

Doubly linked lists and stacks

1. Reverse a doubly linked list and print the elements

Source code

```
1 //  
2 //  main.c  
3 //  Lab-3-1  
4 //  
5 //  Created by Aaron R on 27/08/21.  
6 //  Reverse a doubly linked list and print the elements  
7  
8 #include <stdio.h>  
9 #include <stdlib.h>  
10 struct node  
11 {  
12     int data;  
13     struct node *next;  
14     struct node *prev;  
15 };  
16 struct node *head, *tail;  
17 int length=0;  
18  
19 void display(void)  
20 {  
21     // ##### To display the elements of the doubly linked list #####  
22     struct node * temp;  
23     temp = head;  
24     if (head==NULL)  
25     {  
26         printf("The doubly linked list is empty\n");  
27     }  
28  
29     else  
30     {  
31         // traversing from the first node  
32         while (temp->next!=NULL)  
33         {  
34             printf("%d\n",temp->data);  
35             temp = temp->next;  
36         }  
37         // printing the last node's data  
38         printf("%d\n",temp->data);  
39     }  
40 }  
41 void initList(void)  
42 {  
43     // ##### To initialise a doubly linked list with n number of elements #####  
44     struct node *new;  
45     printf("Enter the number of nodes with which to initialise the doubly linked list: ");  
46     int n;  
47     scanf("%d",&n);  
48     length = n;  
49     for (int i=0; i<n; i++)  
50     {  
51         new = (struct node *)malloc(sizeof(struct node));  
52         printf("Enter data of node %d: ",i+1);  
53         scanf("%d",&new->data);  
54         new->next = NULL;  
55         new->prev = NULL;  
56         if (head == NULL)  
57         {  
58             // If there are no elements in the doubly linked list  
59             tail = new;  
60             head = tail;  
61         }
```

```
62     else
63     {
64         // If there are nodes in the doubly linked list
65         tail->next = new;
66         new->prev = tail;
67         tail = new;
68     }
69 }
70 }
71
72 void insert_head(void)
73 {
74     // ##### To insert a node at the head #####
75     struct node *new;
76     new = (struct node *)malloc(sizeof(struct node));
77     printf("Enter data of new node: ");
78     scanf("%d",&new->data);
79     new->next = NULL;
80     new->prev = NULL;
81     if (head==NULL)
82     {
83         // If there are no nodes in the doubly linked list
84         head = new;
85         tail = new;
86     }
87     else
88     {
89         // If there are nodes in the doubly linked list
90         // Update the previous pointer of the head node from NULL to the new node
91         head->prev = new;
92
93         // Update the next pointer of the new node to the previous head node
94         new->next = head;
95
96         // Update the head pointer to the new node
97         head = new;
98     }
99     length+=1;
100 }
101
102 void insert_end(void)
103 {
104     // ##### To insert a node at the tail #####
105     struct node *new;
106     new = (struct node *)malloc(sizeof(struct node));
107     printf("Enter data of new node: ");
108     scanf("%d",&new->data);
109     new->next = NULL;
110     new->prev = NULL;
111     if (head==NULL)
112     {
113         // If there are no nodes in the doubly linked list
114         head = new;
115         tail = new;
116     }
117     else
118     {
119         // If there are nodes in the doubly linked list
120         // Update the next pointer of the tail node from NULL to the new node
121         tail->next = new;
122
123         // Update the previous pointer of the new node to the previous tail node
124         new->prev = tail;
125
126         // Update the tail pointer to the new node
127         tail = new;
128     }
129     length+=1;
130 }
131
```

```
132 void insert_pos(int p)
133 {
134     // ##### To insert a node at the pth position #####
135     struct node *new,*temp;
136     new = (struct node *)malloc(sizeof(struct node));
137     temp = head;
138     printf("Enter data of new node: ");
139     scanf("%d",&new->data);
140     new->next = NULL;
141     new->prev = NULL;
142     int i=1;
143     while (i<p-1)
144     {
145         temp = temp->next;
146         i++;
147     }
148     // Updating the prev pointer to p-1th node
149     new->prev = temp;
150
151     // updating the next node to p1th node
152     new->next = temp->next;
153
154     // Updating the next pointer of p-1th node to the new node
155     temp->next = new;
156
157     // Updating the previous pointer of the pth node to the new node
158     new->next->prev = new;
159
160     length+=1;
161 }
162
163 void reverse_list(void)
164 {
165     // ##### To reverse a doubly linked list #####
166
167     struct node *current_node, *next_node;
168     current_node = head;
169     while (current_node!=NULL)
170     {
171         // Storing the next node in next_node
172         next_node = current_node->next;
173
174         // Swapping the prev and next pointers of the current_node
175         current_node->next = current_node->prev;
176         current_node->prev = next_node;
177
178         // Updating current_node to the next node
179         current_node = next_node;
180     }
181
182     // Swapping head and tail pointers;
183     current_node = head;
184     head = tail;
185     tail = current_node;
186 }
187
188 int main(int argc, const char * argv[])
189 {
190     head = NULL;
191     // Initialising doubly linked list
192     initList();
193     display();
194
195     int more_nodes = 0;
196     printf("Would you like to insert more nodes?\n1 - Yes\n");
197     scanf("%d",&more_nodes);
```

```
199     {
200         while (more_nodes==1)
201     {
202         printf("####Positions####\n0 : beginning\n%d or -1 : end\nBetween 0 and %d : position p\nEnter the
203             position to insert the new node: ",length,length);
204         int p;
205         scanf("%d",&p);
206         if (p==0)
207             insert_head();
208         else if (p==-1 || p==length)
209             insert_end();
210         else if (p>1 && p<length)
211             insert_pos(p);
212         else
213             printf("Invalid position");
214
215         printf("Current elements of doubly linked lists: \n");
216         display();
217
218         printf("Would you like to insert another node?\n1 - Yes\n");
219         scanf("%d",&more_nodes);
220     }
221
222     printf("Reversed doubly linked list:\n");
223     // Reversing the doubly linked list
224     reverse_list();
225     display();
226
227     return 0;
228 }
```

```
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

```
1 //  
2 // main.c  
3 // Lab-3-2  
4 //  
5 // Created by Aaron R on 27/08/21.  
6  
7 // Create a doubly linked list and  
8 // delete a node at the beginning, end and any position in between.  
9  
10 #include <stdio.h>  
11 #include <stdlib.h>  
12  
13 struct node  
14 {  
15     int data;  
16     struct node *next;  
17     struct node *prev;  
18 };  
19 struct node *head, *tail;  
20 int length=0;  
21  
22 void display(void)  
23 {  
24     // ##### To display the elements of the doubly linked list #####  
25     struct node * temp;  
26     temp = head;  
27     if (head==NULL)  
28     {  
29         printf("The doubly linked list is empty\n");  
30     }  
31  
32     else  
33     {  
34         // traversing from the first node  
35         while (temp->next!=NULL)  
36         {  
37             printf("%d\n",temp->data);  
38             temp = temp->next;  
39         }  
40         // printing the last node's data  
41         printf("%d\n",temp->data);  
42     }  
43 }
```

```
44 void initList(void)
45 {
46     // ##### To initialise a doubly linked list with n number of elements #####
47     struct node *new;
48     printf("Enter the number of nodes with which to initialise the doubly linked list: ");
49     int n;
50     scanf("%d",&n);
51     length = n;
52     for (int i=0; i<n; i++)
53     {
54         new = (struct node *)malloc(sizeof(struct node));
55         printf("Enter data of node %d: ",i+1);
56         scanf("%d",&new->data);
57         new->next = NULL;
58         new->prev = NULL;
59         if (head == NULL)
60         {
61             // If there are no elements in the doubly linked list
62             tail = new;
63             head = tail;
64         }
65         else
66         {
67             // If there are nodes in the doubly linked list
68             tail->next = new;
69             new->prev = tail;
70             tail = new;
71         }
72     }
73 }
74
75 void delete_head(void)
76 {
77     // ##### To delete a node whose position is the head #####
78
79     struct node * temp;
80     temp = head;
81     if (head == tail)
82     {
83         head = tail = NULL;
84         length-=1;
85     }
86     else
87     {
88         // Updating head to the next element
89         head = head->next;
90
91         // Updating the prev pointer of head is NULL
92         head->prev = NULL;
93         free(temp);
94         length-=1;
95     }
96 }
97
98
99 void delete_end(void)
100 {
101     // ##### To delete a node whose position is the tail #####
102     struct node * temp;
103     temp = tail;
104     if (head == tail)
105     {
106         head = tail = NULL;
107         length-=1;
108     }
109 }
```

```

110     else
111     {
112         // Updating the tail pointer to the previous element
113         tail = tail->prev;
114
115         // The new tail pointer's next pointer is updated to NULL
116         tail->next = NULL;
117         free(temp);
118         length-=1;
119     }
120
121 }
122
123 void delete_pos(int p)
124 {
125     // ##### To delete a node whose position is p #####
126
127     struct node * temp;
128     temp = head;
129     int i=1;
130
131     // Traversing to get to the pth node
132     while (i<p)
133     {
134         temp = temp->next;
135         i++;
136     }
137     // Updating the p-1th node's next poitner to p+1th node
138     temp->prev->next = temp->next;
139
140     // Updating the p+1th node's prev pointer to p-1th node
141     temp->next->prev = temp->prev;
142     free(temp);
143     length-=1;
144 }
145
146 int main(int argc, const char * argv[])
147 {
148     head = NULL;
149     tail = NULL;
150     // Initialising doubly linked list
151     initList();
152     display();
153
154     int cond;
155     printf("Would you like to delete nodes?\n1 - Yes\n");
156     scanf("%d",&cond);
157     while (cond==1)
158     {
159         if (length==0)
160         {
161             printf("There are no more nodes left to delete\n");
162             break;
163         }
164         else
165         {
166             printf("##### Positions to delete #####\n1 - head\n-1 or %d - tail\nBetween 1 and %d - pth
167                 element\nEnter the position to delete: ",length,length);
168             int p;
169             scanf("%d",&p);
170             if (p==1)
171                 delete_head();
172             else if(p== -1 || p==length)
173                 delete_end();
174             else if(p>1 && p<length)
175                 delete_pos(p);
176             else
177                 printf("Invalid position\n");

```

```
177         printf("Current elements in doubly linked list: \n");
178         display();
179         printf("Enter 1 to delete more nodes: ");
180         scanf("%d",&cond);
181     }
182 }
183 }
184
185 return 0;
186 }
```

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

```
1 //  
2 //  main.c  
3 //  Lab-3-3  
4 //  
5 //  Created by Aaron R on 27/08/21.  
6 //  Devise a program to create a list of integers dynamically .Write functions (i) Find  
7 //  he maximum and minimum from the list. (iii) Display the contents of the list.  
8  
9 #include <stdio.h>  
10 #include <stdlib.h>  
11  
12 struct node  
13 {  
14     int data;  
15     struct node * link;  
16 };  
17  
18 void display (struct node *temp)  
19 {  
20     // Function to display the data in the linked list  
21  
22     // If head is NULL, the linked list is empty  
23     if (temp==NULL)  
24         printf("The linked list is empty\n");  
25     else  
26         // Else, display it's elements  
27         while ((temp)!=NULL)  
28         {  
29             printf("%d\n", (temp)->data);  
30             temp = (temp)->link;  
31         }  
32 }  
33  
34 void insert_head(struct node ** current, struct node ** head,int * count)  
35 {  
36     // Function to insert node at the head  
37     *current = (struct node *) malloc(sizeof(struct node));  
38  
39     // Input the node's data  
40     printf("Enter the data: ");  
41     scanf("%d",&(*current)->data);  
42  
43     // The address part of the new node points to the previous head  
44     (*current)->link = *head;  
45  
46     // The head variable is updated to the current one  
47     *head = *current;  
48  
49     *count+=1;  
50 }  
51
```

```
52 void insert_end(struct node ** current, struct node ** head,int * count)
53 {
54     // Function to insert node at the end
55     *current = (struct node *) malloc(sizeof(struct node));
56     // Input the node's data
57     printf("Enter the data: ");
58     scanf("%d",&(*current)->data);
59
60     // The link of the new node is NULL
61     (*current)->link = NULL;
62
63     // A temporary node is created
64     struct node * temp;
65     temp = *head;
66
67     // It is used to traverse to the end of the existing linked list
68     while (temp->link!=NULL)
69     {
70         temp = temp->link;
71     }
72
73     // The link of the node at the end of the existing linked list is updated to the new node
74     temp->link = *current;
75
76     *count+=1;
77 }
78
79 void insert_position(struct node ** current, struct node ** head,int * count,int p)
80 {
81     // Function to insert node at the head
82     *current = (struct node *) malloc(sizeof(struct node));
83
84     // Input the node's data
85     printf("Enter the data: ");
86     scanf("%d",&(*current)->data);
87
88     // A temporary node is created
89     struct node * temp;
90
91     // The temporary node is used to traverse and get to the pth position
92     temp = *head;
93     int i=1;
94     while (i<p)
95     {
96         temp = temp->link;
97         i+=1;
98     }
99
100    // The link of the new node is updated to the link of the node at (p+1)th position
101    (*current)->link = temp->link;
102
103    // The link of the node at the pth position is updated to the new node
104    temp->link = (*current);
105
106    *count+=1;
107 }
108
109 void MinMax (struct node *temp)
110 {
111     // Function to display the min and max in the linked list
112
113     int min = temp->data;
114     int max = temp->data;
115
116     // If head is NULL, the linked list is empty
117     if (temp==NULL)
118         printf("The linked list is empty\n");
```

```

119     else
120     // Else, compute and display it's min and max
121     {
122         while ((temp)!=NULL)
123         {
124             if (temp->data>max)
125                 max = temp->data;
126             if (temp->data<min)
127                 min = temp->data;
128
129             temp = (temp)->link;
130         }
131         printf("\nMin element: %d\nMax element: %d\n",min,max);
132     }
133 }
134
135
136 int main(int argc, const char * argv[])
137 {
138     // Initialising the linked list
139     printf("Initialising a linked list...\n");
140     struct node *head, *current;
141     int count=0;
142     head = NULL;
143     current = NULL;
144
145     // Displaying the initial elements of the linked list
146     printf("Elements of the linked list:\n");
147     display(head);
148
149     int c=1;
150     while (c==1)
151     {
152         int p;
153         printf("\n~##### Insertion of data into linked list #####~\n");
154         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
155             currently %d nodes in the linked list\nEnter the position: ",count,count);
156         scanf("%d",&p);
157
158         if (p==0)
159             insert_head(&current,&head,&count);
160
161         else if (p==count||p==-1)
162             insert_end(&current,&head,&count);
163
164         else if((p>0)&&(p<=count))
165             insert_position(&current, &head, &count,p);
166
167         else
168             printf("Invalid position\n");
169
170         printf("Elements of the linked list:\n");
171         display(head);
172
173         printf("To insert another node, press 1: ");
174         scanf("%d",&c);
175     }
176     printf("\n---#####---\n");                                ~End.\n\n");
177     printf("Final linked list:\n");
178     display(head);
179     MinMax(head);
180
181     return 0;
182 }
```

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 

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

```
1 //  
2 //  main.c  
3 //  Lab-3-4 (infix to postfix)  
4 //  
5 //  Created by Aaron R on 03/09/21.  
6 //  
7  
8 #include <stdio.h>  
9 #include <ctype.h>  
10 #include <stdlib.h>  
11  
12 struct node  
13 {  
14     int data;  
15     struct node * link;  
16 };  
17  
18 struct node * stack;  
19  
20 void push(char c)  
21 {  
22     // ##### Function to insert node at the head #####  
23     struct node * current;  
24     current = (struct node *) malloc(sizeof(struct node));  
25  
26     // Input the node's data  
27     current->data = c;  
28  
29     // The address part of the new node points to the previous head  
30     current->link = stack;  
31  
32     // The head variable is updated to the current one  
33     stack = current;  
34 }  
35  
36 char pop(void)  
37 {  
38     // ##### Function to delete the node at the beginning #####  
39     struct node * temp;  
40     if (stack==NULL)  
41     {  
42         printf("The stack is empty. No elements to pop.\n");  
43         exit(0);  
44     }  
45     else  
46     {  
47         temp=stack;  
48         char item = stack->data;  
49         // The address of the head is updated to the next node  
50         stack = stack->link;  
51  
52         // free the memory of the previous head  
53         free(temp);  
54  
55         return item;  
56     }  
57  
58 }
```

```
59 char peek(void)
60 {
61     // Function to return the top value of the stack
62     if (stack == NULL)
63         return 0;
64
65     else
66         return stack->data;
67
68 }
69
70 int check(char x)
71 {
72     // To check precedence
73     if (x == '(')
74         return 0;
75     else if (x == '+' || x == '-')
76         return 1;
77     else if (x == '*' || x == '/')
78         return 2;
79     return -1;
80 }
81
82 int main()
83 {
84     char exp[100];
85     char *temp, x;
86     printf("Enter infix expression : ");
87     scanf("%[^\\n]", exp);
88     printf("\n");
89
90     temp = exp;
91     printf("Postfix expression:\n");
92     while (*temp != '\\0')
93     {
94         if (isalnum(*temp))
95             printf("%c", *temp);
96         else if (*temp == '(')
97             push(*temp);
98         else if (*temp == ')')
99         {
100             while ((x = pop()) != '(')
101                 printf("%c", x);
102         }
103     else
104     {
105         while (check(peek()) >= check(*temp))
106             printf("%c", pop());
107         push(*temp);
108     }
109     temp+=1;
110 }
111
112 while (stack != NULL)
113 {
114     printf("%c", pop());
115 }
116 printf("\n#####\n");
117 return 0;
118 }
```

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

```
1 //  
2 //  main.c  
3 //  Lab-3-5 (postfix eval)  
4 //  
5 //  Created by Aaron R on 03/09/21.  
6 //  Evaluate the given postfix expression A B C + * D* using stacks.  
7 //  Assume your values for A, B, C, and D. Implement the same using C  
8  
9 #include <stdio.h>  
10 #include <stdlib.h>  
11 #include <string.h>  
12 #include <ctype.h>  
13  
14 char variables[100];  
15 int values[100];  
16 int variable_count=0;  
17  
18 struct node  
19 {  
20     int data;  
21     struct node * link;  
22 };  
23  
24 struct node * top;  
25  
26 void push(char c)  
27 {  
28     // ##### Function to insert node at the head #####  
29     struct node * current;  
30     current = (struct node *) malloc(sizeof(struct node));  
31  
32     // Input the node's data  
33     current->data = c;  
34  
35     // The address part of the new node points to the previous head  
36     current->link = top;  
37  
38     // The head variable is updated to the current one  
39     top = current;  
40 }  
41  
42 int pop(void)  
43 {  
44     // ##### Function to delete the node at the beginning #####  
45     struct node * temp;  
46     if (top==NULL)  
47     {  
48         printf("The stack is empty. No elements to pop.\n");  
49         exit(0);  
50     }  
51     else  
52     {  
53         temp=top;  
54         int item = top->data;  
55         // The address of the head is updated to the next node  
56         top = top->link;  
57  
58         // free the memory of the previous head  
59         free(temp);  
60  
61         return item;  
62     }  
63 }  
64 }
```

```
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 }
```

```
128 int main(int argc, const char * argv[])
129 {
130     char input[100];
131     printf("Enter postfix expression to be evaluated: ");
132     scanf("%[^\\n]",input);
133
134
135     // Make a list of variables used
136     for (int i=0; i<strlen(input); i++)
137     {
138         if (isalpha(input[i]))
139         {
140             variables[variable_count]=input[i];
141             variable_count++;
142         }
143     }
144
145     // Remove duplicates in variable set, if any
146     rmdup(variables,variable_count);
147
148     // Get the values of the variables
149     for (int i=0; i<variable_count; i++)
150     {
151         printf("Enter the value of %c: ",variables[i]);
152         scanf("%d",&values[i]);
153     }
154
155     // Evaluate expression
156     printf("Answer: %d\\n",eval(input));
157     return 0;
158 }
```

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
```

All Output 

 Filter

 |  

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

Source Code

```
1 //  
2 // main.c  
3 // Lab-3-6  
4 //  
5 // Created by Aaron R on 03/09/21.  
6 // Write a program to convert any given Infix expression to a prefix form.  
7 // Example:  
8 // Infix: A * (B + C) * D  
9 // Prefix: **A+B+CD  
10  
11 #include<stdio.h>  
12 #include<stdlib.h>  
13 #include<string.h>  
14  
15 int top = -1;  
16 void push(char x,char stack[])  
17 {  
18  
19     if (top==100-1)  
20         printf("Overflow\n");  
21     else  
22     {  
23         top++;  
24         stack[top] = x;  
25     }  
26 }  
27  
28 char pop(char stack[])  
29 {  
30     char item;  
31     if (top==-1) {  
32         printf("Underflow\n");  
33         exit(0);  
34     }  
35     else  
36     {  
37         item = stack[top];  
38         top--;  
39         printf("popped: %c\n",item);  
40         return item;  
41     }  
42 }  
43  
44  
45 int precedence(char op)  
46 {  
47     //To check the precedence  
48  
49     char operators[7] = {'^', '*', '/', '+', '-'};  
50     int p=0;  
51     for(int i = 0; i<7; i++)  
52     {  
53         if(op == operators[i])  
54         {  
55             p = i+1;  
56         }  
57     }  
58  
59     if (p==1)  
60         return 3;  
61     else if (p==2||p==3)  
62         return 2;  
63     else if (p==4||p==5)  
64         return 1;  
65     return 0;  
66 }
```



```
136             stack[top] = temp1[i];
137         }
138     }
139     else if(top == 0 && temp1[i]!='('){
140         stack[top] = temp1[i];
141         top+=1;
142     }
143
144
145     else if(top != 0 && temp1[i]==')'){
146         while(stack[top-1]!='('){
147             prefixLen+=1;
148             prefix[prefixLen-1] = stack[top-1];
149             stack[top-1] = '\0';
150             top-=1;
151         }
152         stack[top-1] = '\0';
153         top-=1;
154     }
155     else{
156         top+=1;
157         stack[top-1] = temp1[i];
158     }
159
160     }
161 }
162
163 if(top!=0)
164 {
165     while(top>=0)
166     {
167         prefixLen += 1;
168         prefix[prefixLen-1] = stack[top];
169         stack[top] = '\0';
170         top -= 1;
171     }
172 }
173 else
174 {
175     prefixLen += 1;
176     prefix[prefixLen-1] = stack[top];
177     stack[top] = '\0';
178     top -= 1;
179 }
180
181 reverse(prefix, result);
182 }
183
184 int main()
185 {
186     char string[50], stack[50], prefix[50], temp1[50], result[50];
187     printf("Enter the infix expression: ");
188     scanf("%[^\\n]", string);
189
190     InfixToPrefix(string, stack, prefix, temp1, result);
191     printf("Prefix Expression: %s\\n", result);
192 }
193
194 }
```

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 $**A+B*CD$ using stacks.
Assume your values for A, B, C, and D. Implement the same using C
Source Code

```
1 //  
2 //  main.c  
3 //  Lab-3-7 (prefix eval)  
4 //  
5 //  Created by Aaron R on 03/09/21.  
6  
7 //  Evaluate the given prefix expression **A+B*CD using stacks. Assume your values  
8 //  for A, B, C, and D. Implement the same using C.  
9  
10  
11 #include <stdio.h>  
12 #include <stdlib.h>  
13 #include <string.h>  
14 #include <ctype.h>  
15  
16 char variables[100];  
17 int values[100];  
18 int variable_count=0;  
19  
20 struct node  
21 {  
22     int data;  
23     struct node * link;  
24 };  
25  
26 struct node * top;  
27  
28  
29 void strrev(char *str)  
30 {  
31     // Function to reverse string  
32  
33     /* skip null */  
34     if (str == 0)  
35     {  
36         return;  
37     }  
38  
39     /* skip empty string */  
40     if (*str == 0)  
41     {  
42         return;  
43     }  
44  
45     /* get range */  
46     char *start = str;  
47     char *end = start + strlen(str) - 1; /* -1 for \0 */  
48     char temp;  
49  
50     /* reverse */  
51     while (end > start)  
52     {  
53         /* swap */  
54         temp = *start;  
55         *start = *end;  
56         *end = temp;  
57  
58         /* move */  
59         ++start;  
60         --end;  
61     }  
62 }  
63 }
```

```
64
65 void push(char c)
66 {
67     // ##### Function to insert node at the head #####
68     struct node * current;
69     current = (struct node *) malloc(sizeof(struct node));
70
71     // Input the node's data
72     current->data = c;
73
74     // The address part of the new node points to the previous head
75     current->link = top;
76
77     // The head variable is updated to the current one
78     top = current;
79 }
80
81 int pop(void)
82 {
83     // ##### Function to delete the node at the beginning #####
84     struct node * temp;
85     if (top==NULL)
86     {
87         printf("The stack is empty. No elements to pop.\n");
88         exit(0);
89     }
90     else
91     {
92         temp=top;
93         int item = top->data;
94         // The address of the head is updated to the next node
95         top = top->link;
96
97         // free the memory of the previous head
98         free(temp);
99
100        return item;
101    }
102 }
103 }
104
105
106 int eval(char input[])
107 {
108     int result=0;
109     for (int i=0; i<strlen(input); i++)
110     {
111         if (isalpha(input[i]))
112         {
113             // If operand is encountered, push it into stack
114             for (int j=0; j<variable_count; j++)
115             {
116                 if (input[i]==variables[j])
117                 {
118                     push(values[j]);
119                     break;
120                 }
121             }
122         }
123     }
124 }
```

```
123     else
124     {
125         // If operator is encountered
126         // Pop 2 operands from stack
127
128         int a = pop();
129         int b = pop();
130
131         // Result = temp2 <operator> temp1
132         if (input[i]=='+')
133             result = b + a;
134         else if (input[i]=='-')
135             result = b - a;
136         else if (input[i]=='*')
137             result = b * a;
138         else
139             result = b/a;
140
141         push(result);
142     }
143 }
144 return result;
145 }

146 void rmdup(char *array, int length)
147 {
148     char *current , *end = array + length - 1;
149
150     for ( current = array + 1; array < end; array++, current = array + 1 )
151     {
152         while ( current <= end )
153         {
154             if ( *current == *array )
155             {
156                 *current = *end--;
157                 variable_count-=1;
158             }
159             else
160             {
161                 current++;
162             }
163         }
164     }
165 }
166 int main(int argc, const char * argv[])
167 {
168     char input[100];
169     printf("Enter prefix expression to be evaluated: ");
170     scanf("%[^\\n]",input);
171
172     strrev(input);
173
174     // Make a list of variables used
175     for (int i=0; i<strlen(input); i++)
176     {
177         if (isalpha(input[i]))
178         {
179             variables[variable_count]=input[i];
180             variable_count++;
181         }
182     }
183 }
```

```
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
```

All Output  Filter |  