14.8 Gestione di una pila mediante array

Passiamo ora a considerare il problema seguente: far gestire all'utente una struttura astratta di tipo pila, per mezzo delle operazioni di inserimento ed eliminazione di elementi, e visualizzare la pila stessa. La parte informazione di un elemento della pila è costituito da un valore intero. Bisogna implementare la pila per mezzo di un array. La soluzione del problema ha una notevole valenza didattica, perché mostra visivamente la logica di funzionamento di una pila.
Dovendo utilizzare un array, è necessario stimare il numero massimo di elementi che la pila può contenere. Si utilizzerà la variabile di tipo intero `punt_testa` come indice (riferimento) alla testa della pila, cioè all'elemento dove si effettuerà il prossimo inserimento (Figura 14.6). Se si suppone che sia n=5, `punt_testa` potrà assumere valori compresi da 0 a 5, dove 0 indicherà pila vuota, 5 pila piena. Se `pila` è il nome dell'array le dichiarazioni corrispondenti sono:

```
#define LUN_PILA 4
     int pila[LUN_PILA];
     int punt_testa = 0;
```

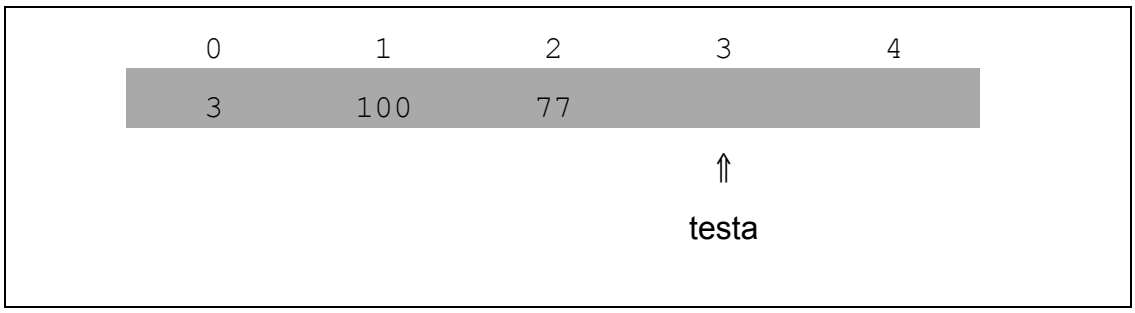


Figura 14.6 Implementazione di una pila mediante un array; il valore del puntatore alla testa è 3

Una prima suddivisione del problema porta a individuare i seguenti sottoproblemi:

- inserzione in testa alla pila (*push*);
- eliminazione in testa alla pila (*pop*);
- visualizzazione della pila.

Nel problema si richiede che l'utente gestisca la pila, ovvero si possano effettuare le tre operazioni in qualsiasi sequenza, un numero qualunque di volte. Per facilitare il compito si visualizzano le opzioni in un menu con le possibili scelte:

```
ESEMPIO UTILIZZO STRUTTURA ASTRATTA: PILA

        1. Per inserire un elemento
        2. Per eliminare un elemento
        3. Per visualizzare la pila
        0. Per finire

         Scegliere una opzione:
```

L'utente deve inserire il numero corrispondente all'opzione desiderata (1 per inserire, 2 per eliminare ecc.). La funzione `gestione_pila` presiede al trattamento della scelta:

```
while(scelta!=0) {
        visualizza menu;
        leggi l'opzione dell'utente nella variabile scelta;
        switch(scelta) {
          case 1:
              leggi l'informazione da inserire;
              esegui l'inserimento in testa alla pila;
              break;
          case 2:
              esegui l'eliminazione in testa alla pila;
              visualizza l'informazione eliminata;
              break;
          case 3:
              visualizza la pila;
              break;
    }
```

Si noti che l'operazione di inserimento di un elemento della pila non si può effettuare se la pila è piena (Figura 14.7), cioè se il numero di elementi presenti nella pila è uguale a *n*, 5 nel nostro caso (`punt_testa=5`). Perciò un ulteriore sottoproblema è quello di determinare se la pila è piena.

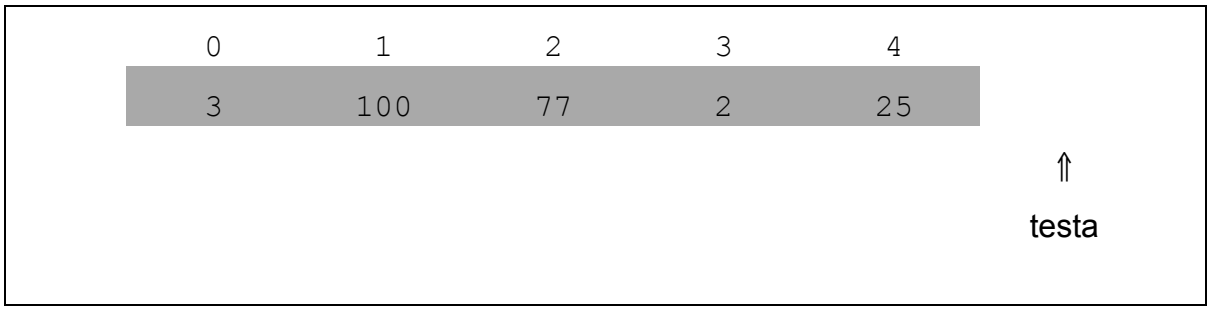


Figura 14.7 Nel caso di pila piena non è possibile effettuare un inserimento; il puntatore alla testa della pila ha valore *n*

D'altra parte l'operazione di eliminazione non si può effettuare nel caso di pila vuota. Ciò avviene quando `punt_testa` ha valore zero. Quindi un ultimo sottoproblema consiste nel determinare se la pila è vuota. Queste osservazioni permettono di dettagliare `gestione_pila`:

```
void gestione_pila(void)
     {
     int pila[LUN_PILA];
     int punt_testa = 0;
     int scelta = -1, ele;
     char pausa;

     while(scelta!=0) {
       visualizza menu
       leggi l'opzione dell'utente nella variabile scelta;
       switch(scelta) {
         case 1:
            Se (la pila è piena):
               l'inserimento è impossibile;
            Altrimenti:
               leggi l'informazione da inserire;
               Esegui l'inserimento in testa alla pila;
            break;
         case 2:
            Se (la pila è vuota):
               l'eliminazione è impossibile;
            Altrimenti:
               esegui l'eliminazione in testa alla pila;
               visualizza l'informazione eliminata;
            break;
         case 3:
            Visualizza la pila;
            break;
        }
     }
     }
```

Non rimane che risolvere i seguenti sottoproblemi:

- inserimento di un elemento in testa alla pila;
- eliminazione di un elemento in testa alla pila;
- verifica di pila piena;
- verifica di pila vuota;
- visualizzazione della pila.

Nel Listato 14.4 riportiamo il programma completo. La chiamata della funzione `inserimento`:

```
punt_testa = inserimento(pila, &punt_testa, ele);
```

prevede il passaggio dei parametri:

- `pila`, il puntatore alla struttura dati (array) che contiene fisicamente la pila;
- `punt_testa`, il puntatore (un valore intero) alla testa della pila;
- `ele`, la variabile che contiene l'elemento da inserire.

Naturalmente `ele` è passato per valore. Nella procedura questi parametri prendono i nomi di `pila`, `p` ed `ele`:

```
inserimento(int *pila, int *p, int ele)
```

La funzione inserimento effettua le seguenti azioni:

```
pila[*p] = ele;
     ++*p;

     return(*p);
```

Inserisce il valore di `ele` in `pila[*p]`, incrementa di uno il valore del puntatore alla testa e lo restituisce al chiamante (Figura 14.8). Si ricordi che `punt_testa` è passato per indirizzo e quindi per far riferimento al suo valore si deve scrivere `*p`.

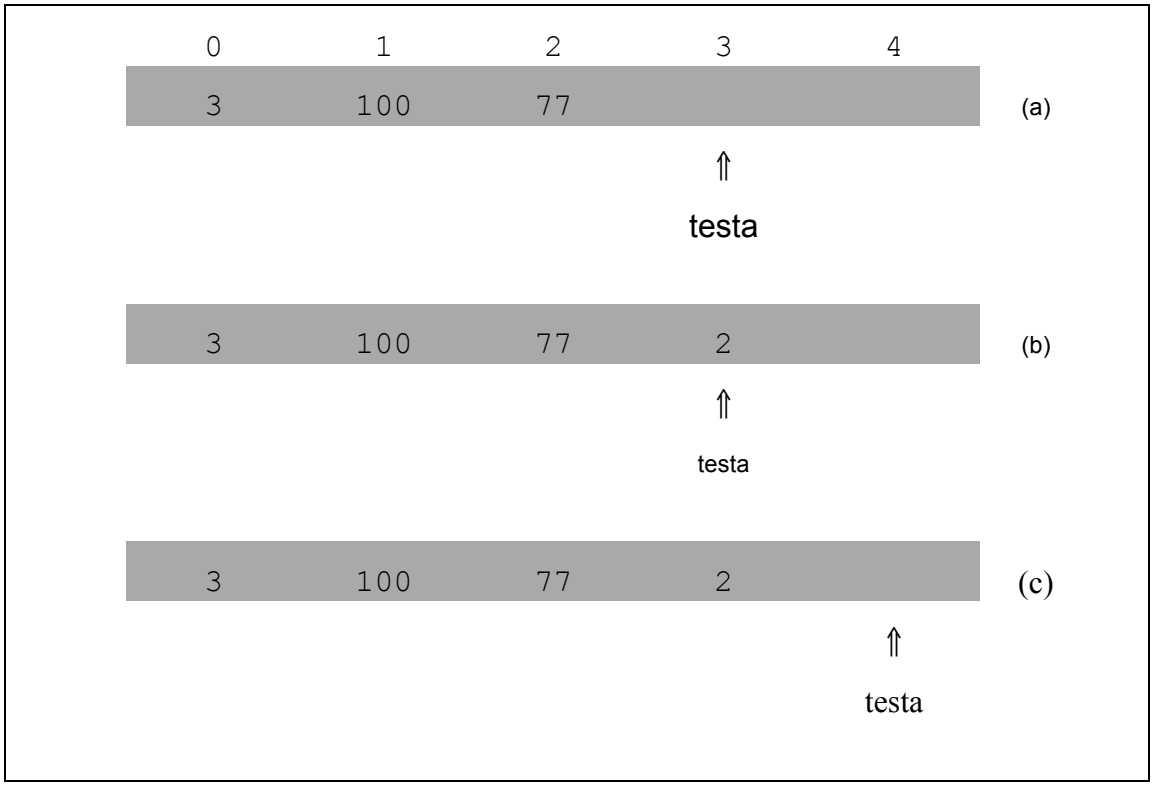


Figura 14.8 a) Stato della pila prima dell'inserimento. b) Si inserisce la nuova informazione (`ele=2`) nell'elemento dell'array indicato da `*p` (`punt_testa`). c) Si incrementa di uno il valore di `*p` (`punt_testa`)

Alla funzione di eliminazione devono essere passati gli stessi parametri della funzione `inserimento`:

```
punt_testa = eliminazione(pila, &punt_testa, &ele);
```

Questa volta il parametro `ele` viene passato per indirizzo in quanto la procedura restituisce il valore dell'elemento eliminato al `main`:

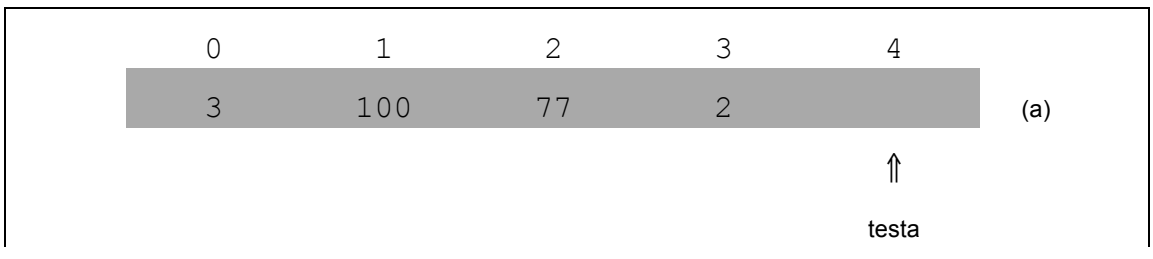
```
eliminazione(int *pila, int *p, int *ele)
```

La funzione decrementa di uno il valore di `*p`, assegna il valore dell'elemento in testa alla `pila` a `ele` e restituisce il nuovo puntatore alla testa:

```
--*p;
     *ele = pila[*p];

     return(*p);
```

Si noti che il valore dell'elemento eliminato (Figura 14.9), sebbene permanga nella struttura fisica dell'array, non fa più parte della pila, in quanto l'ultimo elemento presente nella pila è quello che precede il puntatore alla testa.



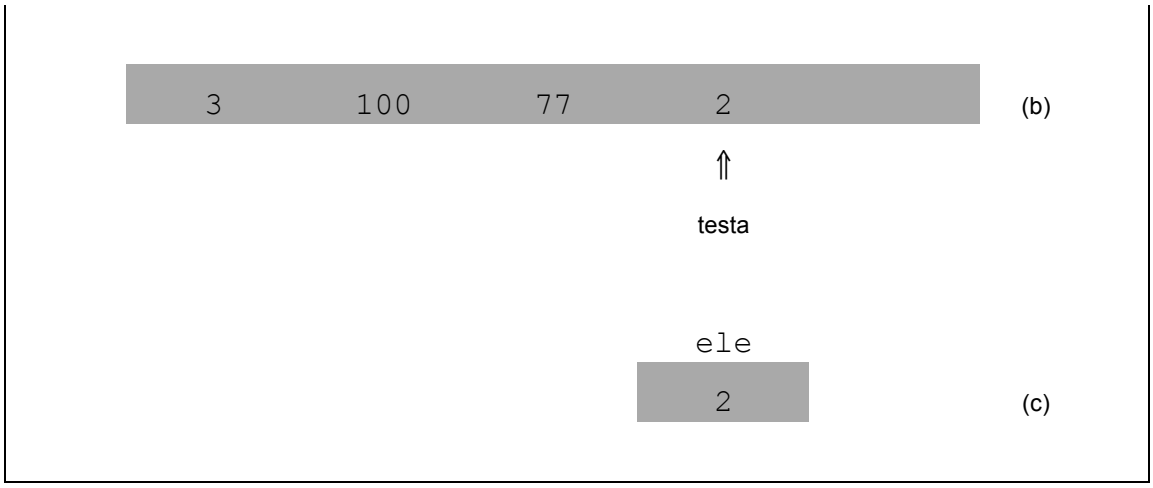


Figura 14.9 a) Stato della pila prima della eliminazione. b) Si decrementa di uno il valore di `*p` (`punt_testa`). c) L'informazione presente in testa è assegnato alla variabile `ele`

Per controllare se la pila è piena è sufficiente verificare che il puntatore alla testa sia uguale a *n*, nell'esempio *n*=5. Effettuiamo questo controllo all'interno di `gestione_pila`:

```
if(punt_testa == LUN_PILA)
        inserimento impossibile, pila piena
```

Per verificare se la pila è vuota invochiamo la funzione `pila_vuota`, passandole `punt_testa`:

```
if(pila_vuota(punt_testa) )
        eliminazione impossibile, pila vuota
     else
        eliminazione dell'elemento in testa
```

L'`if` risponde falso solamente quando l'espressione è uguale a `0`, nel qual caso è possibile procedere a una eliminazione. In effetti la funzione controlla se il puntatore alla testa è uguale a `0`, nel qual caso restituisce 1:

```
if(p==0)
       return(1);
     else
       return(0);
```

Per quel che riguarda il sottoproblema 5 – la visualizzazione della pila – si deve percorrere il vettore utilizzando un indice che va da 1 al valore del puntatore alla testa della pila, stampando di volta in volta il suo valore.

```
/* GESTIONE DI UNA PILA
   Operazioni di inserimento, eliminazione e
   visualizzazione. Utilizza un array di strutture
   per implementare la pila                        */

#include <stdio.h>
#include <malloc.h>

#define LUN_PILA 10

void gestione_pila(void);
inserimento(int *, int *, int);
eliminazione(int *, int *, int *);
pila_vuota(int);
void visualizzazione_pila(int *, int);
```

```
main()
{
gestione_pila();
}


void gestione_pila(void)
{
int pila[LUN_PILA];
int punt_testa = 0;
int scelta = -1, ele;
char pausa;

while(scelta!=0) {
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\t\tESEMPIO UTILIZZO STRUTTURA ASTRATTA: PILA");
  printf("\n\n\n\t\t\t 1. Per inserire un elemento");
  printf("\n\n\t\t\t 2. Per eliminare un elemento");
  printf("\n\n\t\t\t 3. Per visualizzare la pila");
  printf("\n\n\t\t\t 0. Per finire");
  printf("\n\n\n\t\t\t\t Scegliere una opzione: ");
  scanf("%d", &scelta);
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

  switch( scelta ) {
    case 1:
       if(punt_testa >= LUN_PILA) {
          printf("Inserimento impossibile: ");
          printf("memoria disponibile terminata");
          printf("\n\n Qualsiasi tasto per continuare...");
          scanf("%c%c", &pausa, &pausa);
       }
       else {
          printf("Inserire un elemento: ");
          scanf("%d", &ele);
          punt_testa = inserimento(pila, &punt_testa, ele);
       }
       break;
    case 2:
       if(pila_vuota(punt_testa)) {
          printf("Eliminazione impossibile: pila vuota");
          printf("\n\n Qualsiasi tasto per continuare...");
          scanf("%c%c", &pausa, &pausa);
       }
       else {
          punt_testa = eliminazione(pila, &punt_testa, &ele);
          printf("Eliminato: %d", ele );
          printf("\n\n Qualsiasi tasto per continuare...");
          scanf("%c%c", &pausa, &pausa);
       }
       break;
    case 3:
       visualizzazione_pila(pila, punt_testa);
       printf("\n\n Qualsiasi tasto per continuare...");
       scanf("%c%c", &pausa, &pausa);
       break;
  }
}
```

```
}


void visualizzazione_pila(int *pila, int p)
{
printf("\n<------ Testa della pila ");
while(p>=1)
  printf("\n%d", pila[--p]);
}


inserimento(int *pila, int *p, int ele)
{
pila[*p] = ele;
++*p;
return(*p);
}


eliminazione(int *pila, int *p, int *ele)
{
--*p;
*ele = pila[*p];
return(*p);
}


int pila_vuota(int p)
{
if(p==0)
  return(1);
else
  return(0);
}
```

Listato 14.4 Gestione di una pila implementata mediante un array