Sviluppo di ADT generici

Lezione 15 di Programmazione 2

Docente: Giuseppe Scollo

Università di Catania, sede di Comiso (RG)
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Studi in Informatica applicata, AA 2006-7

Indice

ADT parametrici e classi generiche

costruttori di tipo predefiniti quali l'array specificano in modo **generico** strutture dati omogenee perché sono **parametrici in un tipo** (parametro)

anche le classi definite nella Standard Template Library (**STL**) hanno lo stesso pregio

i **template** C++ permettono di definire ADT parametrici, nonché operazioni generiche nel tipo di oggetti a cui si applicano: la loro definizione **non dipende dal tipo parametro**

ecco, ad esempio, il codice per una generica operazione di scambio dei valori fra due argomenti dello stesso tipo (qualsiasi), passati per riferimento:

```
template <class Tipo>
  void scambia(Tipo& x, Tipo& y)
  { Tipo t = x; x = y ; y = t; }
```

casi tipici importanti di classi generiche sono le **classi contenitori**, che specificano strutture dati omogenee, cioè collezioni di oggetti di uno stesso tipo (qualsiasi), dotate di operazioni tipiche di **inserimento** e **rimozione** di oggetti dal contenitore

l'ADT pila: interfaccia

un'importante classe contenitore è la **pila**, ingl. **stack** : sequenza finita, di lunghezza variabile, di dati omogenei, ad accesso sequenziale con disciplina **LIFO**

la pila è una struttura dati fondamentale:

- nella **computazione**: valutazione di **espressioni** (v. esempi appresso ed esercizi)
- nella **concorrenza**: gestione delle **interruzioni**
- per la **modularità** degli algoritmi: nidificazione delle esecuzioni di procedure

per esplorare implementazioni diverse dell'ADT con diverse strutture dati, usiamo una direttiva `#include` per la parte privata anziché specificarla direttamente con l'interfaccia:

```
template <typename Elem>
class Stack {
  public:
    Stack(int);         // costruttore (parametro: capacità della pila)
    bool empty() const; // query: restituisce true sse la pila è vuota
    void push(Elem);    // inserimento di un elemento
    Elem pop();         // rimozione di un elemento (restituito)
  private:
    #include "stack-private.h"
};
```

valutazione di un'espressione postfissa

problema: codificare una funzione C++ che, data una stringa che rappresenta un'espressione aritmetica (di tipo intero) in forma postfissa, ne restituisca il valore
soluzione: usare internamente una pila di interi per la valutazione dell'espressione

```
#include <cstdlib>
#include <cstring>
#include "stack.h"
int posteval (char *p)
{ int N = strlen(p); // lunghezza della stringa argomento
  Stack<int> save(N);
  for (int i = 0; i < N; i++)
  { if (p[i] == '+') save.push(save.pop() + save.pop());
    if (p[i] == '-') { int h = save.pop(); save.push(save.pop() - h); }
    if (p[i] == '*') save.push(save.pop() * save.pop());
    if (p[i] == '/') { int h = save.pop(); save.push(save.pop() / h); }
    if (p[i] == '%') { int h = save.pop(); save.push(save.pop() % h); }
    if ((p[i] >= '0') && (p[i] <= '9')) save.push(0);
    while ((p[i] >= '0') && (p[i] <= '9'))
      save.push(10*save.pop() + (p[i++]-'0'));
  }
  return save.pop() ;
}
```

conversione da forma infissa a postfissa

problema: codificare una procedura C++ che converta in forma postfissa la stringa che rappresenta un'espressione aritmetica (di tipo intero) in forma infissa in parentesi
soluzione: uso di una pila di caratteri per differire il posizionamento degli operatori nella forma postfissa

```
#include <cstdlib>
#include <cstring>
#include "stack.h"
void in2post(char *a, char *p)
{ int N = strlen(a), j = 0;
  Stack<char> ops(N);
  for (int i = 0; i < N; i++)
  { if (a[i] == ')') { p[j++] = ' '; p[j++] = ops.pop(); }
    if ((a[i] == '+') || (a[i] == '-') ||
        (a[i] == '*') || (a[i] == '/') || (a[i] == '%') )
    { ops.push(a[i]); p[j++] = ' '; }
    if ((a[i] >= '0') && (a[i] <= '9')) p[j++] = a[i];
  }
  p[j] = 0; // terminazione della stringa postfissa
  return;
}
```

implementazione della pila con un array

strutture dati private **stack-private.h:**

```
Elem *s; // puntatore alla struttura dati su cui si implementa la pila
int N;   // numero di elementi nella pila + 1
         // (ordinale del prossimo elemento da aggiungere alla pila)
```

implementazione delle operazioni **stack.c++** :

```
#include "stack.h"
template <typename Elem>
Stack<Elem>::Stack(int maxN)
{ s = new Elem[maxN]; N = 0; }
template <typename Elem>
bool Stack<Elem>::empty() const
{ return N == 0; }
template <typename Elem>
void Stack<Elem>::push(Elem elem)
{ s[N++] = elem; }
template <typename Elem>
Elem Stack<Elem>::pop()
{ return s[--N]; }
```

implementazione della pila con una lista

strutture dati private **stack-private.h:**

```
struct lista { Elem elem; lista* next;
               lista(Elem e, lista *l) {elem = e; next = l; } };
typedef lista *plink; plink head;
```

implementazione delle operazioni **stack.c++:**

```
#include "stack.h"
template <typename Elem>
Stack<Elem>::Stack(int)
{ head = 0; }
template <typename Elem>
bool Stack<Elem>::empty() const
{ return head == 0; }
template <typename Elem>
void Stack<Elem>::push(Elem elem)
{ plink t = new lista(elem, head); head = t; }
template <typename Elem>
Elem Stack<Elem>::pop()
{ plink t = head; Elem e = head->elem;
  head = head->next; delete t; return e; }
```

esercizi

1. codificare una funzione C++ che calcoli il valore di una stringa che rappresenta un'espressione aritmetica (di tipo intero) in forma infissa in parentesi

   suggerimento: usare le funzioni ***posteval*** e ***in2post*** definite qui

2. codificare una procedura C++ che, data una stringa che rappresenta una espressione aritmetica (di tipo intero) in forma postfissa, produca la stringa dell'equivalente forma infissa in parentesi, usando una pila
3. esercizio 4.22 del testo (Sedgewick, 2003), a p.158: modificare l'interfaccia della pila sostituendo **empty** con **count**, che restituisce il numero di elementi nella pila, e implementarla usando sia array che liste concatenate
4. esercizi 4.23 e 4.24 del testo (Sedgewick, 2003), a p.158: modificare le due implementazioni della pila viste sopra in modo che si invochi una funzione membro **error()** quando il client invoca una **pop** su una pila vuota o una **push** su una pila piena