|  |
| --- |
|  |
|  |  |
|  | // Write a c program to sort in ascending order using temporary stack |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  |  |
|  | // Stack is represented using linked list |
|  | struct stack |
|  | { |
|  | int data; |
|  | struct stack \*next; |
|  | }; |
|  |  |
|  | // Utility function to initialize stack |
|  | void initStack(struct stack \*\*s) |
|  | { |
|  | \*s = NULL; |
|  | } |
|  |  |
|  | // Utility function to chcek if stack is empty |
|  | int isEmpty(struct stack \*s) |
|  | { |
|  | if (s == NULL) |
|  | return 1; |
|  | return 0; |
|  | } |
|  |  |
|  | // Utility function to push an item to stack |
|  | void push(struct stack \*\*s, int x) |
|  | { |
|  | struct stack \*p = (struct stack \*)malloc(sizeof(\*p)); |
|  |  |
|  | if (p == NULL) |
|  | { |
|  | fprintf(stderr, "Memory allocation failed.\n"); |
|  | return; |
|  | } |
|  |  |
|  | p->data = x; |
|  | p->next = \*s; |
|  | \*s = p; |
|  | } |
|  |  |
|  | // Utility function to remove an item from stack |
|  | int pop(struct stack \*\*s) |
|  | { |
|  | int x; |
|  | struct stack \*temp; |
|  |  |
|  | x = (\*s)->data; |
|  | temp = \*s; |
|  | (\*s) = (\*s)->next; |
|  | free(temp); |
|  |  |
|  | return x; |
|  | } |
|  |  |
|  | // Function to find top item |
|  | int top(struct stack \*s) |
|  | { |
|  | return (s->data); |
|  | } |
|  |  |
|  | // Recursive function to insert an item x in sorted way |
|  | void sortedInsert(struct stack \*\*s, int x) |
|  | { |
|  | // Base case: Either stack is empty or newly inserted |
|  | // item is greater than top (more than all existing) |
|  | if (isEmpty(\*s) || x > top(\*s)) |
|  | { |
|  | push(s, x); |
|  | return; |
|  | } |
|  |  |
|  | // If top is greater, remove the top item and recur |
|  | int temp = pop(s); |
|  | sortedInsert(s, x); |
|  |  |
|  | // Put back the top item removed earlier |
|  | push(s, temp); |
|  | } |
|  |  |
|  | // Function to sort stack |
|  | void sortStack(struct stack \*\*s) |
|  | { |
|  | // If stack is not empty |
|  | if (!isEmpty(\*s)) |
|  | { |
|  | // Remove the top item |
|  | int x = pop(s); |
|  |  |
|  | // Sort remaining stack |
|  | sortStack(s); |
|  |  |
|  | // Push the top item back in sorted stack |
|  | sortedInsert(s, x); |
|  | } |
|  | } |
|  |  |
|  | // Utility function to print contents of stack |
|  | void printStack(struct stack \*s) |
|  | { |
|  | while (s) |
|  | { |
|  | printf("%d ", s->data); |
|  | s = s->next; |
|  | } |
|  | printf("\n"); |
|  | } |
|  |  |
|  | // Driver Program |
|  | int main(void) |
|  | { |
|  | struct stack \*top; |
|  |  |
|  | initStack(&top); |
|  | push(&top, 30); |
|  | push(&top, -5); |
|  | push(&top, 18); |
|  | push(&top, 14); |
|  | push(&top, -3); |
|  |  |
|  | printf("Stack elements before sorting:\n"); |
|  | printStack(top); |
|  |  |
|  | sortStack(&top); |
|  | printf("\n\n"); |
|  |  |
|  | printf("Stack elements after sorting:\n"); |
|  | printStack(top); |
|  |  |
|  | return 0; |
|  | } |