Structuri de date fundamentale

CAPITOLUL II

Cuprins

Introducere

Cantitatea mare de informații pe care o prelucrează un sistem de calcul reprezintă **o abstractizare** a lumii înconjurătoare

Informația – o mulțime de date selectate referitoare la lumea reală

Datele selectate - mulțimea de date considerată cea mai reprezentativă pentru problema tratată, se presupune că din ea pot fi deduse rezultatele dorite

Introducere

Date – abstractizare a realității

Abstractizare = o simplificare

Alegerea unei abstractizări convenabile:

1. Definirea unui set reprezentativ de date care modelează situația reală
2. Stabilirea reprezentării abstractizării

Tipuri de date

Un **tip de date** este o **colecție de valori** la care se asociază **o serie de operații**

Ex: tipul boolean:

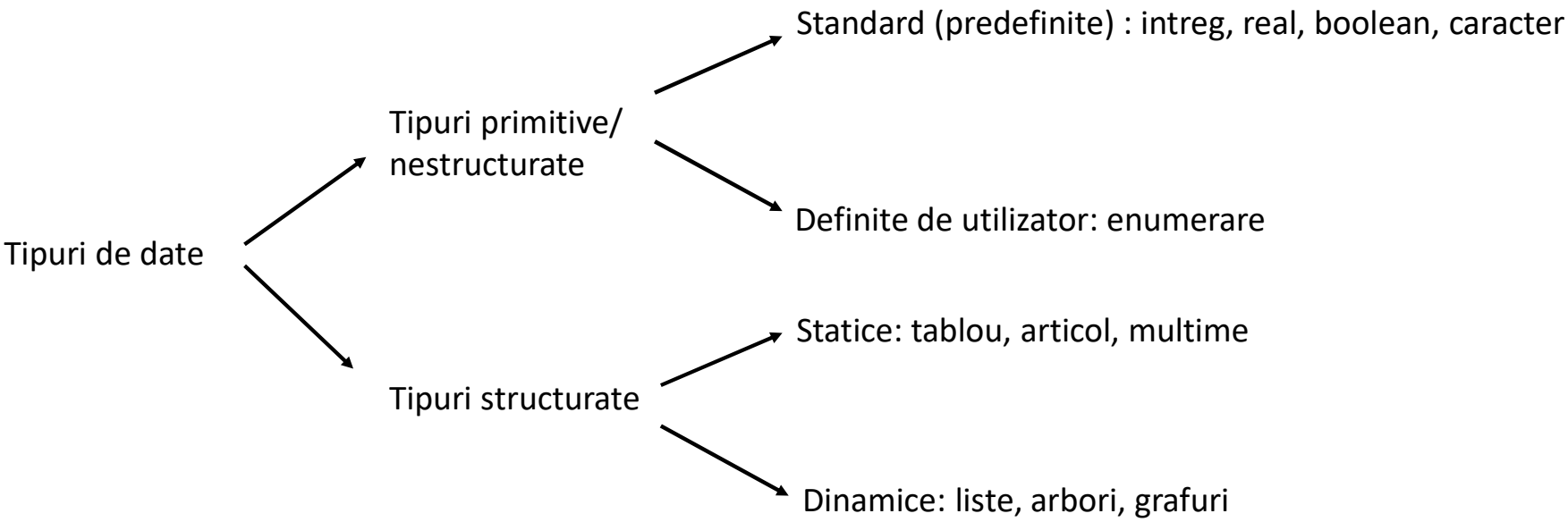
- Valori: fals, adevarat
- Operații: negare, ȘI logic, SAU logic, etc.

Trebuie făcută o diferență între **conceptul** logic de tip de date și **implementarea** acestuia

Ex. O listă este un concept logic

Lista poate fi implementată atât cu tablouri cât și cu pointeri

Tipuri de date



Tipuri de date abstracte

Tipul de date abstract (TDA) este un model **matematic** împreună cu totalitatea operațiilor definite pe acest model

La nivel conceptual algoritmii pot fi proiectați în termenii unor TDA

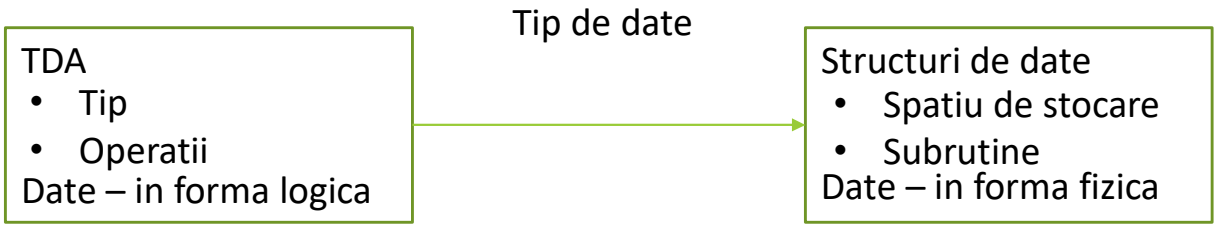
TDA nu conține specificații legate de cum poate fi implementat

Ex. Pentru un automobil avem trei operații de bază: viraj, accelerație, frânare. Două automobile diferite pot implementa operațiile în mod diferit. Cu toate acestea, putem conduce tipuri diferite de mașini, fără să ne intereseze modul de implementare a acestor funcționalități, prin simplul fapt că avem o reprezentare abstractă uniformă a acestora (mașina + operațiile = TDA)

Pentru implementarea într-un limbaj de programare concret trebuie o reprezentare a TDA în termenii **tipurilor de date** și ai operatorilor definiți în acel limbaj

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 de date abstracte

Ex. Pentru un **TDA listă** de întregi pot fi definite următoarele operații:

- Inserția unui nou întreg în listă
- Verificare listă vidă
- Numărul de elemente din listă
- Parcurgere listă cu afișare elemente
- Ștergere element

Pentru același TDA pot exista mai multe implementări (ex. cu tablouri, cu pointeri)

Tipuri de date abstracte

Definirea unui **tip de date abstract** presupune

- Precizarea **modelului matematic**
- Definirea **operatorilor** asociați

Conceptul de TDA ne poate ajuta să ne concentrăm pe aspecte cheie, chiar și în probleme non computaționale

Conceptul TDA este foarte important în domeniul calculatoarelor și tehnologiei informației, pentru că ne ajută să abordăm probleme complexe prin abstractizare

Tipuri nestructurate

Tipul **enumerare**

- Există numeroase situații când utilizăm numere întregi chiar atunci când nu sunt implicate valori numerice
- În acest caz întregii reprezintă abstractizări ale unor entități
- Tipul enumerare = tip de date primitiv, nestructurat precizat prin enumerarea tutoror valorilor sale posibile

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)

Tipuri nestructurate

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

Tipuri nestructurate

Tipul **boolean**

TDA boolean are două valori: True (Adevărat) și False (Fals)

Operatorii specifici acestui tip sunt operatorii logici: conjuncție, reuniune și negație

P	Q	P&&Q	P||Q	!P
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Tipuri nestructurate

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)

Tipuri nestructurate

Tipuri **standard predefinite**

- acele tipuri care sunt disponibile în marea majoritate a sistemelor de calcul ca și caracteristici hardware
- pot include numerele întregi, valorile logice, caractere și numere fracționale

Tipuri nestructurate

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.

Tipuri nestructurate

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

Tipuri nestructurate

Operații pe TDA întreg:

- AtribuireIntregi (i,e) – memorează valoarea lui e în vaiabila i
- k = AdunareÎntregi(i,j) (funcție)
- k = ScădereÎntregi(i,j) (funcție)
- k = ÎnmulțireÎntregi(i,j) (funcție)
- k = ÎmpărțireÎntregi(i,inz) (funcție)
- inn = Modulo(i,inz) (funcție)
- b = EgalZero(i) (funcție)
- b = MaiMareCaZero(i) (funcție)
- b = MaiMicCaZero(i) (funcție)

Tipuri nestructurate

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.

Tipuri nestructurate

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

Tipuri nestructurate

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Tipuri nestructurate

Litere mari	**A**	**Z**
Hexazecimal	X41	X5A
Zecimal	65	90
Cifre	**0**	**9**
Hexazecimal	X30	X39
Zecimal	48	57
Literele mici	**a**	**z**
Hexazecimal	X61	x7A
Zecimal	97	122
Caracterul blanc	**X20 (hexazecimal)**	**32 (zecimal)**

Tipuri nestructurate

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

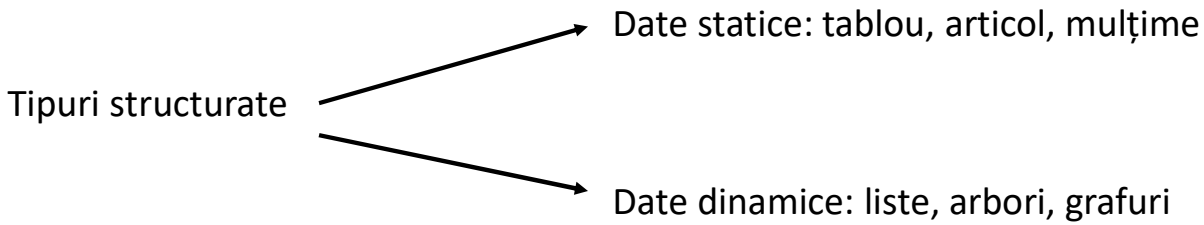
char c; int n;

n=c-’0’;

c=n+’0’;

Tipuri structurate

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



Tipuri structurate

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

Tipuri structurate

Tipul de date **tablou**

- La definirea unui tablou, se precizează în principiu:
  1. Metoda de structurare, care este implementată direct în limbaj
  2. Numele tipului de date rezultat (tipTablou)
  3. Tipul de bază al tabloului (tipElement)
  4. Tipul indice al tabloului (tipIndice)
  5. Dimensiunea tabloului, adică numărul de elemente

Tipuri structurate

Particularități limbajul C

- Tabloul poate fi implementat ca o structură de date statică pentru care rezervarea de memorie se realizează în faza de compilare a programului.
- Nu se precizează explicit metoda de structurare. Ea este indicată de prezența parantezelor drepte în sintaxa declarației
- Tipul indice este obligatoriu un tip întreg
- Domeniul valorilor indicilor este restrâns la mulțimea întregilor pozitivi
- Între paranteze drepte se indică dimensiunea tabloului (în cazul alocării statice).

Tipuri structurate

```
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;
```

Tipuri structurate

Operatori pentru TDA tablou:

Constructorul – crează o instanță a unui tablou

Selectorul – acesta selectează o componentă individuală a unui tablou

```
/*constructia unui tablou*/
int vect[5] = { 0, 1, 2, 3, 4 };
int mat[2][3] = { { 0, 1, 2 }, { 3, 4, 5 } };
char str[18] = "Structuri de date";
/*selectarea unui element*/
vect[1]           //index constant
vect[2*i]   //expresie index
```

Tipuri structurate

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.

Tipuri structurate

TDA **Tablou**

Model Matematic: Secvență de elemente de același tip. Indicele asociat aparține unui tip ordinal finit. Există o corespondență biunivocă între indici și elementele tabloului.

Notații

- tipElement – tipul elementelor tabloului
- a – tablou unidimensional
- i – index
- e- obiect de tip tipElement

Tipuri structurate

Operații pe TDA tablou**:**

DepuneInTablou(a,i,e) – funcție care depune valoarea lui e în cel de-al i-lea element al tabloului a

e = FurnizeazăDinTablou(a,i) – funcție care returnează valoarea celui de-al i-lea element al tabloului a

Tipuri structurate

/* 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)
```

Tipuri structurate

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

Tipuri structurate

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

Tipuri structurate

```
/*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;
```

Tipuri structurate

Identificatorii tipului de date articol:

- numeCamp1…numeCampn introduși la definirea tipului sunt nume conferite de către programator componentelor individuale ale tipului structurat
- sunt utilizați ca selectori de articol care permit accesul ala câmpurile unei variabile structurate de tip articol
- Exemple

```
data_calendar d;
d.luna=9;
```

Tipuri structurate

TDA articol

- **Model matematic**: O colecție finită de elemente numite câmpuri, care pot aparține unor tipuri diferite. Există o corespondență biunivocă între lista identificatorilor de câmpuri și colecția de elemente.
- **Notații**
  - a – obiect de tip articol
  - id – identificator nume câmp
  - e – obiect de același tip cu câmpul id din articolul a

Operatori pentru TDA articol:

- DepuneArticol (a,id,e) – memorează valoarea lui e în câmpul id al lui a
- e = FurnizeazaArticol (a,id) – returnează valoarea câmpului id din a

Tipuri structurate

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

Tipuri structurate

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.

Tipuri structurate

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

Tipuri structurate

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 $S_i \in T_0$

Tipuri structurate

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

Tipuri structurate

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

Tipuri structurate

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

Tipuri structurate

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*

Tipuri structurate

TDA **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*

Tipuri structurate

Exemplu in C

```
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
```

Exerciții

Ex1. Descrieți o reprezentare pentru numere întregi care să nu aibă nicio restricție legată de valoarea maximă ce poate fi reprezentată (în afară de restricțiile legate de resursele de memorie ale sistemului de calcul). Descrieți pe scurt cum poate fi folosită această reprezentare pentru calcule de adunare, înmulțire și ridicare la putere.

Ex2. Definiți un TDA listă de întregi. Specificați ce operații doriți să fie definite pentru acest TDA. Apoi propuneti un fișier de tip header pentru o implementare a unei biblioteci in C pentru TDA abstract definit anterior.

Ex3. Implementați un algoritm de sortare generic pentru vectori de elemente de tip structură. Funcția trebuie să poată fi folosită fără a trebui să fie definit în prealabil tipul structură și să poată sorta vectorul indiferent de tipul câmpului cheie după care se va face sortarea (într-un mod asemănător cu funcția qsort).

Bibliografie selectivă

- Crețu, V. Structuri de date și algorimi, Editura Orizonturi Universitare Timișoara, 2011
- Shaffer, C., “Data Structures and Algorithm Analysis”, 2012