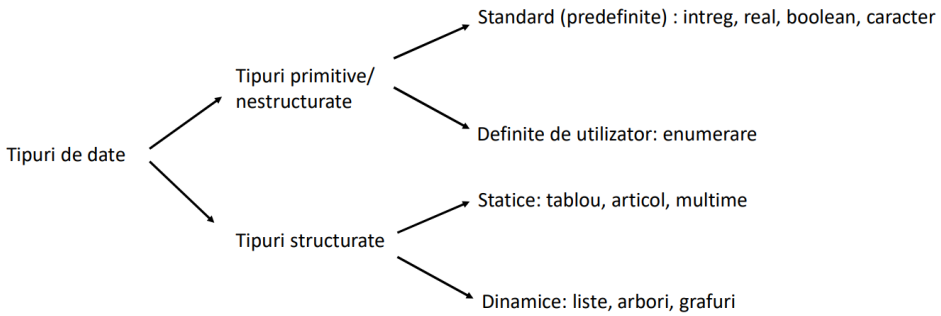
01.10.2024

Structuri de date și algoritmi (curs2 - S1)

Capitolul 2. Tipuri de date

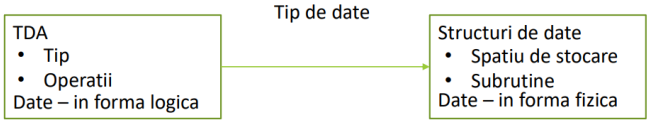
Un tip de date este o colectie de valori la care se asociaza o serie de operatii



articol == struct

Tipuri de date abstracte

Tipul de date = o implementare a unui TDA



Ex. in C/C++ tipul de date **int** reprezintă **o abstractizare a tipului abstract întreg**

Aceast tip nu reprezintă pe deplin TDA întreg, deoarece are limitări legate de mulțimea de valori care pot fi reținute

Dacă pentru o anumită aplicație aceste limitări nu sunt acceptabile, atunci trebuie folosită o altă implementare pentru numerel întregi (ex. cazul numerelor foarte mari)

Tipuri nestructurate

Definirea tipului enumerare – variante în C/C++

enum tipEnumerare {c1,c2,c3…cn} variabila;

typedef enum {c1,c2,c3…cn} numeTip;

Ex.

typedef enum {luni, marti, miercuri, joi, vineri} zileleSaptamanii;

typedef enum {luni = 1 , marti, miercuri, joi, vineri} zileleSaptamanii;

Utilizarea sa presupune atât faza de implementare (declarare) cât și de instanțiere (declarare de variabile aparținând tipului)

Proprietățile tipului enumerare:

- Tipul enumerare este un tip **numeric**
- Tipul enumerare este un tip **ordonat**
  (ci ; cj) <=> (i<j)
- Cardinalul tipului este card(tipEnumerare) = n
- Este un tip nestructurat definit de utilizator
- Suportă operatorii clasici de atribuire și comparare

Tipul boolean

- Poate fi implementat folosind tipul enumerare in limbajul C
  typedef enum {False, True} Boolean;

False = 0

True = 1

Atenție: în cazul operațiilor logice orice valoare diferită de 0 se consideră echivalentă cu valoarea de adevăr True/Adevărat

- In cazul C++ avem un tip predefinit numit bool cu valorile true (1) si false (0)

Tipul **întreg**

- implementează **o submulțime** a numerelor întregi
- dimensiunea întregilor variază de la un sistem de calcul la altul
- procesul de calcul se întrerupe în cazul obținerii unui rezultat în afara setului reprezentabil (NAN – not a number)
- Pe mulțimea numerelor întregi în afara operatorilor clasici de comparare și atribuire se definesc și operatori standard:

  Adunare, scădere, înmulțire, împărțire întreagă și modulo
- Implementările curente ale limbajelor de programare conțin mai multe categorii de tipuri întregi (short, int, long, etc.) care diferă de regulă prin numărul de cifre binare utilizate în reprezentare.

TDA întreg

- Modelul matematic: elemente scalare cu valori în mulțimea numerelor întregi {...;-2;-1;0;1;2;...}
- Notații:

  i,j,k – întregi

  inz – întreg nonzero

  inn – întreg nonnegativ

  e – valoare întreaga

  b - valoare booleană

Tipul **real**

- Implementează **o submulțime** a numerelor reale
- Aritmetica numerelor reale este **aproximativă**, în limitele erorilor de rotunjire cauzate de efectuarea calculelor cu un număr finit de cifre zecimale
- Operțiile care conduc la valori care depășesc domeniul de reprezentabilitate al implementprii duc la erori
- Implementările curente ale limbajelor de programare conțin mai multe categorii de tipuri reale (float, double, etc.) care diferă de regulă prin dimensiune și precizie.

Tipul **standard caracter**

- Cuprinde o mulțime de caractere afișabile
- Codificarea ASCII (American Standard Code for Information Interchange)
- De regulă limbajele definesc tipul de date primitiv char
- În C tipul char este un tip întreg

Stabilirea naturii unui caracter

('A' <= x) && (x >= 'Z') – x este literă mare

('a' <= x) && (x >= 'z') – x este literă mică

('0' <= x) && (x <= '9') – x este cifră

Implementarea funcțiilor de transfer întreg-caracter

char c; int n;

n=c-'0';

c=n+'0';

Tipuri de date structurate

Tipurile structurate sunt conglomerate de valori componente ale unuia sau mai multe tipuri constitutive definite anterior

Tipul de date **tablou**

- Un tablou este o structură **omogenă**, el constând dintr-o mulțime de componente de același tip numit tip de bază
- Tabloul este o structură de date cu **acces direct** (random-access), deoarece oricare dintre elementele sale sunt direct și în mod egal accesibile
- Pentru a preciza o componentă individuală, numelui întregii structuri i se asociază un **indice** care selectează poziționalcomponenta dorită

```
tipElement numeTablou[nrElemente]; /*definirea unui tablou*/
typedef tipElement tipTablou[nrElemente]; /*definirea unui tip tablou*/
tipTablou numeTablou;
/*definirea unui tablou exemple*/
float tablou[10];
char sir[10];
/*definirea unui tip tablou exemplu*/
typedef float tipTablou[100];
tipTablou a, b;
```

Tablouri **multidimensionale**

Accesarea unui element:

```
nume_tablou[indice1][indice2]..[indiceN]
```

Elementele tabloului sunt dispuse în memorie unul după altul, chiar și in cazul tablourilor multidimensionale.

Pentru tabloul:

```
int m[ LIN ] [COL] ;
```

elementul m[i][j] se află pe poziția i*COL+j, deci la i*COL+j elemente distanță de începutul tabloului.

```
/* Exemplu de implementare */
#define numarMaxElem valoareIntreaga
typedef tipElement tipTablou[numarMaxElem];
int i;
tipTablou a;
tipElement e;
a[i]=e;   //DepuneInTablou(a,I,e)
e=a[i];   //FurnizeazaDinTablou(a,i)
```

Structura **articol**

Metoda cea mai generală de a obține tipuri structurate este aceea de a reuni elemente ale mai multor tipuri, unele dintre ele fiind la rândul lor structurate, într-un tip compus

Termenul care descrie o dată compusă de această natură în programare este **struct** (C/C++), respectiv record (Pascal).

Exemple:

- Numerele complexe
- Puncte de coordonate

```
/*definire structura*/
struct nume_structura{
      tip_1 listaNumeCamp1;
      tip_2 listaNumeCamp2;
      tip_3 listaNumeCamp3;
      ...
      tip_n listaNumeCampn;
} lista_var_structura;
```

```
/*exemple*/
typedef struct data{
      int zi, luna, an;
} data_calendar;
data_calendar data_nasterii;
typedef struct student
{     char nume[30], prenume[30];
      data_calendar data_nasterii;
      enum stare{admis,respins} situatie;  }student;
```

```
/*exemplu*/
struct punct
{     int x;
      int y;};
struct punct p1;
double dist;
dist = sqrt((double)(p1.x*p1.x) + (double)(p1.y*p1.y));
```

Structura **uniune**

O uniune este de fapt o variabilă care poate memora la momente diferite de timp obiecte de tipuri și de dimensiuni diferite. Compilatorul este cel care păstrează evidența și aliniază în mod corespunzător datele memorate

Sintaxa:

- asemănătoare cu cea de la structuri, dar cu cuvântul cheie union în față.

Diferența dintre struct și union:

- pentru union lista de câmpuri reprezintă **o listă de variante**, pentru fiecare tip:
  - o variabilă structură conține **toate** câmpurile declarate
  - o variabilă uniune conține **exact una** din variantele declarate.

Dimensiunea unui tip uniune este dată de cel mai mare tip din lista de variante.

```
/*exemplu*/
union u{
int vi;
double vr;
char *vs; };
union u valoare;
valoare.vi=6;            //SAU
 //valoare.vr=7.5;    //SAU
//valoare.vs="NAN";
```

Structura **secvență**

- Tipurile structurate prezentate anterior au cardinalitate finită
- Cele mai multe structuri avansate (secvențe, liste, arbori, grafuri, etc.) sunt caracterizate prin cardinalitate infinită

Structura secvență având tipul de date T_0 se definește ca

S_0 = <> (secvența vidă)

Si = <S_{i-1}, S_i>, unde 0<1 și S_i Є T_0

Structura **secvență** este

- Fie o secvență vidă
- Fie o secvență cu un element
- Fie o concatenare a unei secvențe cu o altă secvență

Definirea **recursivă** a tipului secvență duce la o cardinalitate infinită

- Fiecare valoare a tipului secvență conține de fapt un număr finit de componente de tip T_0
- Numărul este teoretic nemărginit deoarece, pornind de la orice secvență se poate construi o secvență mai lungă

Consecințe ale cardinalității infinite

- Volumul de memorie necesar reprezentării unei structuri avansate, nu poate fi cunoscut în momentul compilării
- Este necesară aplicarea unor scheme de alocare dinamică a memoriei

TDA **secvență**

- **Modelul matematic**: Secvență de elemente de același tip. Un indicator la secvență indică elementul următor la care se poate realiza accesul. Accesul la elemente este stric secvențial.
- **Notații**
  - TipElement – tipul unui element al secvenței. Nu poate fi de tip secvență
  - f – variabilă secvență
  - e – variabilă de tip element
  - b – valoare booleană
  - numeFisierDisc – șir de caractere

TDA **secvență**

- **Operatori**
  - ***Atribuire (f, numeFisierDisc)*** *– atribuie variabilei secvență f numele unui fișier disc precizat*
  - ***Rescrie (f)*** *– mută indicatorul secvenței la începutul lui f și deschide fișierul f în regim de scriere. Dacă fișierul f nu există, el este creat. Dacă există, vechea sa variantă se pierde și se creează un nou fișier vid*
  - ***ResetSecvență (f)*** *– mută indicatorul la începutul secvenței f și deschide secvența în regim de consultare. Dacă f nu există se semnalează o eroare de execuție*
  - ***DeschideSecvență (f)*** *– în anumite implementări joacă rol de rescriere sau reset de secvență*

- **Operatori** (continuare)
  - ***b = Eof(f)*** *– funcție care returnează valoarea true dacă indicatorul secvenței indică marcherul de sfârșit de fișier al lui f*
  - ***FurnizeazăSecvență (f,e)*** *– acționează în regim de consultare. Atâta vreme cât Eof (f) este fals, furnizează în e, următorul element al secvenței f și avansează indicatorul acesteia*
  - ***DepuneSecvență (f,e)*** *– pentru secvența f deschisă în regim de scriere, copiază valoarea lui e în elementul următor al secvenței f și avansează indicatorul secvenței*
  - ***Adaugă (f)*** *– deschide secvența f în regim de scriere, poziționând indicatorul la sfârșitul fișierului cu posibilitatea de a adăuga elemente noi la sfârșitul secvenței.*
  - ***ÎnchideSecvența (f)*** *– închide secvența*

```
FILE * fp;

FILE *fopen(const char *filename, const char *mode)   //
deschidere, atribuire, rescrie/adaugă

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
//furnizează secvență

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE
*stream) //depunde secvență

int feof(FILE *stream) //b=Eof(f)

void rewind(FILE *stream) //reset secvență

int fclose(FILE *stream) //închide secvența
```