

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define StackLimit 50
5
6  typedef int StackElementType;
7
8  typedef struct {
9      int Top;
10     StackElementType Element[StackLimit];
11 } StackType;
12
13 typedef enum {
14     FALSE, TRUE
15 } boolean;
16
17 void CreateStack(StackType *Stack);
18 void Push(StackType *Stack, StackElementType Item);
19 void Pop(StackType *Stack, StackElementType *Item);
20 boolean EmptyStack(StackType Stack);
21 boolean FullStack(StackType Stack);
22
23 int main()
24 {
25     StackType myStack;
26     StackElementType M,item,currentMemory;
27     CreateStack(&myStack);
28
29     /*???????? ?????????? ??????(????? ?????????).*/
30     printf("Please enter maximum memory address:");
31     scanf("%d",&M);
32
33     /*????????? ?????????????? ?????? ?????? ?? ?????? ? ????? 0(????????? ?????????).*/
34     do{
35         printf("Please enter the next relative memory address:");
36         scanf("%d",&item);
37
38         Push(&myStack,item);
39     }while(item != 0);
40
41     /*"?????????" ??? ?????????? 0 ??? ??? ?????? ?? ?????? ?????????????????? ??
42     ??? ?????????????? ??? ??? ?????????? ?? ??????????.*/
43     Pop(&myStack,&myStack.Element[myStack.Top]);
44
45     /*????????? ?????????? ?????? (????? ?????????).*/
46     printf("Please enter the current memory address:");
47     scanf("%d",&currentMemory);
48
49     /*????????? ????????? ???-??? ?????? ?? ?????????????? ????? ? ??????
50     ? ?????????? ??????? ?? ??????????? ?? ?????????? ??????.*/
51     while(myStack.Top != -1)
52     {
53         currentMemory+=myStack.Element[myStack.Top];
54         Pop(&myStack,&myStack.Element[myStack.Top]);
55
56         /*(????????? ??????????)*/
57         if((currentMemory >= 0) && (currentMemory <= M))
58             printf("Executing instruction: %d\n",currentMemory);
59         /*(????????? ??????????)*/
60         else
61         {
62             printf("Access Violation Exception at address:%d\n",currentMemory);
63             break;
64         }
65     }
66

```

```

67     return 0;
68 }
69
70 void CreateStack(StackType *Stack)
71 {
72     Stack -> Top = -1;
73     // (*Stack).Top = -1;
74 }
75
76 boolean EmptyStack(StackType Stack)
77 {
78     return (Stack.Top == -1);
79 }
80
81 boolean FullStack(StackType Stack)
82 {
83     return (Stack.Top == (StackLimit - 1));
84 }
85
86 void Push(StackType *Stack, StackElementType Item)
87 {
88     if (!FullStack(*Stack)) {
89         Stack -> Top++;
90         Stack -> Element[Stack -> Top] = Item;
91     } else
92         printf("Full Stack...");
93 }
94
95 void Pop(StackType *Stack, StackElementType *Item)
96 {
97     if (!EmptyStack(*Stack)) {
98         *Item = Stack -> Element[Stack -> Top];
99         Stack -> Top--;
100     } else
101         printf("Empty Stack...");
102 }

```