```c
/*le funzione minimizza e aggiungi sono ok*/

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

struct stackNode {
 char comando[100];//comando
 int valore;//valore
 struct stackNode *nextPtr;
};

typedef struct stackNode STACKNODE;
typedef STACKNODE *STACKNODEPTR;

void listacomandi(STACKNODEPTR *, char [100]);//è una lista
void freelistacomandi(STACKNODEPTR *);//anche questa
STACKNODEPTR push(STACKNODEPTR *, int);
void pop(STACKNODEPTR *);//stampa la cima della pila e la elimina
void popnoprint(STACKNODEPTR *);//elimina la cima della pila
void stacksize(STACKNODEPTR);//dimensione della pila
void printstack(STACKNODEPTR);
STACKNODEPTR aggiungi(STACKNODEPTR *);//rimpiazza i primi due elementi dello stack
STACKNODEPTR minimizza(STACKNODEPTR *);//rimpiazza i primi due elementi dello stack

main(){
 STACKNODEPTR stackPtr = NULL, headPtr = NULL, listacmdPtr = NULL;
 int x, contanome = 0, val;
 char cmd[100] = {'\0'};

 scanf("%s", &cmd);
 while(x = strcmp(cmd, "quit") != 0){
  listacomandi(&listacmdPtr, cmd);
  scanf("%s", &cmd);
 }
 while(listacmdPtr != NULL){//gestisce pila e lista
  sprintf(cmd, "%s", (listacmdPtr)->comando);

  if(x = strcmp(cmd, "push") == 0){
    val = (listacmdPtr)->valore;
    headPtr = push(&stackPtr, val);
  }
  if(x = strcmp(cmd, "add") == 0){
    headPtr = aggiungi(&stackPtr);
    stackPtr = headPtr;
  }
  if(x = strcmp(cmd, "minus") == 0){
    headPtr = minimizza(&stackPtr);
    stackPtr = headPtr;
  }
  if(contanome == 0){
    printf("Steven Carmo\nGomes Andrade\n");
    printf("06\n10\n1983\nstevencarmogomes@yahoo.it\n");
    contanome++;
  }
  if(x = strcmp(cmd, "pop") == 0)
     pop(&stackPtr);
  if(x = strcmp(cmd, "size") == 0)
    stacksize(stackPtr);
  if(x = strcmp(cmd, "print") == 0){
    printstack(stackPtr);
  }
  freelistacomandi(&listacmdPtr);
 }//eowhile che gestisce la pila e la lista
 return 0;
}
```

```
//funzioni per la pila dei valori
STACKNODEPTR push(STACKNODEPTR *topPtr, int info){
 STACKNODEPTR newPtr;

 newPtr = malloc(sizeof(STACKNODE));

 if(newPtr != NULL){
  newPtr->valore = info;
  newPtr->nextPtr = *topPtr;
  *topPtr = newPtr;
 }
}

void pop(STACKNODEPTR *topPtr){
 STACKNODEPTR tempPtr;

 tempPtr = *topPtr;
 printf("%d\n", tempPtr->valore);
 *topPtr = (*topPtr)->nextPtr;
 free(tempPtr);
}

void popnoprint(STACKNODEPTR *topPtr){
 STACKNODEPTR tempPtr;

 tempPtr = *topPtr;
 *topPtr = (*topPtr)->nextPtr;
 free(tempPtr);
}

void stacksize(STACKNODEPTR topPtr){
 int conta = 0;

 while(topPtr != NULL){
  conta++;
  topPtr = topPtr->nextPtr;
 }
 printf("%d\n", conta);
}

void printstack(STACKNODEPTR currentPtr){

 if(currentPtr != NULL){
  while(currentPtr != NULL){
   printf("%d ", currentPtr->valore);
   currentPtr = currentPtr->nextPtr;
  }
  printf("\n");
 }
 else printf("\n");
}

STACKNODEPTR aggiungi(STACKNODEPTR *topPtr){
 int a = 0, b = 0, somma = 0;
 //rimpiazza i primi due elementi dello stack

 if((topPtr != NULL) && ((*topPtr)->nextPtr != NULL)){
  a = (*topPtr)->valore;
  b = (*topPtr)->nextPtr->valore;
  somma = a + b;
  (*topPtr)->valore = somma;
  (*topPtr)->nextPtr->valore = somma;
  popnoprint(&(*topPtr));
  return *topPtr;
 }
```

```c
 else return;
}

STACKNODEPTR minimizza(STACKNODEPTR *topPtr){
 int a = 0, b = 0, diff = 0;
 //rimpiazza i primi due elementi dello stack

 if((topPtr != NULL) && ((*topPtr)->nextPtr != NULL)){
  a = (*topPtr)->valore;
  b = (*topPtr)->nextPtr->valore;
  diff = a - b;
  (*topPtr)->valore = diff;
  (*topPtr)->nextPtr->valore = diff;
  popnoprint(&(*topPtr));
  return *topPtr;
 }
 else return;
}

//funzioni per la lista dei comandi
void listacomandi(STACKNODEPTR *slistaPtr, char cmdvet[100]){
 STACKNODEPTR newPtr = NULL, previousPtr = NULL, currentPtr = NULL;
 int d;

 newPtr = malloc(sizeof(STACKNODE));

 if(newPtr != NULL){
   strcpy(newPtr->comando, cmdvet);//comando
   if(d = strcmp(cmdvet, "push") == 0)
     scanf("%d", &newPtr->valore);//valore
   newPtr->nextPtr = NULL;

   previousPtr = NULL;
   currentPtr = *slistaPtr;

   while(currentPtr != NULL){//scorro la lista
     previousPtr = currentPtr;//passa al...
     currentPtr = currentPtr->nextPtr;//...prox nodo
   }
   if(previousPtr == NULL){//se la lista è vuota
     newPtr->nextPtr = *slistaPtr;//inserisco l'elemento
     *slistaPtr = newPtr;//in testa
   }
   else{//lista non vuota
     previousPtr->nextPtr = newPtr;
     newPtr->nextPtr = currentPtr;
   }
 }
}

void freelistacomandi(STACKNODEPTR *slistaPtr){
 STACKNODEPTR tempPtr = NULL;

 if(*slistaPtr != NULL){
   tempPtr = *slistaPtr;
   *slistaPtr = (*slistaPtr)->nextPtr;//prossimo comando
   free(tempPtr);
 }
}
```