Tipuri de date abstracte. Recursivitate

15 decembrie 2003

Programe compuse din mai multe fișiere

Implicit, obiectele declarate la nivel de fișier sunt *unice* într-un program (două declarații ale aceluiași identificator în fișiere diferite reprezintă *același obiect*, v. curs 3).
⇒ obiectul va fi *definit* într-un singur fișier, *declarat* în toate fișierele ce-l utilizează. Declarații care nu sunt definiții:
– pentru variabile: cu specificatorul `extern`
– pentru funcții, doar prototipul (antetul), nu și corpul funcției

Fazele compilării:
– compilarea în fișiere *obiect* `.c -> .o`
(cod mașină, dar conține încă nume de variabile în loc de adrese fixe)
– editarea de legături (linkeditarea): referințele la un identificator (*simbol*) din toate fișierele obiect înlocuite prin aceeași adresă

Obiectele cu specificatorul `static` nu sunt vizibile în afara fișierului
⇒ același identificator poate fi refolosit pentru obiecte diferite

Structurarea programelor din mai multe fișiere

– câte un fișier pentru porțiunile de cod care formează o entitate logică
– cu un minim de interacțiune (fără variabile globale nenecesare, etc.)
– declarațiile de tipuri, funcții și variabile ce trebuie exportate se pun într-un fișier antet `.h`
– acesta e inclus de fiecare fișier `.c` care îl necesită
– pentru a nu include/declara în duplicat, se poate încadra în

```
#ifndef __FISIERULMEU_H
#define __FISIERULMEU_H
/* aici vine continutul propriu-zis */
#endif
```


Tipuri de date abstracte

TDA = un model matematic cu un set de operații asupra lui
⇒ o structură de date + funcții care operează pe ea
⇒ noțiunea de *clasă* din programarea orientată pe obiecte

Pentru implementarea TDA în C:
– tipul de date (ex. structura) e ascuns în partea de implementare
– în fișierul `.h` se declară doar un `typedef` pt. tip (sau pointer)

```
typedef struct node {      /* în fişierul .c cu implementarea */
  int info;                /* sau/şi alte câmpuri */
  struct node *nxt;
} node_t;
typedef struct node *list; /* în fişierul .h */
```

– utilizatorul, care include doar fișierul `.h` nu are acces la structura internă a tipului; accesul e permis doar prin funcții care citesc/modifică componentele unei variabile de acest tip (ca și pt. `FILE`)

Tipuri de date abstracte (cont.)

Decizii de proiectare:
– ce operații că fie incluse
– dacă se transmit obiecte sau doar pointeri la obiecte
(pointerii sunt necesari pentru funcții care modifică obiectul)
– dacă rezultatul unei operații e returnat (eventual alocat dinamic), sau depus într-un obiect specificat (deja alocat) transmis ca parametru
– dacă funcția returnează un obiect, sau un cod de succes/eroare
(și obiectul e depus la adresa dată de un pointer parametru)

Vezi exemple de cod pentru:
– numere complexe, inclusiv suportul standard în C99 (`complex.h`)
– matrici: variante de operații cu 2 sau 3 argumente (incl. rezultatul)
– mulțimi (de întregi): cu număr fix/precizat/variabil de elemente

Recursivitatea: Exemple

– *în tipuri de date recursive*
Codul se scrie natural pornind de la definiția recursivă a structurii:
Ex. *o listă este vidă sau un element urmat de o listă*
Se pot defini atunci:
– *membru*: e primul element, sau membru în coada listei
– *șterge*: primul element, sau șterge din coada listei, etc.

La fel, se pot defini recursiv funcții care copiază sau transformă liste.

– *în analiza sintactică*
Producțiile din gramatica unui limbaj sunt tipic recursive:

expresie ::= termen | expresie + termen | expresie - termen
termen ::= factor | termen * factor | termen / factor
factor ::= număr | (expresie)

Primele două producții sunt *recursive la stânga*, pentru că neterminalul din partea stângă a lui ::= apare și ca prim element într-o variantă
⇒ se pot transforma și implementa (vezi exemplu) folosind cicluri