STACK: -

Problem Statement

Julie is enthusiastic about learning data structures, and she has started with one of the fundamental data structures: a stack. She is eager to practice using stacks with various operations like push, pop, and display. Can you help her by designing an interactive program that allows her to perform these stack operations and visualize how the stack changes?

Manages a stack using an array. Supports the following stack operations:

1. Push an element onto the stack.
2. Pop an element from the stack.
3. Display the elements currently in the stack.
4. Exit the program.

Note: This is a sample question asked in a TCS interview.
Input format :
The program expects the following inputs in a loop until the user chooses to exit.

An integer indicating the choice of operation:
1: Push an element onto the stack
2: Pop an element from the stack
3: Display the elements in the stack
4: Exit the program

If the choice is 1, the program expects an integer value to be pushed onto the stack.
Output format :
The program produces the following outputs based on the chosen operation:

Choice 1:
If the push operation is successful, it displays "Element <value> pushed onto the stack."
Choice 2:
If the pop operation is successful, it displays "Element <value> popped from the stack."
If the stack is empty, it displays "Stack Underflow. Cannot perform the pop operation."
Choice 3:
If the stack is empty, it displays "Stack is empty."
If the stack has elements, it displays "Elements in the stack: <element1> <element2> ..."
Choice 4:
If the choice is 4, it displays "Exiting the program."
If the choice is greater than 4 or an invalid input, it displays "Invalid choice."

Each output is followed by a newline character.

Refer to the sample output for the exact format.
Code constraints :
The maximum size of the stack is defined as MAX_SIZE = 100.
0 <= stack elements <= 200
Sample test cases :
Input 1 :

```
1
5
3
2
3
4
```

Output 1 :

```
Element 5 pushed onto the stack.
Elements in the stack: 5
Element 5 popped from the stack.
Stack is empty.
Exiting the program.
```

Input 2 :

```
1
10
1
20
1
30
2
3
4
```

Output 2 :

```
Element 10 pushed onto the stack.
Element 20 pushed onto the stack.
Element 30 pushed onto the stack.
Element 30 popped from the stack.
Elements in the stack: 20 10
Exiting the program.
```

Input 3 :

```
1
25
1
36
2
2
3
2
4
```

Output 3 :

```
Element 25 pushed onto the stack.
Element 36 pushed onto the stack.
Element 36 popped from the stack.
Element 25 popped from the stack.
Stack is empty.
Stack Underflow. Cannot perform pop operation.
Exiting the program.
```

Input 4 :

```
5
3
4
```

Output 4 :

```
Invalid choice.
Stack is empty.
Exiting the program.
```

```
#include <iostream>

using namespace std;


const int MAX_SIZE = 100;
```

```
int stack[MAX_SIZE];
int top = -1;

void push(int value) {
    if (top == MAX_SIZE - 1) {
        return;
    }

    stack[++top] = value;
    cout << "Element " << value << " pushed onto the stack." << endl;
}

void pop() {
    if (top == -1) {
        cout << "Stack Underflow. Cannot perform pop operation." << endl;
        return;
    }

    int element = stack[top--];
    cout << "Element " << element << " popped from the stack." << endl;
}

void displayStack() {
    if (top == -1) {
        cout << "Stack is empty." << endl;
        return;
    }

    std::cout << "Elements in the stack: ";
    for (int i = top; i >= 0; --i) {
```

```
        cout << stack[i] << " ";
    }
    cout << endl;
}

int main() {
    int choice, value;

    do {

        std::cin >> choice;

        switch (choice) {
            case 1:

                cin >> value;
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                cout << "Exiting the program." << endl;
                break;
            default:
                cout << "Invalid choice." << endl;
        }
    } while (choice != 4);
```

```
    return 0;
}
```

Problem Statement

Naveen is learning about data structures and wants to implement a stack data structure using linked list. He also needs to perform a specific operation on the stack: delete even numbers from it. Can you help him write a program to create a stack, push elements onto it, delete even numbers, and display the final stack?

implements a stack using a linked list and performs the following operations:

1. **push (int data):** Add an integer element to the top of the stack.
2. **pop():** Remove the top element from the stack.
3. **deleteEven():** Delete all even numbers from the stack.
4. **printStack():** Display the elements in the stack after pushing and deleting even numbers.

Note: This is a sample question asked in TCS recruitment.

Input format :

The first line contains an integer n, representing the number of elements Nandha wants to push onto the stack.

The next line contains n space-separated integers, each representing the elements to be pushed onto the stack.

Output format :

The output displays the following format:

1. After pushing all elements onto the stack, display the elements in the stack separated by a space.
2. After removing even numbers from the stack using the deleteEven function, print the remaining elements in the stack separated by a space.

Refer to the sample output for the formatting specifications.

Code constraints :

1 <= n <=12

1 <= elements <= 100

Sample test cases :

Input 1 :

```
5
12 45 78 35 59
```

Output 1 :

```
59 35 78 45 12
59 35 45
```

Input 2 :

```
8
12 45 78 23 56 78 99 65
```

Output 2 :

```
65 99 78 56 23 78 45 12
65 99 23 45
```

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

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

struct Stack {
   struct Node* top;
};

struct Node* newNode(int data)
{
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->data = data;
   node->next = NULL;
   return node;
}

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

void push(struct Stack* stack, int data)
{
   struct Node* node = newNode(data);
   node->next = stack->top;
   stack->top = node;
```

```
}

int pop(struct Stack* stack)
{
   if (isEmpty(stack))
      return -1;
   struct Node* temp = stack->top;
   stack->top = stack->top->next;
   int popped = temp->data;
   free(temp);
   return popped;
}

void printStack(struct Stack* stack)
{
   struct Node* curr = stack->top;
   while (curr != NULL) {
      cout<<curr->data<<" ";
      curr = curr->next;
   }
}

void deleteEven(struct Stack* stack)
{
   struct Stack* temp = (struct Stack*)malloc(sizeof(struct Stack));
   temp->top = NULL;
   while (!isEmpty(stack)) {
      int val = pop(stack);
      if (val % 2 == 1)
         push(temp, val);
   }
```

```
    while (!isEmpty(temp)) {
        push(stack, pop(temp));
    }
    printStack(stack);
}

int main()
{
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->top = NULL;
    int i, n, ele;
    cin>>n;
    for (i = 0; i < n; i++) {
        cin>>ele;
        push(stack, ele);
    }
    printStack(stack);
    cout<<endl;
    deleteEven(stack);
    return 0;
}
```

Problem Statement

Lara is learning about linked lists in her computer science class and wants to practice performing operations on linked lists. Specifically, she's interested in deleting the middle element of a linked list. Can you help her by designing a program that allows her to delete the middle element from a linked list and visualize the result?

- Manages a singly linked list.
- Implements a function to delete the middle element from the linked list.
- Provides a function to push elements onto the linked list.
- Displays the elements of the linked list after deleting the middle element.

Note: This is a sample question asked in CTS recruitment.

Input format :

The input consists of an integer n from the user, indicating the number of elements to be pushed onto the linked list.

The linked list is initially filled with integers from n down to 1.

Output format :

The output prints the elements of the linked list after deleting the middle element.

Refer to the sample output for the exact format.

Code constraints :

n should be odd.

3 <= n <= 51

Sample test cases :

Input 1 :

```
7
```

Output 1 :

```
1 2 3 5 6 7
```

Input 2 :

```
11
```

Output 2 :

```
1 2 3 4 5 7 8 9 10 11
```

```
#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

struct Node {

   int data;

   struct Node* next;

};


void deleteMid(struct Node** head_ref, int n) {

   if (*head_ref == NULL || n == 0) {

      return;

   }


   if (n == 1) {

      struct Node* temp = *head_ref;

      *head_ref = temp->next;

      free(temp);

      return;

   }
```

```
    struct Node* prev = NULL;
    struct Node* curr = *head_ref;
    int count = 0;

    while (curr != NULL) {
        if (count == n / 2) {
            prev->next = curr->next;
            free(curr);
            return;
        }

        prev = curr;
        curr = curr->next;
        count++;
    }
}

void push(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void printList(struct Node* node) {
    while (node != NULL) {
        cout<< node->data<<" ";
        node = node->next;
    }
}
```

```
int main() {
   struct Node* head = NULL;
   int i, n;
   cin>>n;
   for (i = n; i > 0; i--) {
      push(&head, i);
   }
   deleteMid(&head, n);
   printList(head);
   return 0;
}
```

Problem Statement

Laddu has implemented a simple stack using an array with a maximum size of MAX_SIZE (defined as 10 in the code). The program provides the following operations:

1. **Push**: Add an integer to the stack.
2. **Pop:** Remove an integer from the top of the stack.
3. **Check if Stack is Full:** Determine if the stack is full.
4. **Check if Stack is Empty:** Determine if the stack is empty.
5. **Exit:** Terminate the program.

Note: This is a sample question asked in a Capgemini interview.
Input format :
If the choice represents the operation to be performed,
1: Push an integer onto the stack.
2: Pop an integer from the stack.
3: Check if the stack is full.
4: Check if the stack is empty.
5: Exit the program.
item, represents the integer to be pushed onto the stack, separated by a space.
Output format :
The output prints one of the following messages:
If the choice is 1 (Push), there is no output.
If the choice is 2 (Pop) and the stack is not empty, print the popped integer. and the stack is empty, print "Stack is empty!".
If the choice is 3 (Check if Full), and the stack is full, print "Stack is full!", and if the stack is not full, print "Stack is not full.".
If the choice is 4 (Check if Empty), and the stack is empty, print "Stack is empty!", and if the stack is not empty, print "Stack is not empty."
If the choice is 5 (Exit), the program should terminate immediately.
If the choice is more than 6, It prints "Invalid input".

Refer to the sample output for the formatting specifications.

Code constraints :

1 <= choice <= 5

Sample test cases :

Input 1 :

```
1 2
1 6
2
3
4
5
```

Output 1 :

```
Stack is not full.
Stack is not empty.
```

Input 2 :

```
1 3
1 4
1 5
1 7
1 18
1 19
1 20
1 21
1 34
1 76
4
3
5
```

Output 2 :

```
Stack is not empty.
Stack is full!
```

Input 3 :

```
1 2
1 4
1 5
1 76
1 89
1 34
1 23
1 65
1 45
1 23
2
2
2
2
2
2
2
2
2
2
4
5
```

Output 3 :

```
Stack is empty!
```

Input 4 :

```
1 6
1 7
9
```

```
5
```

Output 4 :

```
Invalid choice
```

Input 5 :

```
1 4
2
2
5
```

Output 5 :

```
Stack is empty!
```

```
#include <iostream>

using namespace std;

#define MAX_SIZE 10

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

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

void push(int stack[], int &top, int item) {
   if (isFull(top)) {
      cout << "Stack is full!" << endl;
      return;
   }
   stack[++top] = item;
}

int pop(int stack[], int &top) {
   if (isEmpty(top)) {
```

```
        cout << "Stack is empty!" << endl;
        return -1;
    }
    return stack[top--];
}

int main() {
    int stack[MAX_SIZE];
    int top = -1;
    int choice, item;

    while (true) {
        cin >> choice;

        switch (choice) {
            case 1:
                cin >> item;
                push(stack, top, item);
                break;
            case 2:
                pop(stack, top);
                break;
            case 3:
                if (isFull(top)) {
                    cout << "Stack is full!" << endl;
                } else {
                    cout << "Stack is not full." << endl;
                }
                break;
            case 4:
                if (isEmpty(top)) {
```

```
            cout << "Stack is empty!" << endl;
        } else {
            cout << "Stack is not empty." << endl;
        }
        break;
      case 5:
        exit(0);
      default:
        cout << "Invalid choice" << endl;
    }
  }

  return 0;
}
```

Problem Statement

Mira wants to create a program that allows her to manipulate a stack, perform push and pop operations, and display the elements in the stack. She would like a user-friendly interface that presents a menu with the following options:

1. **Push Operation:** Add an integer to the top of the stack.
2. **Pop Operation:** Remove and discard the integer from the top of the stack.
3. **Display Stack:** Display all the elements currently in the stack, from top to bottom.
4. **Exit the Program:** Terminate the program.

Note: This is a sample question asked in a Capgemini interview.
Input format :
The program expects Ganga to enter a choice (an integer) corresponding to the operation she wants to perform:
1: Push the integer value onto the stack.
2: Pop an integer from the stack.
3: Display the elements currently in the stack.
4: Exit the program.
Output format :
The output displays messages according to the chosen menu option and the status of the stack:

"Stack is empty. Cannot perform a pop operation." when attempting to pop from an empty stack.
"Stack is empty." when the stack is empty.
"Elements in the stack:..." followed by the elements in the stack when choosing option 3.
"Exiting the program." when choosing option 4.
"Invalid choice." when entering an invalid menu choice.

Refer to the sample output for the exact format.

Code constraints :
Choice = 1, 2, 3, 4
0 <= Stack elements <= 50
Sample test cases :
Input 1 :

```
1
5
1
10
1
25
2
3
4
```

Output 1 :

```
Elements in the stack: 10 5
Exiting the program.
```

Input 2 :

```
1
5
3
4
```

Output 2 :

```
Elements in the stack: 5
Exiting the program.
```

Input 3 :

```
1
2
1
7
2
2
3
4
```

Output 3 :

```
Stack is empty.
Exiting the program.
```

Input 4 :

```
1
4
2
2
3
4
```

Output 4 :

```
Stack is empty. Cannot perform pop operation.
Stack is empty.
Exiting the program.
```

```
#include <iostream>

using namespace std;


const int MAX_SIZE = 100; // Maximum stack size
```

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

Node* top = nullptr;
Node stackArray[MAX_SIZE];
int stackSize = 0;

void push(int value) {
    stackArray[stackSize].data = value;
    stackArray[stackSize].next = top;
    top = &stackArray[stackSize];
    stackSize++;
}

void pop() {
    if (top == nullptr) {
        cout << "Stack is empty. Cannot perform pop operation." << endl;
        return;
    }

    top = top->next;
    stackSize--;
}

// Function to display the elements in the stack
void displayStack() {
    if (top == nullptr) {
        cout << "Stack is empty." << endl;
```

```
        return;
    }

    Node* current = top;
    cout << "Elements in the stack: ";
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

int main() {
    int choice, value;

    do {
        cin >> choice;

        switch (choice) {
            case 1:
                cin >> value;
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                cout << "Exiting the program." << endl;
```

```
            break;
        default:
            cout << "Invalid choice." << endl;
    }
  } while (choice != 4);

  return 0;
}
```

Problem Statement

Yogi is learning data structure and he wants to write a program to accept multiple infix expressions from the user and convert them into postfix expressions using a stack-based algorithm. The program should prompt the user to enter the number of expressions they wish to convert, and then accept each expression one by one. The program should validate each expression for proper syntax before converting it to postfix.

After converting each infix expression to a postfix, the program should print the corresponding postfix expression to the console. Finally, the program should ask the user if they want to continue converting more expressions or exit the program.

Note: This is a sample question asked in TCS recruitment.
Input format :
The first line of input contains an integer **n** denoting the number of infix expressions to be converted.
The next n lines contain the infix expressions to be converted.
Output format :
The output prints the corresponding postfix expression for **n** inputs on separate lines.

Refer to the sample output for the formatting specifications.
Code constraints :
1 <= n <= 10
The maximum length of an infix expression is 100 characters.
The program should handle the following operators with precedence levels in descending order:

- Parentheses: ()
- Exponentiation: ^
- Multiplication: *
- Division: /
- Addition: +
- Subtraction: -

Sample test cases :
Input 1 :

```
1
A+B*C-D/E^F
```

Output 1 :

```
Postfix expression 1: ABC*+DEF^/-
```

Input 2 :

```
2
A+B-C
```

```
D+E/F-G
```

Output 2 :

```
Postfix expression 1: AB+C-
Postfix expression 2: DEF/+G-
```

```
#include <iostream>
#include <string>

using namespace std;

const int MAX_EXPR_LEN = 100;

int is_operator(char c) {
    if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^' || c == '(' || c == ')') {
        return 1;
    }
    return 0;
}

int precedence(char c) {
    if (c == '^') {
        return 3;
    } else if (c == '*' || c == '/') {
        return 2;
    } else if (c == '+' || c == '-') {
        return 1;
    } else {
        return 0;
    }
}

struct CharStack {
```

```
char data[MAX_EXPR_LEN];
int top;

CharStack() {
   top = -1;
}

void push(char c) {
   if (top < MAX_EXPR_LEN - 1) {
      data[++top] = c;
   } else {
      cout << "Stack overflow!" << endl;
      exit(EXIT_FAILURE);
   }
}

char pop() {
   if (top >= 0) {
      return data[top--];
   } else {
      cout << "Stack underflow!" << endl;
      exit(EXIT_FAILURE);
   }
}

char peek() {
   if (top >= 0) {
      return data[top];
   } else {
      return '\0'; // Return null character for an empty stack.
   }
```

```
    }

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

string infix_to_postfix(const string& infix) {
    CharStack operators;
    string postfix;
    char c;

    for (int i = 0; i < infix.length(); i++) {
        c = infix[i];
        if (!is_operator(c)) {
            postfix += c;
        } else {
            if (c == '(') {
                operators.push(c);
            } else if (c == ')') {
                while (operators.peek() != '(') {
                    postfix += operators.pop();
                }
                operators.pop(); // Pop the '('
            } else {
                while (!operators.empty() && operators.peek() != '(' && precedence(c) <=
precedence(operators.peek())) {
                    postfix += operators.pop();
                }
                operators.push(c);
            }
```

```
        }
    }

    while (!operators.empty()) {
        postfix += operators.pop();
    }

    return postfix;
}

int main() {
    int num_expressions;
    string infix, postfix;

    cin >> num_expressions;

    for (int i = 1; i <= num_expressions; i++) {
        cin >> infix;
        postfix = infix_to_postfix(infix);
        cout << "Postfix expression " << i << ": " << postfix << endl;
    }

    return 0;
}
```

Problem Statement

Venu is currently learning about postfix expressions in his computer science class. He has recently written a program to validate whether a given postfix expression is valid or not. However, he wants you to help him.

A valid postfix expression is one that adheres to the following rules:

1. It contains only digits, operators (+, -, *, /), and spaces.

2. It has valid operator placement, meaning that there must be at least two operands for every operator, and no operands should be left unused.
3. Division by zero is not allowed.

Example

Input 1
23+
Output 1
Valid postfix expression

Explanation:

In this input, The expression has two operands, 2 and 3, and one operator, +. The operator + is correctly placed after the two operands. When you evaluate this expression, you add operands 2 and 3, resulting in 5. So, the expression is a valid postfix expression, and the output is "Valid postfix expression."

Input 2
42*+
Output 2
Invalid postfix expression

Explanation:

In this input, the postfix expression is 42*+. The expression has two operands, 4 and 2, and two operators * and +.
However, the operator + is placed immediately after the operand 2, which is not allowed in postfix notation. There should be at least two operands for every operator. So, the expression is invalid due to incorrect operator-placement, and the output is "Invalid postfix expression."

Note: This is a sample question asked in TCS recruitment.
Input format :
The input consists of a single line containing the string S, which represents the postfix expression to be validated.
Output format :
The output displays either "Valid postfix expression" or "Invalid postfix expression" based on whether the input expression is valid or not.

Refer to the sample output for the formatting specifications.
Code constraints :
1 <= |S| <= 100
Sample test cases :
Input 1 :

```
23+
```

Output 1 :

```
Valid postfix expression
```

Input 2 :

```
42*+
```

Output 2 :

```
Invalid postfix expression
```

```
#include <iostream>
```

```
#include <cstring>

using namespace std;

bool isOperator(char c) {
   return (c == '+' || c == '-' || c == '*' || c == '/');
}

bool isValidPostfixExpression(const char* postfix) {
   int stack[100];
   int top = -1;

   for (int i = 0; postfix[i]; i++) {
      char c = postfix[i];

      if (isdigit(c)) {
         int operand = 0;
         while (isdigit(c)) {
            operand = operand * 10 + (c - '0');
            i++;
            c = postfix[i];
         }
         stack[++top] = operand;
      } else if (isOperator(c)) {
         if (top < 1) {
            return false;
         }
         int operand2 = stack[top--];
         int operand1 = stack[top--];

         switch (c) {
```

```
            case '+':
                stack[++top] = operand1 + operand2;
                break;
            case '-':
                stack[++top] = operand1 - operand2;
                break;
            case '*':
                stack[++top] = operand1 * operand2;
                break;
            case '/':
                if (operand2 == 0) {
                    return false; // Division by zero
                }
                stack[++top] = operand1 / operand2;
                break;
        }
    } else if (c != ' ') {
        return false;
    }
  }

  return (top == 0);
}

int main() {
  char postfixExpression[100];

  cin.getline(postfixExpression, sizeof(postfixExpression));

  if (isValidPostfixExpression(postfixExpression)) {
    cout << "Valid postfix expression" << endl;
```

```
    } else {
        cout << "Invalid postfix expression" << endl;
    }

    return 0;
}
```

Problem Statement

In a prestigious educational institute, Professor Smith designs a programming challenge for Computer Science students. As part of an assessment on Data Structures and Algorithms, students are presented with a postfix expression that involves mathematical operations.

The challenge requires students to develop a robust algorithm to evaluate these expressions accurately and efficiently. This exercise not only hones their coding skills but also emphasizes the importance of understanding stack-based computations, enhancing their problem-solving capabilities for real-world software development scenarios.

Write a program to evaluate the given postfix expression and display the result.

Note: This is a sample question asked in Infosys recruitment.
Input format :
The input consists of a postfix mathematical expression.
The expression will contain real numbers and mathematical operators (+, -, *, /), without any space.
Output format :
The output prints the result of evaluating the given postfix expression.

Refer to the sample output for formatting specifications.
Code constraints :
The arithmetic operators to be included in the expression are +, -, *, and /.
Sample test cases :
Input 1 :

```
82/
```

Output 1 :

```
4
```

Input 2 :

```
545*+5/
```

Output 2 :

```
5
```

Input 3 :

```
82-4+
```

Output 3 :

```
10
```

```
#include <iostream>

#include <cstdlib>
```

```
using namespace std;

#define MAX_SIZE 100

struct Stack {
    int data[MAX_SIZE];
    int top;
};

void initialize(struct Stack* stack) {
    stack->top = -1;
}

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

void push(struct Stack* stack, int value) {
    if (stack->top == MAX_SIZE - 1) {
        cout << "Stack Overflow!" << endl;
        exit(EXIT_FAILURE);
    }
    stack->data[++(stack->top)] = value;
}

int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        cout << "Stack Underflow!" << endl;
        exit(EXIT_FAILURE);
    }
```

```
    return stack->data[(stack->top)--];
}

bool isDigit(char ch) {
    return (ch >= '0' && ch <= '9');
}

bool isOperator(char ch) {
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/');
}

int evalPostfix(char postfix[]) {
    struct Stack stack;
    initialize(&stack);

    int i = 0;
    while (postfix[i] != '\0') {
        char ch = postfix[i];

        if (isDigit(ch)) {
            push(&stack, ch - '0');
        } else if (isOperator(ch)) {
            int x = pop(&stack);
            int y = pop(&stack);

            int result;
            if (ch == '+') {
                result = y + x;
            } else if (ch == '-') {
                result = y - x;
            } else if (ch == '*') {
```

```
            result = y * x;
        } else if (ch == '/') {
            result = y / x;
        } else {
            exit(EXIT_FAILURE);
        }

        push(&stack, result);
      }
      i++;
   }

   return pop(&stack);
}

int main() {
   char postfix[MAX_SIZE];
   cin.getline(postfix, sizeof(postfix));

   int result = evalPostfix(postfix);
   cout << result << endl;

   return 0;
}
```

Problem Statement:

Meenu is studying computer science and is currently learning about expressions in infix notation. She needs a program to convert infix expressions to postfix notation to help her with her studies. Can you help her by providing a program that performs this conversion?

The program should support the following operations:

- Check if a character is an operator (+, -, *, or /).
- Determine the precedence of an operator.
- Convert an infix expression to postfix notation.

Example

Input:
(3+4)5
Output:
34+5

Note: This is a sample question asked in TCS recruitment.
Input format :
The input consists of the infix expression to be converted to postfix notation.
Output format :
The output displays the postfix expression equivalent of the input infix expression.

Refer to the sample output for formatting specifications.
Code constraints :
The input line will not exceed a length of 100 characters.
The infix expression may contain operators (+, -, *, /) and parentheses (,).
Sample test cases :
Input 1 :

```
A+B*C-D/E
```

Output 1 :

```
Postfix expression: ABC*+DE/-
```

Input 2 :

```
3+4*5/(6-2)
```

Output 2 :

```
Postfix expression: 345*62-/+
```

Input 3 :

```
(3+4)5
```

Output 3 :

```
Postfix expression: 34+5
```

```
#include <iostream>

#include <string>


using namespace std;


const int MAX_SIZE = 100;  // Maximum stack size


bool isOperator(char ch) {

   if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {

      return true;

   }

   return false;
```

```
}

int precedence(char ch) {
    if (ch == '+' || ch == '-') {
        return 1;
    } else if (ch == '*' || ch == '/') {
        return 2;
    }
    return 0;
}

void infixToPostfix(string infix, string& postfix) {
    char stack[MAX_SIZE];
    int top = -1;  // Stack top pointer

    for (int i = 0; i < infix.length(); i++) {
        char ch = infix[i];
        if (isalnum(ch)) {
            postfix += ch;
        } else if (isOperator(ch)) {
            while (top >= 0 && stack[top] != '(' && precedence(ch) <= precedence(stack[top])) {
                postfix += stack[top];
                top--;
            }
            top++;
            stack[top] = ch;
        } else if (ch == '(') {
            top++;
            stack[top] = ch;
        } else if (ch == ')') {
            while (top >= 0 && stack[top] != '(') {
```

```
            postfix += stack[top];
            top--;
        }
        if (top >= 0 && stack[top] == '(') {
            top--;
        }
    }
  }

  while (top >= 0) {
    postfix += stack[top];
    top--;
  }
}

int main() {
  string infix, postfix;
  cin >> infix;

  infixToPostfix(infix, postfix);

  cout << "Postfix expression: " << postfix << endl;

  return 0;
}
```

Problem Statement

Janani is a dedicated math teacher who wants to simplify the process of solving complex mathematical expressions for her students.

She wishes to create a tool that can convert standard mathematical expressions written in infix notation into postfix notation. This tool will help her students better understand the order of operations and practice solving equations step-by-step.

Input format :

The input consists of a single line containing a mathematical expression in infix notation.
The expression contains the operators +, -, *, and /, as well as parentheses ().
Operand values are integers, and there are no spaces in the input.

Output format :

The output will be a single line containing the corresponding postfix expression obtained from the given infix expression.

Refer to the sample output for the formatting specifications.

Code constraints :

The input expression has at most 100 characters.and valid mathematical expression.

Sample test cases :

Input 1 :

```
(5-3)*(7+2)
```

Output 1 :

```
Postfix expression: 53-72+*
```

Input 2 :

```
(1+7)*3-8/(5+2)
```

Output 2 :

```
Postfix expression: 17+3*852+/-
```

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>


#define MAX_SIZE 100


// Stack structure

typedef struct {

   char data[MAX_SIZE];

   int top;

} Stack;


void initialize(Stack *s) {

   s->top = -1;

}


void push(Stack *s, char value) {

   if (s->top < MAX_SIZE - 1) {
```

```
        s->data[++(s->top)] = value;
    } else {
        printf("Stack Overflow\n");
        exit(EXIT_FAILURE);
    }
}

char pop(Stack *s) {
    if (s->top >= 0) {
        return s->data[(s->top)--];
    } else {
        printf("Stack Underflow\n");
        exit(EXIT_FAILURE);
    }
}

int isOperator(char ch) {
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/');
}

int precedence(char op) {
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return 0;
}

void infixToPostfix(const char *infix, char *postfix) {
    Stack operatorStack;
    initialize(&operatorStack);
```

```
    int i = 0, j = 0;


    while (infix[i] != '\0') {
        if (isdigit(infix[i])) {
            postfix[j++] = infix[i];
        } else if (isOperator(infix[i])) {
            while (operatorStack.top >= 0 && precedence(operatorStack.data[operatorStack.top]) >=
precedence(infix[i])) {
                postfix[j++] = pop(&operatorStack);
            }
            push(&operatorStack, infix[i]);
        } else if (infix[i] == '(') {
            push(&operatorStack, infix[i]);
        } else if (infix[i] == ')') {
            while (operatorStack.top >= 0 && operatorStack.data[operatorStack.top] != '(') {
                postfix[j++] = pop(&operatorStack);
            }
            if (operatorStack.top >= 0 && operatorStack.data[operatorStack.top] == '(') {
                pop(&operatorStack);
            }
        }
        i++;
    }


    while (operatorStack.top >= 0) {
        if (operatorStack.data[operatorStack.top] == '(') {
            exit(EXIT_FAILURE);
        }
        postfix[j++] = pop(&operatorStack);
    }
```

```
    postfix[j] = '\0';
}

int main() {
    char infix[MAX_SIZE], postfix[MAX_SIZE];
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```