Doubly linked lists and stacks

1. Reverse a doubly linked list and print the elements

Source code

```c
//
//  main.c
//  Lab-3-1
//
//  Created by Aaron R on 27/08/21.
//  Reverse a doubly linked list and print the elements

#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *head, *tail;
int length=0;

void display(void)
{
    // #### To display the elements of the doubly linked list ####
    struct node * temp;
    temp = head;
    if (head==NULL)
    {
        printf("The doubly linked list is empty\n");
    }

    else
    {
        // traversing from the first node
        while (temp->next!=NULL)
        {
            printf("%d\n",temp->data);
            temp = temp->next;
        }
        // printing the last node's data
        printf("%d\n",temp->data);
    }
}
void initList(void)
{
    // #### To initialise a doubly linked list with n number of elements ####
    struct node *new;
    printf("Enter the number of nodes with which to initialise the doubly linked list: ");
    int n;
    scanf("%d",&n);
    length = n;
    for (int i=0; i<n; i++)
    {
        new = (struct node *)malloc(sizeof(struct node));
        printf("Enter data of node %d: ",i+1);
        scanf("%d",&new->data);
        new->next = NULL;
        new->prev = NULL;
        if (head == NULL)
        {
            // If there are no elements in the doubly linked list
            tail = new;
            head = tail;
        }
```

```c
        else
        {
            // If there are nodes in the doubly linked list
            tail->next = new;
            new->prev = tail;
            tail = new;
        }
    }
}

void insert_head(void)
{
    // #### To insert a node at the head ####
    struct node *new;
    new = (struct node *)malloc(sizeof(struct node));
    printf("Enter data of new node: ");
    scanf("%d",&new->data);
    new->next = NULL;
    new->prev = NULL;
    if (head==NULL)
    {
        // If there are no nodes in the doubly linked list
        head = new;
        tail = new;
    }
    else
    {
        // If there are nodes in the doubly linked list
        // Update the previous pointer of the head node from NULL to the new node
        head->prev = new;

        // Update the next pointer of the new node to the previous head node
        new->next = head;

        // Update the head pointer to the new node
        head = new;
    }
    length+=1;
}

void insert_end(void)
{
    // #### To insert a node at the tail ####
    struct node *new;
    new = (struct node *)malloc(sizeof(struct node));
    printf("Enter data of new node: ");
    scanf("%d",&new->data);
    new->next = NULL;
    new->prev = NULL;
    if (head==NULL)
    {
        // If there are no nodes in the doubly linked list
        head = new;
        tail = new;
    }
    else
    {
        // If there are nodes in the doubly linked list
        // Update the next pointer of the tail node from NULL to the new node
        tail->next = new;

        // Update the previous pointer of the new node to the previous tail node
        new->prev = tail;

        // Update the tail pointer to the new node
        tail = new;
    }
    length+=1;
}
```

```c
void insert_pos(int p)
{
    // #### To insert a node at the pth position ####
    struct node *new,*temp;
    new = (struct node *)malloc(sizeof(struct node));
    temp = head;
    printf("Enter data of new node: ");
    scanf("%d",&new->data);
    new->next = NULL;
    new->prev = NULL;
    int i=1;
    while (i<p-1)
    {
        temp = temp->next;
        i++;
    }
    // Updating the prev pointer to p-1th node
    new->prev = temp;

    // updating the next node to p1th node
    new->next = temp->next;

    // Updating the next pointer of p-1th node to the new node
    temp->next = new;

    // Updating the previous pointer of the pth node to the new node
    new->next->prev = new;

    length+=1;
}

void reverse_list(void)
{
    // #### To reverse a doubly linked list ####

    struct node *current_node, *next_node;
    current_node = head;
    while (current_node!=NULL)
    {
        // Storing the next node in next_node
        next_node = current_node->next;

        // Swapping the prev and next pointers of the current_node
        current_node->next = current_node->prev;
        current_node->prev = next_node;

        // Updating current_node to the next node
        current_node = next_node;
    }

    // Swapping head and tail pointers;
    current_node = head;
    head = tail;
    tail = current_node;
}

int main(int argc, const char * argv[])
{
    head = NULL;
    // Initialising doubly linked list
    initList();
    display();

    int more_nodes = 0;
    printf("Would you like to insert more nodes?\n1 - Yes\n");
    scanf("%d",&more_nodes);
```

```
    {
        while (more_nodes==1)
        {
            printf("####Positions####\n0 : beginning\n%d or -1 : end\nBetween 0 and %d : position p\nEnter the
                position to insert the new node: ",length,length);
            int p;
            scanf("%d",&p);
            if (p==0)
                insert_head();
            else if (p==-1 || p==length)
                insert_end();
            else if (p>1 && p<length)
                insert_pos(p);
            else
                printf("Invalid position");

            printf("Current elements of doubly linked lists: \n");
            display();

            printf("Would you like to insert another node?\n1 - Yes\n");
            scanf("%d",&more_nodes);
        }
    }

    printf("Reversed doubly linked list:\n");
    // Reversing the doubly linked list
    reverse_list();
    display();

    return 0;
}
```

```
Enter the number of nodes with which to initialise the doubly linked list: 3
Enter data of node 1: 10
Enter data of node 2: 20
Enter data of node 3: 30
10
20
30
Would you like to insert more nodes?
1 - Yes
1
####Positions####
0 : beginning
3 or -1 : end
Between 0 and 3 : position p
Enter the position to insert the new node: 0
Enter data of new node: 40
Current elements of doubly linked lists:
40
10
20
30
Would you like to insert another node?
1 - Yes
1
####Positions####
0 : beginning
4 or -1 : end
Between 0 and 4 : position p
Enter the position to insert the new node: 2
Enter data of new node: 50
Current elements of doubly linked lists:
40
50
10
20
30
```

Output(s):

```
Would you like to insert another node?
1 - Yes
1
####Positions####
0 : beginning
5 or -1 : end
Between 0 and 5 : position p
Enter the position to insert the new node: 5
Enter data of new node: 60
Current elements of doubly linked lists:
40
50
10
20
30
60
Would you like to insert another node?
1 - Yes
2
Reversed doubly linked list:
60
30
20
10
50
40
Program ended with exit code: 0
```

All Output ⌃ Filter

2. Create a doubly linked list and delete a node at the beginning, end and any position in between.

Source Code

```c
//
//  main.c
//  Lab-3-2
//
//  Created by Aaron R on 27/08/21.

//  Create a doubly linked list and
//  delete a node at the beginning, end and any position in between.

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

struct node
{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *head, *tail;
int length=0;

void display(void)
{
    // #### To display the elements of the doubly linked list ####
    struct node * temp;
    temp = head;
    if (head==NULL)
    {
        printf("The doubly linked list is empty\n");
    }

    else
    {
        // traversing from the first node
        while (temp->next!=NULL)
        {
            printf("%d\n",temp->data);
            temp = temp->next;
        }
        // printing the last node's data
        printf("%d\n",temp->data);
    }
}
```

```c
void initList(void)
{
    // #### To initialise a doubly linked list with n number of elements ####
    struct node *new;
    printf("Enter the number of nodes with which to initialise the doubly linked list: ");
    int n;
    scanf("%d",&n);
    length = n;
    for (int i=0; i<n; i++)
    {
        new = (struct node *)malloc(sizeof(struct node));
        printf("Enter data of node %d: ",i+1);
        scanf("%d",&new->data);
        new->next = NULL;
        new->prev = NULL;
        if (head == NULL)
        {
            // If there are no elements in the doubly linked list
            tail = new;
            head = tail;
        }
        else
        {
            // If there are nodes in the doubly linked list
            tail->next = new;
            new->prev = tail;
            tail = new;
        }
    }
}

void delete_head(void)
{
    // #### To delete a node whose position is the head ####

    struct node * temp;
    temp = head;
    if (head == tail)
    {
        head = tail = NULL;
        length-=1;
    }
    else
    {
        // Updating head to the next element
        head = head->next;

        // Updating the prev pointer of head is NULL
        head->prev = NULL;
        free(temp);
        length-=1;
    }

}

void delete_end(void)
{
    // #### To delete a node whose position is the tail ####
    struct node * temp;
    temp = tail;
    if (head == tail)
    {
        head = tail = NULL;
        length-=1;
    }

```

```c
    else
    {
        // Updating the tail pointer to the previous element
        tail = tail->prev;

        // The new tail pointer's next pointer is updated to NULL
        tail->next = NULL;
        free(temp);
        length-=1;
    }

}

void delete_pos(int p)
{
    // #### To delete a node whose position is p ####

    struct node * temp;
    temp = head;
    int i=1;

    // Traversing to get to the pth node
    while (i<p)
    {
        temp = temp->next;
        i++;
    }
    // Updating the p-1th node's next poitner to p+1th node
    temp->prev->next = temp->next;

    // Updating the p+1th node's prev pointer to p-1th node
    temp->next->prev = temp->prev;
    free(temp);
    length-=1;
}

int main(int argc, const char * argv[])
{
    head = NULL;
    tail = NULL;
    // Initialising doubly linked list
    initList();
    display();

    int cond;
    printf("Would you like to delete nodes?\n1 - Yes\n");
    scanf("%d",&cond);
    while (cond==1)
    {
        if (length==0)
        {
            printf("There are no more nodes left to delete\n");
            break;
        }
        else
        {
            printf("#### Positions to delete ####\n1 - head\n-1 or %d - tail\nBetween 1 and %d - pth
                element\nEnter the position to delete: ",length,length);
            int p;
            scanf("%d",&p);
            if (p==1)
                delete_head();
            else if(p==-1 || p==length)
                delete_end();
            else if(p>1 && p<length)
                delete_pos(p);
            else
                printf("Invalid position\n");
```

```
            printf("Current elements in doubly linked list: \n");
            display();
            printf("Enter 1 to delete more nodes: ");
            scanf("%d",&cond);
        }

    }

    return 0;
}
```

Output(s):

```
Enter the number of nodes with which to initialise the doubly linked list: 5
Enter data of node 1: 10
Enter data of node 2: 20
Enter data of node 3: 30
Enter data of node 4: 40
Enter data of node 5: 50
10
20
30
40
50
Would you like to delete nodes?
1 - Yes
1
#### Positions to delete ####
1 - head
-1 or 5 - tail
Between 1 and 5 - pth element
Enter the position to delete: 1
Current elements in doubly linked list:
20
30
40
50
Enter 1 to delete more nodes: 1
#### Positions to delete ####
1 - head
-1 or 4 - tail
Between 1 and 4 - pth element
Enter the position to delete: 4
Current elements in doubly linked list:
20
30
40
Enter 1 to delete more nodes: 1
#### Positions to delete ####
1 - head
-1 or 3 - tail
Between 1 and 3 - pth element
Enter the position to delete: 2
Current elements in doubly linked list:
20
40
```

```
Enter 1 to delete more nodes: 1
#### Positions to delete ####
1 - head
-1 or 2 - tail
Between 1 and 2 - pth element
Enter the position to delete: 1
Current elements in doubly linked list:
40
Enter 1 to delete more nodes: 1
#### Positions to delete ####
1 - head
-1 or 1 - tail
Between 1 and 1 - pth element
Enter the position to delete: 1
Current elements in doubly linked list:
The doubly linked list is empty
Enter 1 to delete more nodes: 1
There are no more nodes left to delete
Program ended with exit code: 0
```

All Output ⌃ Filter

3. Devise a program to create a list of integers dynamically .Write functions (i) Find the maximum and minimum from the list. (iii) Display the contents of the list.

Source Code

```c
//
//  main.c
//  Lab-3-3
//
//  Created by Aaron R on 27/08/21.
//  Devise a program to create a list of integers dynamically .Write functions (i) Find
//  he maximum and minimum from the list. (iii) Display the contents of the list.

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

struct node
{
    int data;
    struct node * link;
};

void display (struct node *temp)
{
    // Function to display the data in the linked list

    // If head is NULL, the linked list is empty
    if (temp==NULL)
        printf("The linked list is empty\n");
    else
    // Else, display it's elements
        while ((temp)!=NULL)
        {
            printf("%d\n",(temp)->data);
            temp = (temp)->link;
        }
}

void insert_head(struct node ** current, struct node ** head,int * count)
{
    // Function to insert node at the head
    *current = (struct node *) malloc(sizeof(struct node));

    // Input the node's data
    printf("Enter the data: ");
    scanf("%d",&(*current)->data);

    // The address part of the new node points to the previous head
    (*current)->link = *head;

    // The head variable is updated to the current one
    *head = *current;

    *count+=1;
}

```

```c
void insert_end(struct node ** current, struct node ** head,int * count)
{
    // Function to insert node at the end
    *current = (struct node *) malloc(sizeof(struct node));
    // Input the node's data
    printf("Enter the data: ");
    scanf("%d",&(*current)->data);

    // The link of the new node is NULL
    (*current)->link = NULL;

    // A temporary node is created
    struct node * temp;
    temp = *head;

    // It is used to traverse to the end of the existing linked list
    while (temp->link!=NULL)
    {
        temp = temp->link;
    }

    // The link of the node at the end of the existing linked list is updated to the new node
    temp->link = *current;

    *count+=1;
}

void insert_position(struct node ** current, struct node ** head,int * count,int p)
{
    // Function to insert node at the head
    *current = (struct node *) malloc(sizeof(struct node));

    // Input the node's data
    printf("Enter the data: ");
    scanf("%d",&(*current)->data);

    // A temporary node is created
    struct node * temp;

    // The temporary node is used to traverse and get to the pth position
    temp = *head;
    int i=1;
    while (i<p)
    {
        temp = temp->link;
        i+=1;
    }

    // The link of the new node is updated to the link of the node at (p+1)th position
    (*current)->link = temp->link;

    // The link of the node at the pth position is updated to the new node
    temp->link = (*current);

    *count+=1;
}

void MinMax (struct node *temp)
{
    // Function to display the min and max in the linked list

    int min = temp->data;
    int max = temp->data;

    // If head is NULL, the linked list is empty
    if (temp==NULL)
        printf("The linked list is empty\n");
```

```
    else
    // Else, compute and display it's min and max
    {
        while ((temp)!=NULL)
        {
            if (temp->data>max)
                max = temp->data;
            if (temp->data<min)
                min = temp->data;

            temp = (temp)->link;
        }
        printf("\nMin element: %d\nMax element: %d\n",min,max);
    }
}


int main(int argc, const char * argv[])
{
    // Initialising the linked list
    printf("Initialising a linked list...\n");
    struct node *head, *current;
    int count=0;
    head = NULL;
    current = NULL;

    // Displaying the initial elements of the linked list
    printf("Elements of the linked list:\n");
    display(head);

    int c=1;
    while (c==1)
    {
        int p;
        printf("\n~##### Insertion of data into linked list #####~\n");
        printf("1) 0 to insert a new node at the head\n2) %d or -1 to insert a new node at the end\nThere are
            currently %d nodes in the linked list\nEnter the position: ",count,count);
        scanf("%d",&p);

        if (p==0)
            insert_head(&current,&head,&count);

        else if (p==count||p==-1)
            insert_end(&current,&head,&count);

        else if((p>=0)&&(p<=count))
            insert_position(&current, &head, &count,p);

        else
            printf("Invalid position\n");

        printf("Elements of the linked list:\n");
        display(head);

        printf("To insert another node, press 1: ");
        scanf("%d",&c);

    }
    printf("\n---##############################---\n            ~End.\n\n");
    printf("Final linked list:\n");
    display(head);
    MinMax(head);
    return 0;
}
```

Output(s):

```
Initialising a linked list...
Elements of the linked list:
The linked list is empty

~##### Insertion of data into linked list #####~
1) 0 to insert a new node at the head
2) 0 or -1 to insert a new node at the end
There are currently 0 nodes in the linked list
Enter the position: 0
Enter the data: 10
Elements of the linked list:
10
To insert another node, press 1: 1

~##### Insertion of data into linked list #####~
1) 0 to insert a new node at the head
2) 1 or -1 to insert a new node at the end
There are currently 1 nodes in the linked list
Enter the position: 0
Enter the data: 20
Elements of the linked list:
20
10
To insert another node, press 1: 1

~##### Insertion of data into linked list #####~
1) 0 to insert a new node at the head
2) 2 or -1 to insert a new node at the end
There are currently 2 nodes in the linked list
Enter the position: 30
Invalid position
Elements of the linked list:
20
10
To insert another node, press 1: 1

~##### Insertion of data into linked list #####~
1) 0 to insert a new node at the head
2) 2 or -1 to insert a new node at the end
There are currently 2 nodes in the linked list
Enter the position: -1
Enter the data: 70
Elements of the linked list:
20
10
70
To insert another node, press 1: 1

~##### Insertion of data into linked list #####~
1) 0 to insert a new node at the head
2) 3 or -1 to insert a new node at the end
There are currently 3 nodes in the linked list
Enter the position: 2
Enter the data: 100
Elements of the linked list:
20
10
100
70
To insert another node, press 1: 2

---##############################---
            ~End.

Final linked list:
20
10
100
70

Min element: 10
Max element: 100
Program ended with exit code: 0
```

All Output Filter

4. Write a program to convert any given Infix expression to a postfix form.
Example:
Infix: A * (B + C) * D
Postfix: A B C + * D*

Source Code

```c
//
//  main.c
//  Lab-3-4 (infix to postfix)
//
//  Created by Aaron R on 03/09/21.
//

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

struct node
{
    int data;
    struct node * link;
};

struct node * stack;

void push(char c)
{
    // ##### Function to insert node at the head #####
    struct node * current;
    current = (struct node *) malloc(sizeof(struct node));

    // Input the node's data
    current->data = c;

    // The address part of the new node points to the previous head
    current->link = stack;

    // The head variable is updated to the current one
    stack = current;
}

char pop(void)
{
    // #### Function to delete the node at the beginning ####
    struct node * temp;
    if (stack==NULL)
    {
        printf("The stack is empty. No elements to pop.\n");
        exit(0);
    }
    else
    {
        temp=stack;
        char item = stack->data;
        // The address of the head is updated to the next node
        stack = stack->link;

        // free the memory of the previous head
        free(temp);

        return item;
    }

}
```

```c
char peek(void)
{
    // Function to return the top value of the stack
    if (stack == NULL)
        return 0;

    else
        return stack->data;

}

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

int main()
{
    char exp[100];
    char *temp, x;
    printf("Enter infix expression : ");
    scanf("%[^\n]", exp);
    printf("\n");

    temp = exp;
    printf("Postfix expression:\n");
    while (*temp != '\0')
    {
        if (isalnum(*temp))
            printf("%c", *temp);
        else if (*temp == '(')
            push(*temp);
        else if (*temp == ')')
        {
            while ((x = pop()) != '(')
                printf("%c", x);
        }
        else
        {
            while (check(peek()) >= check(*temp))
                printf("%c", pop());
            push(*temp);
        }
        temp+=1;
    }

    while (stack != NULL)
    {
        printf("%c", pop());
    }
    printf("\n############\n");
    return 0;
}
```

Output(s):

```
Enter infix expression : A*(B+C)*D

Postfix expression:
ABC+*D*
###########
Program ended with exit code: 0
```

All Output Filter

5. Evaluate the given postfix expression A B C + * D* using stacks. Assume your values for A, B, C, and D. Implement the same using C

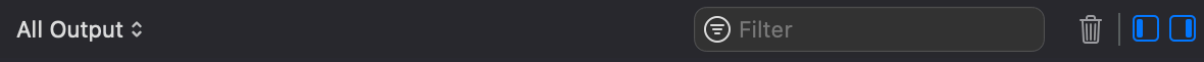
```c
//
//  main.c
//  Lab-3-5 (postfix eval)
//
//  Created by Aaron R on 03/09/21.
//  Evaluate the given postfix expression A B C + * D* using stacks.
//  Assume your values for A, B, C, and D. Implement the same using C

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

char variables[100];
int values[100];
int variable_count=0;

struct node
{
    int data;
    struct node * link;
};

struct node * top;

void push(char c)
{
    // ##### Function to insert node at the head #####
    struct node * current;
    current = (struct node *) malloc(sizeof(struct node));

    // Input the node's data
    current->data = c;

    // The address part of the new node points to the previous head
    current->link = top;

    // The head variable is updated to the current one
    top = current;
}

int pop(void)
{
    // #### Function to delete the node at the beginning ####
    struct node * temp;
    if (top==NULL)
    {
        printf("The stack is empty. No elements to pop.\n");
        exit(0);
    }
    else
    {
        temp=top;
        int item = top->data;
        // The address of the head is updated to the next node
        top = top->link;

        // free the memory of the previous head
        free(temp);

        return item;
    }

}
```

```
65
66
67  int eval(char input[])
68  {
69      int result=0;
70      for (int i=0; i<strlen(input); i++)
71      {
72          if (isalpha(input[i]))
73          {
74              // If operand is encountered, push it into stack
75              for (int j=0; j<variable_count; j++)
76              {
77                  if (input[i]==variables[j])
78                  {
79                      push(values[j]);
80                      break;
81                  }
82              }
83          }
84          else
85          {
86              // If operator is encountered
87              // Pop 2 operands from stack
88
89              int a = pop();
90              int b = pop();
91
92              // Result = temp2 <operator> temp1
93              if (input[i]=='+')
94                  result = b + a;
95              else if (input[i]=='-')
96                  result = b - a;
97              else if (input[i]=='*')
98                  result = b * a;
99              else
100                 result = b/a;
101
102             push(result);
103         }
104     }
105     return result;
106 }
107
108 void rmdup(char *array, int length)
109 {
110     char *current , *end = array + length - 1;
111
112     for ( current = array + 1; array < end; array++, current = array + 1 )
113     {
114         while ( current <= end )
115         {
116             if ( *current == *array )
117             {
118                 *current = *end--;
119                 variable_count-=1;
120             }
121             else
122             {
123                 current++;
124             }
125         }
126     }
127 }
```

```c
int main(int argc, const char * argv[])
{
    char input[100];
    printf("Enter postfix expression to be evaluated: ");
    scanf("%[^\n]",input);


    // Make a list of variables used
    for (int i=0; i<strlen(input); i++)
    {
        if (isalpha(input[i]))
        {
            variables[variable_count]=input[i];
            variable_count++;
        }
    }

    // Remove duplicates in variable set, if any
    rmdup(variables,variable_count);

    // Get the values of the variables
    for (int i=0; i<variable_count; i++)
    {
        printf("Enter the value of %c: ",variables[i]);
        scanf("%d",&values[i]);
    }

    // Evaluate expression
    printf("Answer: %d\n",eval(input));
    return 0;
}
```

Output(s):

```
Enter postfix expression to be evaluated: ABC+*D*
Enter the value of A: 5
Enter the value of B: 2
Enter the value of C: 9
Enter the value of D: 1
Answer: 55
Program ended with exit code: 0
```



6. Write a program to convert any given Infix expression to a prefix form. Example: Infix: A * (B + C) * D Prefix: **A+BCD

Source Code

```c
//
//  main.c
//  Lab-3-6
//
//  Created by Aaron R on 03/09/21.
//  Write a program to convert any given Infix expression to a prefix form.
//  Example:
//  Infix: A * (B + C) * D
//  Prefix: **A+BCD

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

int top = -1;
void push(char x,char stack[])
{

    if (top==100-1)
        printf("Overflow\n");
    else
    {
        top++;
        stack[top] = x;
    }
}

char pop(char stack[])
{
    char item;
    if (top==-1) {
        printf("Underflow\n");
        exit(0);
    }
    else
    {
        item = stack[top];
        top--;
        printf("popped: %c\n",item);
        return item;
    }

}

int precedence(char op)
{
    //To check the precedence

    char operators[7] = {'^', '*', '/', '+', '-'};
    int p=0;
    for(int i = 0; i<7; i++)
    {
        if(op == operators[i])
        {
            p = i+1;
        }
    }

    if (p==1)
        return 3;
    else if (p==2||p==3)
        return 2;
    else if (p==4||p==5)
        return 1;
    return 0;
}
```

```
67
68  void reverse(char exp[], char reverse[])
69  {
70      // To reverse string
71
72      int revExpLen = 0;
73      for(int i=(int)strlen(exp)-1; i>=0; i--)
74      {
75          if(exp[i]=='(')
76          {
77              reverse[revExpLen] = ')';
78              reverse++;
79          }
80          else if(exp[i]==')')
81          {
82              reverse[revExpLen] = '(';
83              revExpLen++;
84          }
85          else
86          {
87              reverse[revExpLen] = exp[i];
88              revExpLen+=1;
89          }
90      }
91  }
92
93  void InfixToPrefix(char string[], char stack[], char prefix[], char temp1[], char result[])
94  {
95
96      // To convert infix to prefix
97
98      reverse(string, temp1);
99
100     int prefixLen = 0;
101     int top = 0;
102     for(int i = 0; i<strlen(temp1); i++)
103     {
104
105         if(((int)temp1[i]>=65 && (int)temp1[i]<=122) || ((int)temp1[i]>=48 && (int)temp1[i]<=57))
106         {
107             prefix[prefixLen] = temp1[i];
108             prefixLen += 1;
109         }
110         else
111         {
112             if(temp1[i] == '(')
113             {
114                 top+=1;
115                 stack[top-1] = temp1[i];
116                 stack[top] = '\0';
117             }
118             else{
119                 if(top != 0 && temp1[i]!=')' && temp1[i]!='(' && stack[top-1]!='('){
120                                         if(precedence(temp1[i])>precedence(stack[top-1]))
121                     {
122                         top++;
123                         stack[top-1] = temp1[i];
124                     }
125
126                     else if (precedence(temp1[i])<=precedence(stack[top-1]))
127                     {
128
129                         while((precedence(stack[top-1])>=precedence(temp1[i]) ||
                                temp1[i]!='(')&&top>0&&stack[top-1]!=temp1[i])
130                         {
131                             prefixLen += 1;
132                             prefix[prefixLen-1] = stack[top-1];
133                             stack[top-1] = '\0';
134                             top -= 1;
135                         }
```

```
                        stack[top] = temp1[i];
                    }
                }
                else if(top == 0 && temp1[i]!=')'){
                    stack[top] = temp1[i];
                    top+=1;
                }


                else if(top != 0 && temp1[i]==')'){
                    while(stack[top-1]!='('){
                        prefixLen+=1;
                        prefix[prefixLen-1] = stack[top-1];
                        stack[top-1] = '\0';
                        top-=1;
                    }
                    stack[top-1] = '\0';
                    top-=1;
                }
                else{
                    top+=1;
                    stack[top-1] = temp1[i];
                }
            }
        }
    }

    if(top!=0)
    {
        while(top>=0)
        {
            prefixLen += 1;
            prefix[prefixLen-1] = stack[top];
            stack[top] = '\0';
            top -= 1;
        }
    }
    else
    {
        prefixLen += 1;
        prefix[prefixLen-1] = stack[top];
        stack[top] = '\0';
        top -= 1;
    }

    reverse(prefix, result);
}

int main()
{
    char string[50], stack[50], prefix[50], temp1[50], result[50];
    printf("Enter the infix expression: ");
    scanf("%[^\n]", string);

    InfixToPrefix(string, stack, prefix, temp1, result);
    printf("Prefix Expression: %s\n", result);
}
```

Output(s):

```
Enter infix expression: A*(B+C)*D
popped: *
popped: (
popped: +
popped: )
popped: *
Prefix Expression: *)+(*ABCD
Program ended with exit code: 0
```

All Output ⌃ Filter

7. Evaluate the given prefix expression **A+BCD using stacks. Assume your values for A, B, C, and D. Implement the same using C

Source Code

```c
//
//  main.c
//  Lab-3-7 (prefix eval)
//
//  Created by Aaron R on 03/09/21.

//  Evaluate the given prefix expression **A+BCD using stacks. Assume your values
//  for A, B, C, and D. Implement the same using C.


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

char variables[100];
int values[100];
int variable_count=0;

struct node
{
    int data;
    struct node * link;
};

struct node * top;


void strrev(char *str)
{
    // Function to reverse string

    /* skip null */
    if (str == 0)
    {
        return;
    }

    /* skip empty string */
    if (*str == 0)
    {
        return;
    }

    /* get range */
    char *start = str;
    char *end = start + strlen(str) - 1; /* -1 for \0 */
     char temp;

     /* reverse */
     while (end > start)
     {
         /* swap */
         temp = *start;
         *start = *end;
         *end = temp;

         /* move */
         ++start;
         --end;
     }
 }

```

```c

void push(char c)
{
    // ##### Function to insert node at the head #####
    struct node * current;
    current = (struct node *) malloc(sizeof(struct node));

    // Input the node's data
    current->data = c;

    // The address part of the new node points to the previous head
    current->link = top;

    // The head variable is updated to the current one
    top = current;
}

int pop(void)
{
    // #### Function to delete the node at the beginning ####
    struct node * temp;
    if (top==NULL)
    {
        printf("The stack is empty. No elements to pop.\n");
        exit(0);
    }
    else
    {
        temp=top;
        int item = top->data;
        // The address of the head is updated to the next node
        top = top->link;

        // free the memory of the previous head
        free(temp);

        return item;
    }

}


int eval(char input[])
{
    int result=0;
    for (int i=0; i<strlen(input); i++)
    {
        if (isalpha(input[i]))
        {
            // If operand is encountered, push it into stack
            for (int j=0; j<variable_count; j++)
            {
                if (input[i]==variables[j])
                {
                    push(values[j]);
                    break;
                }
            }
        }
```

```
        else
        {
            // If operator is encountered
            // Pop 2 operands from stack

            int a = pop();
            int b = pop();

            // Result = temp2 <operator> temp1
            if (input[i]=='+')
                result = b + a;
            else if (input[i]=='-')
                result = b - a;
            else if (input[i]=='*')
                result = b * a;
            else
                result = b/a;

            push(result);
        }
    }
    return result;
}


void rmdup(char *array, int length)
{
    char *current , *end = array + length - 1;

    for ( current = array + 1; array < end; array++, current = array + 1 )
    {
        while ( current <= end )
        {
            if ( *current == *array )
            {
                *current = *end--;
                variable_count-=1;
            }
            else
            {
                current++;
            }
        }
    }
int main(int argc, const char * argv[])
{
    char input[100];
    printf("Enter prefix expression to be evaluated: ");
    scanf("%[^\n]",input);

    strrev(input);

    // Make a list of variables used
    for (int i=0; i<strlen(input); i++)
    {
        if (isalpha(input[i]))
        {
            variables[variable_count]=input[i];
            variable_count++;
        }
    }
```

```
184
185     // Remove duplicates in variable set, if any
186     rmdup(variables,variable_count);
187
188     // Get the values of the variables
189     for (int i=0; i<variable_count; i++)
190     {
191         printf("Enter the value of %c: ",variables[i]);
192         scanf("%d",&values[i]);
193     }
194
195     // Evaluate expression
196     printf("Answer: %d\n",eval(input));
197     return 0;
198 }
```

Output(s):

```
Enter prefix expression to be evaluated: **A+BCD
Enter the value of D: 4
Enter the value of C: 8
Enter the value of B: 1
Enter the value of A: 6
Answer: 216
Program ended with exit code: 0
```