

Tipuri de date abstracte

30 noiembrie 2005

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 interacțiuni (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)

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:

- în fișierul `.h` se declară minimul necesar pentru a putea compila programul (pentru structuri, adesea doar un `typedef` pt. pointer la tip)
- și declarații de funcții care manipulează tipul respectiv
- structura tipului și definițiile funcțiilor: ascunse în implementare (`.c`)

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

- utilizatorul, care include doar fișierul `.h` nu are acces la structura internă a tipului (`node_t`); accesul e permis doar prin funcții care citesc sau modifică componentele unei variabile de acest tip (ca și 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 funcții
- funcțiile 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 operații să 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 exemplul de cod pentru liste de întregi