Concepte fundamentale ale limbajelor de programare

Tipuri de date

Curs 07

conf. dr. ing. Ciprian-Bogdan Chirila

Universitatea Politehnica Timisoara
Departamentul de Calculatoare si Tehnologia Informatiei

12 aprilie 2023

Tipuri de date

- genereaza o multime de obiecte
- cu o multime de operatii pentru
  - a crea
  - a distruge
  - a modifica
- Tipuri predefinite
  - un set de obiecte predefinite specificat la definirea limbajului
- exista o constructie unitara a obiectelor in LP-urile avansate
  - structura
  - operatii

Cuprins

Tipuri predefinite

- reprezinta baza sistemului de tipuri al unui LP
- reflecta functionalitatea sistemului la nivel hardware
- valorile si operatiile relative la nivelul de date si operatii in cod masina

Tipuri predefinite

- Tipuri in baze numerice
  - C, C++: char, short, int, long, float, double, long double
  - Java: byte, short, int, long, float, double
  - C#: short, ushort, int, uint, long, ulong, float, double, decimal
  - Python: int, float, complex
- Operatii matematice
  - +,-,*,/
  - pentru intregi si reali
  - operatori sunt polimorfici si supraincarcati

Tipuri predefinite

- tipul enumerare boolean cu valori
  - true
  - false
- bool in Algol 68, C++, C#
- boolean in Pascal, Java, Ada
- char in Algol 68, Java, Pascal, C#
- character in Ada
- ASCII
- EBCDIC
  - Extended Binary Coded Decimal Interchange Code

Cuprins

Tipuri definite de programator

- cea mai puternica functionalitate a unui sistem de tipuri este sa faciliteze crearea de tipuri noi
- cu nume

```
type tab=array[1..10] of integer;
typedef struct {int x; int y;} tpoint;
```

- fara nume sau anonime

```
var t:array[1..10] of integer;
struct {int x; int y;} p1, p2;
```

Cuprins

Tipuri scalare

- obiectele de tip scalar sunt constante simple ce nu pot fi descompuse
- integer, real, character, boolean sunt tipuri scalare
- programatorul poate sa isi defineasca propriile tipuri scalare

Tipul enumerare

- utilizatorul specifica intr-o lista valorile tipului

  ```
  type days=(Sunday, Monday, Tuesday, Wednesday, Thursday,
   Friday, Saturday);
  ```

- a inceput in Pascal
- este prezent in majoritatea limbajelor de programare

  `Java, C#: enum Level {LOW, MEDIUM, HIGH}`

Alte tipuri scalare

- sunt importante din punct de vedere al portabilitatii
- in Ada
  - type eps is digits 10;
  - un tip de data numeric cu virgula flotanta cu un numar de 10 zecimale semnificative
- precizia va fi pastrata independent de platforma

Subdomenii

- in Pascal

```
type working_day=Monday..Friday;
small_caps='a'..'z';
index=0..90;
```

- in Ada

```
type eps_1 is new eps range -1.0..1.0;
```

Cuprins

Tipuri de date structurate

- limbajele de programare ofera mecanisme pentru descrierea si manipularea structurilor de date continand
  - scalari
  - alte structuri
- Mecanisme de structurare
  - functionalitati ce ne permit construirea de structuri pornind de la componentele de baza
- Mecanism de selectie
  - functionalitati ce permit accesul la componenta unei structuri

Produsul cartezian

- este reprezentat de obiecte structurate
  - compuse dintr-un numar fix de componente
  - componentele sunt de tipuri diferite
- tipul obiectului structurat corespunde produsului cartezian al multimilor corespunzatoare componentelor;
- daca tipurile componentelor sunt reprezentate de multimile $C_1, C_2, C_3, \ldots, C_n$
- atunci tipul fiecarui element structurat va fi: $T = C_1 x C_2 x \ldots x C_n$

Produsul cartezian

- este numit si
  - articole
  - structuri
- in Pascal and Ada - record
- in Algol 68 and C – structure
  - descrie tipul fiecarei componente
- a selecta o componenta inseamna a specifica obiectul si numele campului selectat

Exemplu de produs cartezian in Ada

```
type complex is
record
 re,im:real;
end record;
---
c:complex;
---
c.re:=1;
c.im:=0;
---
c:=(1,0);
```

Proiectia finita

- este o functie definita pe multimea TI cu valori in multimea TE;
- TI – tipul indexului;
- TE – tipul elementului;
- var a:array[0..99] of char;
- char a[100];
- este o proiectie a multimii 0,1,2,3,. . . ,99 pe multimea caracterelor;
- componentele sirului care sunt numite elemente sunt selectate prin mecanismul de indexare;
- numelui de tablou i se adauga textual o valoare index pentru a selecta un anumit element;

Proiectia finita

- a[k]
  - selecteaza indexul "k" din tabloul "a";
  - poate fi privita ca o aplicatie a functiei "a" cu argumentul "k", rezultand astfel valoarea elementului;
- in Algol, Ada, Python
  - selectia poate fi facuta pe un subiterval nu doar a unui singur element:

```
a[10..19]=(0,1,2,3,4,5,6,7,8,9);
thislist = ["apple", "banana", "cherry", "orange", "kiwi",
thislist[2:5]
```

  - indexul 2 este inclus, indexul 5 nu este inclus;

Momentul cheie de legare a multimii de indecsi

- fixa si legata la compilare:
  - prin codul care stabileste multimea indecsilor;
  - nu poate fi modificat in timpul executiei programelor;
  - este situatia limbajelor Fortran, C, C++, Pascal;
- fixa si legata la rulare:
  - in momentul crearii unui tablou;
  - dimensiunea sa poate fi necunoscuta la compilare;
  - poate depinde de variabile de program;
  - este cazul limbajelor Algol60, Basic sau Ada
  - in limbajele cu alocare dinamica a memoriei, ca si in cazul limbajului C, sunt utilizati pointerii pentru accesul dinamic la tablouri;

Momentul cheie de legare a multimii de indecsi

- flexibila si legata la rulare:
  - multimea indecsilor poate fi modificata;
  - dimensiunea tablourilor poate fi modificata;
  - este cazul limbajelor Snobol4 si Algol68

Secventa

- este o structura formata dintr-un numar aleator de componente de acelasi tip
- oricand poate fi adaugata o componenta
- in mod virtual numarul lor este nelimitat
- in limbajele de programare avem
  - sirurile de caractere (string-urile)
  - fisierele secventiale

Secventa

- pentru stringuri
  - in PL/I, Ada, Basic, Pascal
  - cand declaram un string trebuie sa dam numarul maxim de caractere
  - Operatii
    - dependente de limbajul de programare
    - concatenare
    - selectarea primului caracter
    - selectarea ultimului caracter
    - cautarea unui substring dintr-un string: Knuth-Morris Pratt, Boyer-Moore, Karp-Rabin
    - etc.

Alte tipuri secventiale

- in Python avem: lista, tuplul, intervalul

```
list = ["apple", "banana", "cherry"]
tuple = ("apple", "banana", "cherry")
range = range(6)
```

Recurenta

- un tip T esre recurent daca una dintre componentelesale este de tip T
- exemple tipice sunt
  - listele
  - arborii
- obiectele pot avea forme si dimensiuni arbitrare

Recurenta

```
// pseudocode
type node=record
 info:info_type;
 left, right : ^node;
end;

// C
struct node
{
 info_type info;
 struct node *left, *right;
}

// Java
class Node
{
 private InfoType info;
 private Node left, right;
}
```

Recurenta in practica

- trebuie sa utilizam pointeri
- un obiect recursiv de tipul T trebuie sa aiba o referinta la un obiect de tipul T
- nu un obiect in sine
- C, C++, Java, C#, Pascal, Ada, Algol 68
- in Lisp listele si arborii nu necesita pointeri

Reuniuni variabile

- permit specificarea structurii ce poate avea mai multe alternative
- setul tuturor valorilor posibile reprezinta reuniunea seturilor valorilor componentelor

Reuniuni variabile in C

```
union
{
 float radius;
 float rectangle_sides[2];
 float triangle_sides[3];
}shape;
```

Reuniuni variabile

- la un moment dat variabila de tip "shape" poate avea doar un camp:
  - o variabila de tip float sau
  - tablou de doua elemente de tip float sau
  - tablou de trei elemente de tip float
- intr-o reuniune variabila toate campurile coexista;
- intr-o reuniune variabila va exista valoric doar unul din campurile definite ca alternative;
- valorile sunt suprapuse pe zona de memorie ce acopera cea mai mare varianta;
- ele nu coexista simultan, ci succesiv la diverse momente de timp;

Reuniuni variabile

- in Ada si Pascal exista reuniuni variabile mult mai evoluate;
- reuniunea este parte a unui articol cu variante;

```
type figure=(circle, triangle, rectangle);
shape=record
 length,area : real;
 case shape : figure of
  circle: (radius:real);
  rectangle: (rectangle_sides:array[1..2] of real);
  triangle: (triangle_sides:array[1..3] of real);
end
```

Reuniunile variabile

- sunt periculoase;
- intotdeauna trebuie utilizata varianta corecta;
- toata responsabilitatea este lasata pe umerii programatorului (de ex. in limbajul C);
- nu exista automat posibilitatea de a face verificari in timpul rularii;
- sunt posibile verificarile doar prin cod scris de programator;
- de ex. integrarea intr-o structura cu extra informatii de tip;

Multimi

- T este tipul de baza;
- variabilele de tipul multime(T) pot avea ca si valori orice submultime generate de valorile lui T inclusiv multimea vida;
- Operatii
  - reuniune
  - intersectie
  - diferenta
  - teste de incluziune a submultimilor
  - teste de apartenenta a elementelor

Multimi

- Pascal, Python
  - au tipul multime
  - ex. in Python:

    ```
    x = {"apple", "banana", "cherry"}
    ```
- cand nu exista mecanisme dedicate in limbaj
  - pot fi implementate de programator prin:
    - tablouri de elemente booleene
    - tablouri de biti
    - liste
    - arbori

Tipul dictionar (Python)

```
x = {"name" : "John", "age" : 36}

thisdict =
{
 "brand": "Ford",
 "electric": False,
 "year": 1964,
 "colors": ["red", "white", "blue"]
}
```

Tipul dictionar (JavaScript)

```
var dict = new Object();
// sau folosind o formula prescurtata
var dict = {};

var dict =
{
  FirstName: "Chris",
  "one": 1,
  1: "some value"
};

// utilizand mecanismul de indexare
dict["one"] = 1;
dict[1] = "one";

// actualizand sau adaugand proprietati noi
dict["Age"] = 42;

// accesand direct proprietatea pe baza de nume
dict.FirstName = "Chris";
// este posibil deoarece JS este limbaj dinamic
```

Cuprins

Tipul pointer

- un pointer reprezinta o referinta la un obiect
- este modalitatea uzuala de a implementa structuri de date recursive
- in limbajul C este singura varianta de a transmite parametri prin adresa

Probleme cu tipul pointer

- violarea compatibilitatii tipurilor
- pseudonime
- referinte false

Cuprins

Tipuri imuabile

- sunt tipuri cu valori imuabile sau nemodificabile
- de ex. in Java tipul String
- un string nu poate fi modificat
- orice operatie ce modifica continutul va da nastere la alt obiect de tip string

Tipuri mutabile

- tipuri cu valori modificabile
- de ex. in Java tipul StringBuffer
- un string buffer poate fi modificat
- orice operatie ce modifica continutul se va aplica pe aceeasi instanta de string buffer

Cuprins

Violarea compatibilitatii intre tipuri

- in PL/I o variabila pointer poate referi orice obiect
- la compilare este imposibil de stiut tipul obiectului si de a face verificarile de tip corespunzatoare
- verificarea de tip la rulare este posibila dar este scumpa
- in Pascal si Ada pointerii au asignat tipurile de obiecte pe care ii poate referi
- in C avem pointeri generici `void*`
- pot fi vazuti ca o forma primitiva de polimorfism
- in C++ avem pointers inteligenti (smart):

  `unique\_ptr, shared\_ptr, weak\_ptr;`

Pseudonime

- un acelasi obiect este referit prin mai multe nume;
- prezenta lor in cod afecteaza lizibilitatea;

```
var a,b:^t;
a:=new(t);
b:=a;
//a si b sunt pseudonime
```

Referinte false

- sunt acelea care refera un pointer care nu mai este in viata
- este o eroare de acces

```
var a,b:^t;
a:=new(t);
b:=a;
dispose(a);
```

- "b" este o referinta falsa chiar daca "a" este atribuit cu valoarea nil

Referinte false in C

```
int *p;

void f()
{
 int x;
 p=&x;
}
...
f();
```

Compatibilitati de tip

- T1 si T2 sunt compatibile daca
  - o valoare de tipul T1 poate fi asignata la o variabila de tipul T2 (si viceversa)
  - un parametru de tip T1 corespunde la un parametru actual de tip T2 (si viceversa)

Exemplu

```
type
 t=array[1..100] of integer;
 t1=array[1..100] of integer;
 t2=t1;

var
 a,b:array[1..100] of integer;
 c:t;
 d:t;
 e,f:t1;
 g:t2;
```

Compatibilitati teoretice de tipuri

- echivalenta de nume
- echivalenta structurala

Name equivalence

- cand doua variabile
  - sunt declarate impreuna sau
  - utilizeaza acelasi nume pentru tip
- in exemplu
  - a si b sunt compatibile
  - c si d sunt compatibile
  - e si f sunt compatibile
  - a sau b cu c sau d nu sunt compatibile

Echivalenta structurala

- doua variabile au tipuri compatibile daca au aceeasi structura
- verificarea de tipuri presupune inlocuirea numelui tipului cu definita acestuia
  - este un proces recursiv
  - se termina cand toate tipurile definite de utilizator sunt inlocuite
- doua tipuri sunt compatibile daca au aceeasi descriere
- in exemplul nostru
  - a, b, c, d, e, f, g sunt toate compatibile

Comparatii

- echivalenta structurala
  - simpla la implementare
- echivalenta de nume
  - presupune operatii complexe pentru a determina compatibilitatea de tip
  - permite redefinirea abstractiunilor

```
type
 price=integer;
 students_no=integer;
 cost:price;
 effective:student_no;
```

Comparatii

- variabilele "cost" si "effective"
  - sunt echivalente structural
  - atribuirea de valori de la cost la effective sau invers reprezinta o eroare semantica
- echivalenta structurala
  - in Algol 68
  - in C structurile si uniunile sunt tipuri diferite desi au structuri identice
- echivalenta de nume
  - in Ada
  - in Pascal echivalenta de tip nu este specificata, depinde de implementare

Bibliografie

1. Brian Kernighan, Dennis Ritchie, C Programming Language, second edition, Prentice Hall, 1978.
2. Carlo Ghezzi, Mehdi Jarayeri – Programming Languages, John Wiley, 1987.
3. Horia Ciocarlie – Universul limbajelor de programare, editia 2-a, editura Orizonturi Universitare, Timisoara, 2013.