Tipuri de date abstracte. Recursivitate

29 noiembrie 2004

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
```

chiar dacă fișierul `.h` e inclus repetat (din mai multe locuri), conținutul său e prelucrat doar o dată (când identificatorul ales nu e definit)

Tipuri de date abstracte

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

Pentru implementarea TDA în C:
– în fişierul `.h` se declară minimul necesar pentru a putea compila programul (pentru structuri, adesea doar un `typedef` pt. pointer la tip)
– şi declaraţii de funcţii care manipulează tipul respectiv
– structura tipului şi definiţiile funcţiilor: ascunse în implementare (`.c`)

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

– utilizatorul, care include doar fişierul `.h` nu are acces la structura internă a tipului (`node_t`); accesul e permis doar prin funcţii care citesc sau modifică componentele unei variabile de acest tip (ca şi pt. `FILE`)

Tipuri de date abstracte (cont.)

Spre programarea orientată pe obiecte:
– *încapsulare*: fără acces direct la reprezentarea TDA, componentele sale sunt accesate doar prin funcţii
– funcţiile au de regulă ca prim parametru obiectul pe care operează (sau pointer la el) – similar cu *metodele* apelate pentru un obiect

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

Vezi exemplele de cod pentru: numere complexe, matrici, mulţimi

Recursivitatea: Exemple

– *în tipuri de date recursive*

Codul se scrie natural pornind de la definiţia 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

– *şterge*: primul element, sau şterge din coada listei, etc.

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

– *în analiza sintactică*

Producţiile din gramatica unui limbaj sunt tipic recursive:

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

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