Tipuri de date abstracte

30 noiembrie 2005

Programe compuse din mai multe fişiere

Implicit, obiectele declarate la nivel de fişier sunt *unice* într-un program (două declaraţii ale aceluiaşi identificator în fişiere diferite reprezintă *acelaşi obiect*, v. curs 3).
⇒ obiectul va fi *definit* într-un singur fişier, *declarat* în toate fişierele ce-l utilizează. Declaraţii care nu sunt definiţii:
– pentru variabile: cu specificatorul `extern`
– pentru funcţii, doar prototipul (antetul), nu şi corpul funcţiei

Fazele compilării:
– compilarea în fişiere *obiect* `.c -> .o`
(cod maşină, dar conţine încă nume de variabile în loc de adrese fixe)
– editarea de legături (linkeditarea): referinţele la un identificator (*simbol*) din toate fişierele obiect înlocuite prin aceeaşi adresă

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

Structurarea programelor din mai multe fişiere

– câte un fişier pentru porţiunile de cod care formează o entitate logică
– cu un minim de interacţiune (fără variabile globale nenecesare, etc.)
– declaraţiile de tipuri, funcţii şi variabile ce trebuie exportate se pun într-un fişier antet `.h`
– acesta e inclus de fiecare fişier `.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 fişierul `.h` e inclus repetat (din mai multe locuri), conţinutul 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 exemplul de cod pentru liste de intregi