, char* postfix) {
    int i, k = 0;
    char ch;
    for (i = 0; infix[i]; i++) {
        ch = infix[i];

        if (isalnum(ch)) {
            postfix[k++] = ch;  // Operand goes directly to output
        } else if (ch == '(') {
            push(ch);
        } else if (ch == ')') {
            while (top != -1 && peek() != '(')
                postfix[k++] = pop();
            pop();  // Pop '('
        } else if (is_operator(ch)) {
            while (top != -1 && peek() != '(' &&
                   (precedence(ch) < precedence(peek()) ||
                   (precedence(ch) == precedence(peek()) && !
is_right_associative(ch)))) {
                postfix[k++] = pop();
            }
            push(ch);
        }
    }

    // Pop remaining operators
    while (top != -1) {
        postfix[k++] = pop();
    }

    postfix[k] = '\0';  // Null-terminate output string
}

// Driver code
int main() {
    char infix[MAX], postfix[MAX];

    // Input expression
    scanf("%s", infix);

    infix_to_postfix(infix, postfix);

    printf("%s\n", postfix);
    return 0;
}
int main() {
    char exp[100];
    scanf("%s", exp);

    infixToPostfix(exp);
    return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: magesh v
Email: 240801187@rajalakshmi.edu.in
Roll no: 240801187
Phone: 6381572897
Branch: REC
Department: I ECE FB
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Milton is a diligent clerk at a school who has been assigned the task of managing class schedules. The school has various sections, and Milton needs to keep track of the class schedules for each section using a stack-based system.

He uses a program that allows him to push, pop, and display class schedules for each section. Milton's program uses a stack data structure, and each class schedule is represented as a character. Help him write a program using a linked list.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the character onto the stack. If the choice is 1, the following input is a space-separated character, representing the class schedule to be pushed onto the stack.

Choice 2: Pop class schedule from the stack

Choice 3: Display the class schedules in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

- If the choice is 1, push the given class schedule to the stack and display the following: "Adding Section: [class schedule]"
- If the choice is 2, pop the class schedule from the stack and display the following: "Removing Section: [class schedule]"
- If the choice is 2, and if the stack is empty without any class schedules, print "Stack is empty. Cannot pop."
- If the choice is 3, print the class schedules in the stack in the following: "Enrolled Sections: " followed by the class schedules separated by space.
- If the choice is 3, and there are no class schedules in the stack, print "Stack is empty"
- If the choice is 4, exit the program and display the following: "Exiting the program"
- If any other choice is entered, print "Invalid choice"

Refer to the sample output for the exact format.

Sample Test Case

Input: 1 d
1 h
3
2

3
4
Output: Adding Section: d
Adding Section: h
Enrolled Sections: h d
Removing Section: h
Enrolled Sections: d
Exiting program

Answer

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

struct Node {
  char data;
  struct Node* next;
};

struct Node* top = NULL;

void push(char value) {
  // Create a new node
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  if (newNode == NULL) {
    // Handle memory allocation failure (acts like overflow in fixed-size stacks)
    printf("Memory allocation failed. Cannot add section.\n"); // Matches
previous output for this case
    return;
  }

  // Assign the class schedule character to the new node
  newNode->data = value;

  // Link the new node to the current top
  newNode->next = top;

  // Update the top of the stack to the new node
  top = newNode;

  // Print the push confirmation message as per the output format
  printf("Adding Section: %c\n", value);
}
```

```
// Function to pop a class schedule (character) from the stack
void pop() {
   // Check if the stack is empty (underflow condition)
   if (top == NULL) {
      printf("Stack is empty. Cannot pop.\n"); // Output format for empty pop
(matches sample with period)
      return;
   }

   // Store the top node to free it later
   struct Node* temp = top;

   // Get the class schedule character before popping
   char poppedValue = temp->data;

   // Update the top to the next node
   top = top->next;

   // Free the memory of the popped node
   free(temp);

   // Print the pop confirmation message as per the output format
   printf("Removing Section: %c\n", poppedValue); // Matches output format
}

// Function to display the class schedules in the stack
void displayStack() {
   // Check if the stack is empty
   if (top == NULL) {
      printf("Stack is empty\n"); // Output format for empty display
      return;
   }

   // Traverse the stack from top to bottom and print elements
   struct Node* current = top;
   printf("Enrolled Sections: "); // Output format for non-empty display
   while (current != NULL) {
      printf("%c ", current->data); // Print character followed by a space
      current = current->next;
   }
   printf("\n"); // Newline after displaying elements
}
int main() {
    int choice;
    char value;
    do {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf(" %c", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                printf("Exiting program\n");
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 4);

    return 0;
}
```

Status : Correct ***Marks : 10/10***