FONDAMENTI DI INFORMATICA C

FRANCO ZAMBONELLI

LE PILE (STACK)

PILA (STACK)

Nella lista le varie operazioni di inserimento, accesso e cancellazione possono avvenire in qualunque posizione.

Talvolta è desiderabile limitare tali possibilità e permettere le varie operazioni soltanto in posizioni particolari.

pila (o stack): lista in cui inserimento, cancellazione e prelievo di dati avvengono soltanto in testa.

Serve modellare situazioni in cui si memorizzano e si estraggono elementi secondo la logica *ultimo arrivato primo a uscire (Last In First Out = LIFO).*

Esempi:
1. la gestione dei record di attivazione delle procedure
2. calcolo espressioni matematiche

Operazioni base sulla pila.

NOTA: la posizione non compare più tra i parametri delle procedure.

NOTA: è possibile implementare le operazioni di pila in termini di operazioni base di lista: si lascia l'esercizio allo studente. Qui si esamineranno altri tipi di implementazione, che risultano migliori per efficienza o per semplicità di programmazione.

Descrizione	*intestazione delle funzioni in C*
crea una pila vuota	`void PilaCrea(Stack *S);`
inserisce un elemento in cima	`int Push(Stack *S; Atomo A);`
preleva e cancella l'atomo di testa	`int Pop(Stack *S; Atomo A);`
vero se la pila è vuota	`int PilaVuota(Stack S);`

NOTA: Complessità computazionale delle operazioni di pila su array: costante!

Implementazione di pila con array

Per l'implementazione con array, è sufficiente memorizzare gli elementi consecutivamente a partire dalla prima posizione, come per la lista. Le operazioni avverranno sempre sulla prima posizione libera, per inserire, e sull'ultima posizione occupata, per consultare e cancellare; non sarà quindi necessario prevedere la scrittura di codice per la creazione di spazio o il compattamento. Il file *header* è il seguente:

```
/* PilaArr.h */
#define PilaNoPop 1
#define PilaPiena 2
#define PilaOK 0

typedef int Posiz;
typedef struct {
 Posiz Cima;
 Atomo Dati[MaxDim];
} Pila;

extern void CreaPila(Pila *P);
extern int PilaVuota(Pila *P);
extern int Push(Pila *P, Atomo A);
extern int Pop(Pila *P, Atomo *A);
extern int PilaStato;
```

La libreria della struttura Pila implementata ad array

```
/* PilaArr.c */
#include "Infobase.h"
#include "PilaArr.h"
int PilaStato;

/* CreaPila */
void CreaPila(Pila *P){
 P->Cima = 0;
 PilaStato = PilaOK;
} /* end CreaPila */

/* PilaVuota */
int PilaVuota(Pila *P){ /* *P per economia */
 return (P->Cima==0);
}     /* end PilaVuota */

/* Push */
int Push(Pila *P, Atomo A){
 if ( P->Cima==MaxDim)
  PilaStato = PilaPiena;
 else {
  PilaStato = PilaOK;
  P->Cima=P->Cima+1;
  P->Dati[P->Cima-1] = A;
 }
 return PilaStato;
} /* end Push */

/* Pop */
int Pop(Pila *P, Atomo *A){
 if (P->Cima == 0)
  PilaStato = PilaNoPop;
 else {
  PilaStato = PilaOK;
  *A=P->Dati[P->Cima-1];
  P->Cima=P->Cima-1;
 }
 return PilaStato;
} /* end Pop */
```


Implementazione di pila con puntatori

La struttura degli elementi è identica al caso della lista. È sufficiente un puntatore alla testa, che assume il valore `NULL` quando la pila è vuota. Il file *header* è il seguente:

```
/* PilaPun.h */
#define PilaNoPop 1
#define PilaOK 0

typedef struct  TCellaPila {
 struct TCellaPila  *Prox;
 Atomo Dato;
} CellaPila;
typedef  CellaPila *Pila;

extern void CreaPila(Pila *P);
extern int PilaVuota(Pila P);
extern int Push(Pila *P, Atomo A);
extern int Pop(Pila *P, Atomo *A);
extern int PilaStato;
```


Questa è l'implementazione della pila a puntatori:

```
/* PilaPun.c */
#include "InfoBase.h"
#include "PilaPun.h"
#include <stdlib.h> /* serve per malloc e free
*/
int PilaStato=PilaOK;

/* CreaPila */
void CreaPila(Pila *P) {
 *P=NULL;
} /* end CreaPila */

/* PilaVuota */
int PilaVuota(Pila P) {
 PilaStato=PilaOK;
 return (P==NULL);
}  /* end PilaVuota */

/* Push */
int Push(Pila *P, Atomo A) {
 CellaPila *Temp;

 PilaStato=PilaOK;
 Temp=(Pila)malloc(sizeof(CellaPila));
 Temp->Dato=A;
 Temp->Prox=*P;
 *P=Temp;
 return PilaStato;
} /* end Push */
```



```
/* Pop */
int Pop(Pila *P, Atomo *A) {
 CellaPila *Temp;
 if (*P==NULL)
  PilaStato=PilaNoPop;
 else {
  PilaStato=PilaOK;
  Temp=*P;
  *P=Temp->Prox;
  *A=Temp->Dato;
  free(Temp);
 }
 return PilaStato;
} /* end Pop */
```

Complessità computazionale delle operazioni di pila con puntatori: costante!

Esempio: Programma per rovesciare una sequenza

Rovesciare una sequenza di caratteri forniti in input.
Si richiede prima di memorizzare tutti i caratteri che giungono in input, poi di estrarli nell'ordine inverso. In accordo con la logica di utilizzo della pila si può usare il seguente algoritmo:

```
– crea uno stack vuoto
– finché ci sono elementi nella sequenza
   – acquisisci un elemento
   – inserisci l'elemento nello stack
– finché ci sono elementi nello stack
   – prendi l'elemento in cima allo stack
   – visualizza
   – esegui pop
```


Programma C per rovesciare secquenza

indipendente dal tipo di implementazione scelto per la struttura *pila,*

- si può includere indifferentemente `PilaPun.h` o `PilaArr.h`.
- unica differenza: limitazione delle dimensioni quando si utilizza l'implementazione ad array.

```
#include <stdio.h>
#include "InfoBase.h"
#include "PilaPun.h" /* sostituibile con PilaArr.h */
void RibaltaTesto();


main(){
 char Risposta[1];

 printf("Vuoi ribaltare una riga (S/N) ? ");
 gets(Risposta);
 while (Risposta[0]=='S') {
  printf("Riga da ribaltare ? \n");
  RibaltaTesto(); /* Beta */
  printf("\n");
   /* a capo dopo il ribaltamento */ /* Teta */
  printf("Vuoi ribaltare un'altra riga (S/N) ? \n");
  gets(Risposta);
 }
 return 0;
}
```



```
void RibaltaTesto()
{
 Pila P;
 Atomo C;
 CreaPila(&P);
 do {
  C=getchar();
  if (C!='\n')
   if (Push(&P,C)!=PilaOK)
    printf("non inserito\n");
 }
 while (C!='\n');

 while (Pop(&P,&C)==PilaOK)
  printf("%c",C);
}
```


`InfoBase.h` ed `InfoBase.c`

Contengono la dichiarazione del tipo di atomo e le procedure di acquisizione e visualizzazione che da esso dipendono: per risolvere lo stesso problema con altri tipi di dato è sufficiente modificare questa parte.

```
 /* Infobase.h*/
#define MaxDim 10
#define Null '\n' /* elemento terminatore */

typedef char Atomo;

extern void Acquisisci(Atomo *A);
extern void Visualizza(Atomo A);


/* Infobase.c*/
#include <stdio.h>
#include "InfoBase.h"

void Acquisisci(Atomo *A){
  *A=getchar();
}

void Visualizza(Atomo A){
 printf("%c",A);
}
```