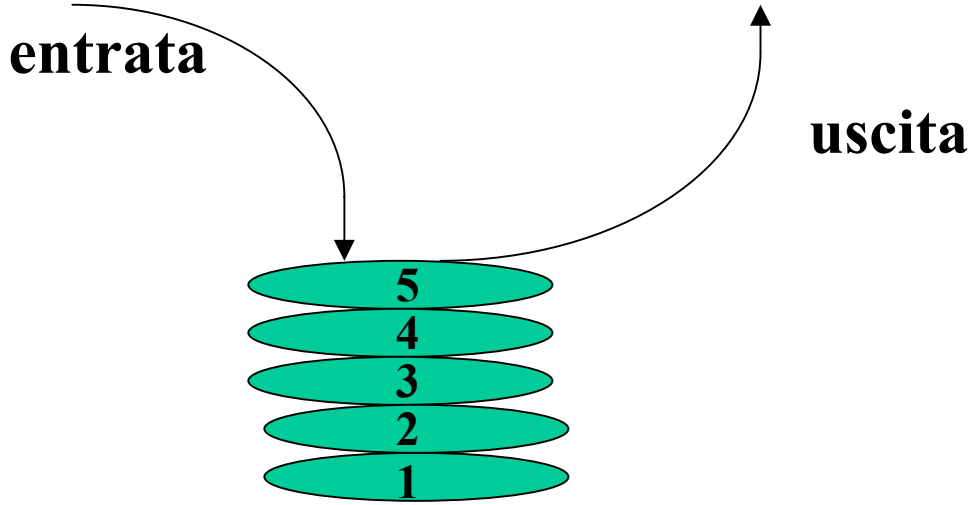
Una PILA è una collezione ad accesso regolamentato: possiamo accedere solo all'ultimo elemento inserito. Dunque inserimenti e cancellazioni possono avvenire solo in "cima" alla pila.



NOME FILE :dichImplPila.h

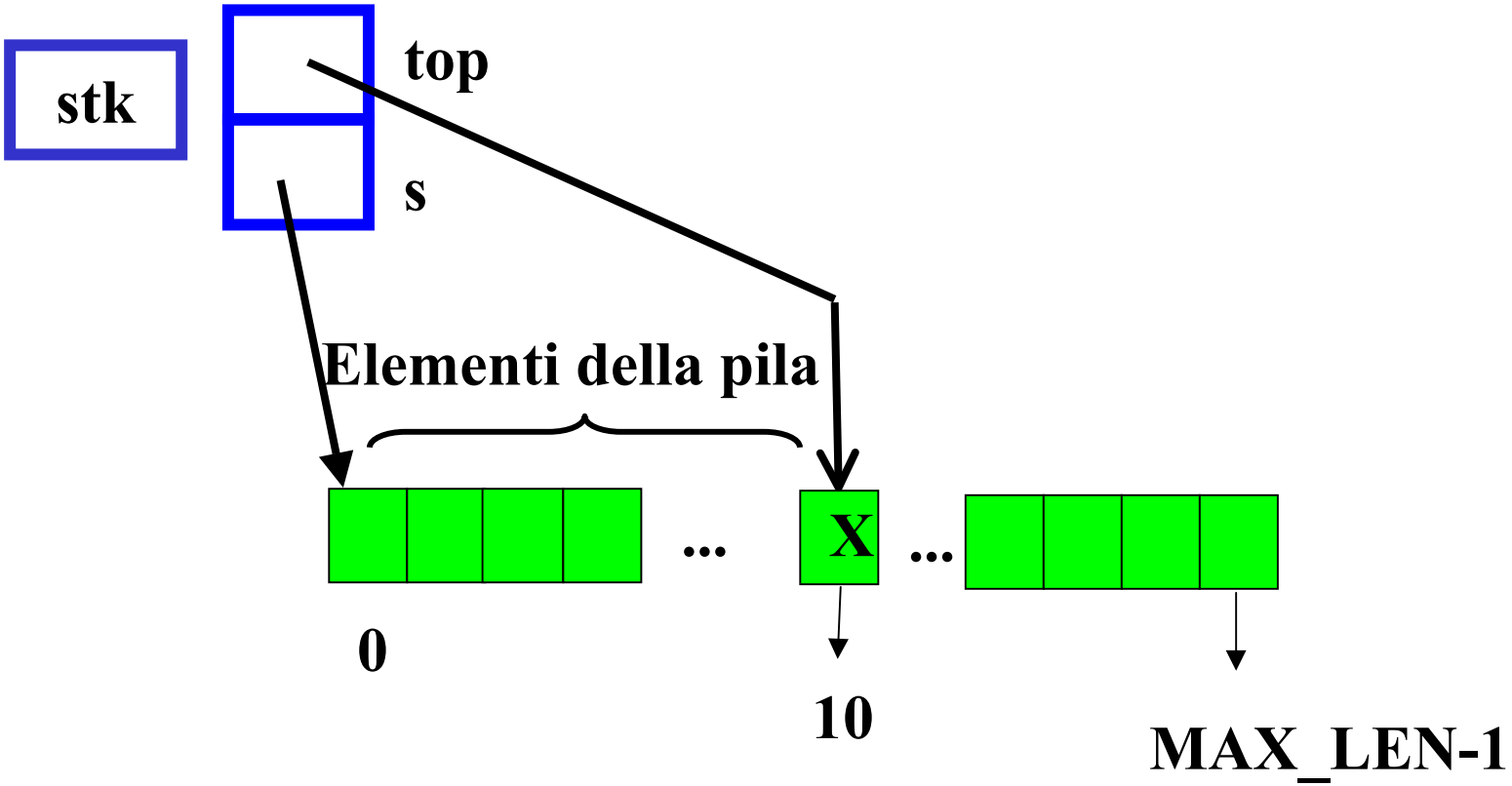
```
/* presentiamo qui un'implementazione dell'ADT  "pila" su vettore,
Invariante di implementazione: gli elementi della pila sono memorizzati in un
vettore,  l'elemento in cima alla pila e' il piu' a destra nel vettore*/

#define MAX_LEN 1000
#define EMPTY -1
#define FULL  (MAX_LEN - 1)

typedef  enum boolean {false,true} boolean;

typedef struct stack {
void* s[MAX_LEN];
int  top;
}stack;
```

stk è una variabile di tipo stack, l'elemento accessibile, la cima della pila, è stk -> s[stk -> top] =X.



NOME FILE: specPila.h

```
#include "dichImplPila.h"

void      reset(stack *stk);
/*inizializza lo stack.
 Prec: stk != NULL */
boolean   empty(const stack *stk);
/* da' vero se lo stack e' vuoto, falso altrimenti
*prec: stk != NULL*/
boolean   full(const stack *stk);
/* da' vero se lo stack e' pieno, falso altrimenti
*prec: stk != NULL*/
void      push(void* el, stack *stk);
/* inserisce el in cima allo stack
Prec: (el != NULL && stk != NULL && !full(stk)) */*/
void *    pop(stack *stk);
/* elimina l'elemento in cima allo stack, se non e' vuoto
*prec: stk != NULL &&  !empty(stk)*/
void *    top(const stack *stk);
/* legge e restituisce l'elemento in cima allo stack, se non e' vuoto
*prec: stk != NULL && !empty(stk) */
```

NOME FILE: opPilaVett.c

```
#include <assert.h>
#include "specPila.h"

void reset(stack *stk)
{assert(stk);
 stk -> top = EMPTY;
}


void* pop(stack *stk)
{assert(stk);
assert(!empty(stk));
return (stk -> s[stk -> top--]);}
```

NOME FILE: opPilaVett.c

```
void push(void* el, stack *stk)
/* inserisce el in cima allo stack
{assert(stk);
assert(el);
assert(!full(stk));
stk -> top++;
stk -> s[stk -> top] = el;
}
```

NOME FILE: opPilaVett.c

```
void* pop(stack *stk)
/* elimina e restituisce l'elemento in cima allo stack,  se non e' vuoto
{assert(stk);
assert(!empty(stk));
return (stk -> s[stk -> top--]);}

void* top(const stack *stk)
/* legge e restituisce l'elemento in cima allo stack, se non e' vuoto
{assert(stk);
assert(!empty(stk));
 return (stk -> s[stk -> top]);}
```

NOME FILE: opPilaVett.c

```
boolean empty(const stack *stk)
/* da' vero se lo stack e' vuoto, falso altrimenti */
{assert(stk);
 return ((boolean) (stk -> top == EMPTY));}


boolean full(const stack *stk)
/* da' vero se lo stack e' pieno falso altrimenti*/

{assert(stk);
return ((boolean) (stk -> top == FULL));}
```

NOME FILE: mainpilavett2.c

per il debugging

```
/*Usiamo uno stack per invertire una stringa*/
#define DEBUG
#include "specPila.h"
int main(void)
{char  *str = "una parola";
  stack   pila;
  reset(&pila);                  /* inizializziamo la pila */
  printf(" La stringa e': \n %s", str);
  putchar('\n');
  while (*str != '\0')  /* durante il ciclo si inserisce la stringa  nello stack */
            if (!full(&pila))
              { push((void*)*str, &pila);
              #ifdef DEBUG
              printf("Il carattere da inserire e': %c\n",*str);
              printf("Il carattere inserito e': %c\n",(char) (top(&pila)) );
              #endif
               str++;}
  printf("La stringa dallo stack: \n");
  while (!empty(&pila))   /*durante il ciclo si estrae il contenuto dallo stack */
            putchar((char)pop(&pila));
 putchar('\n');
 return 0;}
```

Output prodotto con #define DEBUG nel file dichPilaVett.h

```
 La stringa e':
 una parola
Il carattere da inserire e': u
Il carattere inserito e': u
Il carattere da inserire e': n
Il carattere inserito e': n
Il carattere da inserire e': a
Il carattere inserito e': a
Il carattere da inserire e':
...
Il carattere inserito e': r
Il carattere da inserire e': o
Il carattere inserito e': o
Il carattere da inserire e': a
Il carattere inserito e': a
La stringa dallo stack:
alorap anu
```

Output prodotto con #undef DEBUG nel file dichPilaVett.h (inserito dopo #define DEBUG)

```
La stringa e':
 una parola
La stringa dallo stack:
alorap anu
```

Esercizio.

Si costruisca una funzione che prende in input una stringa e restituisce vero (1) se la stringa è palindroma, falso (0) altrimenti. Si utilizzi una pila per controllare che la prima metà della stringa letta a rovescio coincida con la seconda metà, tenendo conto che nel caso di lunghezza dispari il simbolo centrale può essere qualunque.
Esempio: amoraroma è una parola palindroma di lunghezza dispari, mentre asorrosa è una parola palindroma di lunghezza pari.