Algoritmi e Strutture Dati

Nicola Di Mauro

Dipartimento di Informatica
Università degli Studi di Bari “Aldo Moro”

Pile in C++

Esempio di Pila

```
L   L
A   L A
* A L
S   L S
T   L S T
I   L S T I
* I L S T
N   L S T N
* N L S T
F   L S T F
I   L S T F I
R   L S T F I R
* R L S T F I
S   L S T F I S
T   L S T F I S T
* T L S T F I S
* S L S T F I
O   L S T F I O
U   L S T F I O U
* U L S T F I O
T   L S T F I O T
* T L S T F I O
* O L S T F I
* I L S T F
* F L S T
* T L S
* S L
* L
```

Questa lista mostra il risultato di una sequenza di operazioni indicate nella colonna di sinistra (dall'alto in basso), dove una lettera denota una push *e un* asterisco *denota una* pop.
Ogni linea raffigura l'operazione, la lettera estratta (nel caso di pop) e il contenuto della pila dopo l'operazione, in ordine dall'oggetto meno recentemente inserito a quello più recentemente inserito, da sinistra a destra.

Pile: rappresentazione con vettore

```
#ifndef _PILAV_H
#define _PILAV_H
#include <nodo.h>

const int MAXLUNGH=100;

class Pila{
  friend void stampapila(Pila & L);
  public:
    Pila();
    ~Pila();
    void creaPila();
    bool pilaVuota() const;
    tipoelem leggiPila() const;
    void fuoriPila();
    void inPila(tipoelem);
  private:
    tipoelem elementi[MAXLUNGH];
    int testa;
  };
#endif // _PILAV_H
```

pilav.h

La classe nodo

```
#ifndef _NODOPV_H
#define _NODOPV_H
#include <iostream>

typedef int tipoelem;

class Nodo{
  public:
    Nodo();
    Nodo(tipoelem);
    ~Nodo();
    void setElemento(tipoelem);
    tipoelem getElemento() const;
    bool operator ==(Nodo &);
  private:
    tipoelem elemento;
  };

std::ostream &
   operator<<(std::ostream &,
    const Nodo & nodo);

# endif // _NODOPV_H
```

NODOPV.H

```
#include <nodopv.h>
using namespace std;

Nodo::Nodo(){}

Nodo::~Nodo(){}

Nodo::Nodo(tipoelem label){
  elemento=label;
}

void Nodo::setElemento(tipoelem label){
  elemento=label;
}

tipoelem Nodo::getElemento() const{
  return elemento;
}

bool Nodo::operator==(Nodo & n){
  return (getElemento() == n.getElemento());
}

std::ostream & operator<<(std::ostream & out,
     const Nodo & nodo){
  return out << nodo.getElemento();
}
```

nodopv.cpp

Implementazione classe Pila

```
#include <iostream>
#include <pilav.h>

using namespace std;

Pila::Pila(){  creaPila(); }

Pila::~Pila() {};

void Pila::creaPila(){
  testa=0;
}

bool Pila::pilaVuota() const {
  return testa==0;
}

tipoelem Pila::leggiPila() const {
  return elementi[testa-
1].getElemento();
}

void Pila::fuoriPila() {
  if (!pilaVuota())
      testa-=1;
  else
      cout<<"pila vuota"<<endl;
}
```

```
void Pila::inPila(tipoelem el){
  if (testa==MAXLUNGH)
      cout<<"capacità massima
raggiunta"<<endl;
 else {
      elementi[testa].setelemento(el);
      testa++;
  }
}
```

pilav.cpp

La funzione stampaPila

Come implementare la funzione di stampa degli elementi presenti nella pila?

```
// funzione friend della classe Pila
void stampapila(Pila & L){
  for (int i = 0; i< testa; i++)
      cout << L.elementi[i] << " ";
}
```

- friend della classe
  - accesso a private

```
// utilizzando gli operatori: distrugge la pila
void stampapila(Pila & L){
  while (!L.pilavuota()) {
     cout << L.leggiPila() << " ";
     L.fuoriPila();
  }
}
```

- utilizzo di operatori ?????
  - effetto distruttivo della pila

```
// ricorsione tail: visualizzazione in ordine inverso
// con ricostruzione
void stampapila(Pila & L){
  tipoelem Elemento;
  while (!L.pilavuota()) {
     Elemento = L.leggiPila(L);
     L.fuoriPila();
     stampapila(L);
     cout << Elemento << " ";
     L.inPila(Elemento);
  }}
```

- visualizzazione inversa
  - ricorsione tail

Template di classe Pila

```
#ifndef PILAVT_H
#define PILAVT_H

template <class Elemento>
class Pila
{
  public:
    typedef Elemento tipoelem;
    Pila();             //
costruttori
    Pila(int);
    ~Pila();       // distruttore
    void creaPila();    //operatori
    bool pilaVuota() const;
    tipoelem leggiPila() const;
    void fuoriPila();
    void inPila(tipoelem);
  private:
    tipoelem *elementi;
    int testa;
};
#endif // _PILAVT_H
```

pilavt.h

Interfaccia

- Pila(int)
- typedef Elemento tipoelem
- tipoelem *elementi

Template di classe Pila
Implementazione

```
#include <iostream>
#include <pilavt.h>
using namespace std;

template <class Elemento> Pila<Elemento>::Pila(){
  elementi = new tipoelem[100]; // dimensione standard della pila
  MAXLUNGH = 100; creaPila();
}

template <class Elemento> Pila<Elemento>::Pila(int N){
  elementi = new tipoelem[N]; // dimensione N della pila
  MAXLUNGH = N; creaPila();
}

template <class Elemento> Pila<Elemento>::~Pila() { delete[] elementi; };

template <class Elemento> void Pila<Elemento>::creaPila(){ testa=0; }

template <class Elemento> bool Pila<Elemento>::pilaVuota() const{ return (testa==0); }

template <class Elemento> Elemento Pila<Elemento>::leggiPila() const{ return elementi[testa-1]; }

template <class Elemento> void Pila<Elemento>::fuoriPila(){
  if (!pilaVuota()) testa-=1;
  else{ cout<<"nessun elemento nella pila"<<endl; }
}

template <class Elemento> void Pila<Elemento>::inPila(tipoelem el){
  if (testa==MAXLUNGH) cout<<"raggiunta capacità massima della pila"<<endl;
  else{ elementi[testa]=el; testa++; }
}
```

pilavt.cpp

Valutazione di un'espressione postfissa

Utilizzo di un pila per per valutare operazioni aritmetiche

5 * (((9 + 8) * (4 * 6)) + 7)

Il calcolo richiede di memorizzare i risultati intermedi

- Una pila è il meccanismo ideale per memorizzare risultati intermedi in questi calcoli

Iniziamo con il problema più semplice

- l'espressione da valutare è in forma *postfissa*
  - ogni operatore appare dopo i suoi due argomenti
- ogni espressione aritmetica può essere sempre riorganizzata in forma postfissa

5 9 8 + 4 6 * * 7 + * (notazione postfissa)

Valutazione di un'espressione posfiffa

```
    5
9   5   9
8   5   9   8
+   5   17
4   5   17  4
6   5   17  4   6
*   5   17  24
*   5   408
7   5   408 7
+   5   415
*   2075
```

Questa sequenza mostra l'uso di una pila per valutare l'espressione postfissa **5 9 8 + 4 6 * * 7 + ***. Procedendo da sinistra a destra nell'espressione, se incontriamo un numero lo inseriamo (push) nella pila, mentre se incontriamo un operatore, inseriamo nella pila il risultato dell'applicazione dell'operatore ai due operandi che si trovano sulla cima della pila.

Con l'uso di una pila possiamo valutare una qualsiasi espressione postfissa.
Muovendoci da sinistra a destra, interpretiamo ogni operando come il comando di “inserire (*push*) l'operando nella pila”, mentre ogni operatore verrà interpretato come il comando di “estrarre (*pop*) gli operandi dalla cima della pila, eseguire l'operazione e quindi reinserire il risultato nella pila”.

Valutazione di un'espressione postfissa

Client di una pila che legge un'espressione postfissa in cui compaiono operazioni di addizione e moltiplicazione di interi, valuta l'espressione e stampa il risultato.
Il codice per operatori non commutativi, come la sottrazione e la divisione sarebbe leggermente più complicato.

```
#include <iostream>
#include <string>
#include “pila.h”

int main(int argc, char *argv[]){
   char *a = argv[1]; int N = strlen(a);
   pila<int> postf(N); int el;
   for (int i = 0; i < N; i++){
      if (a[i] == '+'){
         el = postf.leggiPila(); postf.fuoriPila();
         postf.inPila( el + postf.leggiPila()); postf.fuoriPila()
      }
      if (a[i] == '*'){
         el = postf.leggiPila(); postf.fuoriPila();
         postf.inPila(el * postf.leggiPila()); postf.fuoriPila()
      }
      if ((a[i] >= '0') && (a[i] <= '9')){
         el = postf.leggiPila(); postf.fuoriPila();
         postf.inPila(10 * el + (a[i++]-'0'));
      }
   }
   cout << postf.fuoriPila() << endl;
}
```

postiffa.cpp

Linguaggio PostScript

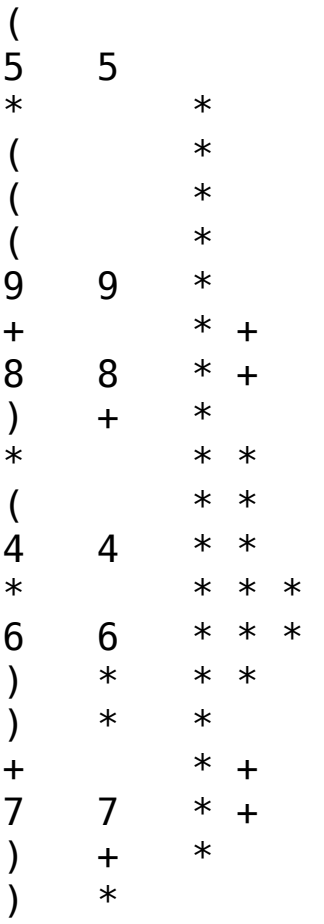
```
/hill {
   dup 0 rlineto
   60 rotate
   dup 0 rlineto
   -120 rotate
   dup 0 rlineto
   60 rotate
   dup 0 rlineto
   pop
 } def
0 0 moveto
144 hill
0 72 moveto
72 hill
stroke
```

Esercizio: estendere il codice `postfissa.cpp` in modo da includere le operazioni – e /.

Conversione di un'espressione infissa in postfissa

E' possibile utilizzare una pila per convertire espressioni infisse, con parentesi che racchiudono ogni operazione, in espressioni postfisse.
Al fine di eseguire tale operazione, inseriamo gli operatori in una pila, mentre gli operandi sono dati direttamente in output. Ogni parentesi chiusa indica che entrambi gli argomenti dell'ultima operazione sono stati dati in output, perciò l'operatore può essere estratto dalla pila e dato anch'esso in output.

Questa sequenza mostra l'uso di una pila per convertire l'espressione (5*(((9+8)*(4*6))+7)) nella sua forma postfissa 5 9 8 + 4 6 * * 7 + *. Procediamo da sinistra a destra nell'espressione: se incontriamo un numero lo scriviamo in output; se incontriamo una parentesi aperta la ignoriamo; se incontriamo un operatore lo inseriamo nella pila; se incontriamo una parentesi chiusa scriviamo l'operatore che sta in cima alla pila in output.



Conversione da forma infissa a forma postfissa

```
#include <iostream>
#include <string>
#include “pila.h”

int main(int argc, char *argv[]){
   char *a = argv[1]; int N = strlen(a);
   pila<char> operatori(N);
   for (int i = 0; i < N; i++){
      if (a[i] == ')')
         cout << operatori.leggiPila() << “ “;
         operatori.fuoriPila();
      if ((a[i] == '+') || (a[i] == '*'))
         operatori.inPila(a[i]);
      if ((a[i] >= '0') && (a[i] <= '9'))
         cout << a[i] << “ “;
   }
   cout << endl;
}
```

infissa2postfissa.cpp

Quicksort (versione iterativa)

- definire l'array a[1..n] da ordinare
- inserire nella pila il limiti *upper* (1) e *lower* (n) dell'array
- fin quando la pila non è vuota
    - rimuovere i limiti upper e lower del segmento di array dalla pila
    - fin quando il segmento non è ridotto al singolo elemento
        - selezionare l'elemento medio del segmento di array dalla pila
        - partizionare il segmento in due parti rispetto all'elemento medio
        - memorizzare nella pila i limiti della partizione più grande per una futura eleborazione ed elaborare la partizione più piccola se contiene più di un elemento

Quicksort iterativo
Implementazione

```
function quicksort(int *a; int n){
   int left, right, newleft, newright, middle, mguess, temp; Pila p;

   p.inPila(1); p.inPila(n);
   while (!p.pilaVuota()){
      right = p.leggiPila(); p.fuoriPila(); left = p.leggiPila(); p.fuoriPila();
      while (left < right){
         newleft = left; newright = right;
         middle = (left + right) /2;
         mguess = a[middle];
         while (a[newleft] < mguess) newleft++;
         while (mguess < a[newright]) newright--;
         while (newleft < newright - 1){
            temp = a[newleft]; a[newleft] = a[newright]; a[newright] = temp;
            newleft++; newright--;
            while (a[newleft] < mguess) newleft++;
            while (mguess < a[newright]) newright--;
         }
         if (newleft <= newright) {
            if (newleft < newright) {
               temp = a[newleft]; a[newleft] = a[newright]; a[newright] = temp;
            }
            newleft++; newright--;
         }
         if (newright < middle) {
            p.inPila(newleft); p.inPIla(right);
            right = newright;
         } else {
            p.inPila(left); p.inPila(newright);
            left = newleft;
         }
      }
   }
}
```

quicksort.cpp

Esercizi

- Realizzazione con puntatori della struttura dati Pila
- Estendere il codice postfissa.cpp in modo da include gli operatori – e /