Stiva

Capitolul

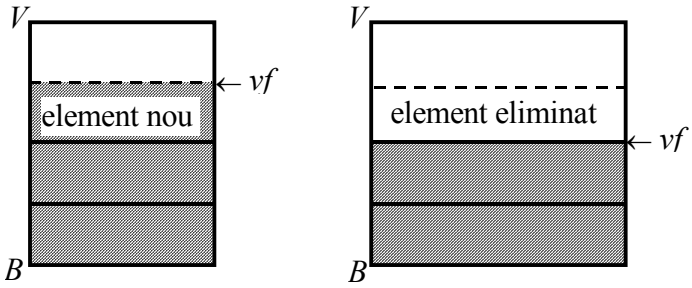
8

- **Stiva, o structură de date specială**
- **Operații cu stive**
- **Evaluarea expresiilor aritmetice**
- **Probleme propuse**
- **Soluțiile problemelor**

Definiție

Prin ***stivă*** înțelegem o structură liniară de date în care atât operațiile de introducere cât şi cele de extragere se efectuează numai la unul dintre capete numit *vârf* (*V*). Pe celălalt capăt îl numim *bază* (*B*).

Modul în care intră şi ies elementele din stivă justifică şi denumirea de *listă LIFO* (***L**ast **I**n – **F**irst **O**ut*).



Prelucrarea unei stive necesită păstrarea şi gestionarea unui *indicator de stivă* (*vf*).

O stivă se numeşte *vidă* dacă nu conține nici un element (*vf* = *B*).

O stivă se numeşte *plină* dacă spațiul de memorie rezervat acestuia este utilizat în întregime (*vf* = *V*).

O imagine sugestivă a unei stive întâlnită în viața de zi cu zi este cea a unei stive de lăzi cu mere (lăzile sunt aşezate una peste alta). Acestea, fiind foarte grele, nu permit decât extragerea lăzii de deasupra stivei sau adăugarea unei lăzi peste cele deja existente.

Implementarea stivelor se poate realiza uşor prin utilizarea unui tablou unidimensional, în cazul căruia vom declara:

D

imStivă o constantă care reprezintă dimensiunea maximă a stivei;

s

t va fi o stivă declarată sub forma unui tablou unidimensional având elemente indexate de la 1 la *DimStivă*;

v

f, indicatorul de stivă (un număr natural cuprins între 0 şi *DimStivă*).

8.1. Operaţii cu stive

8.1.1. Verificarea dacă stiva este vidă

Această operaţie o vom realiza cu subalgoritmul `StivăVidă(st,vf)` care returnează valoarea *adevărat* în cazul în care stiva este vidă şi *fals* în caz contrar:

```
Subalgoritm StivăVidă(st,vf):        { subalgoritm de tip funcţie booleană }
  dacă vf = 0 atunci
    StivăVidă ← adevărat
  altfel
    StivăVidă ← fals
  sfârşit dacă
sfârşit subalgoritm
```

8.1.2. Verificarea dacă stiva este plină

Pentru a realiza această operaţie, vom folosi subalgoritmul `StivăPlină` care, de asemenea, returnează valoarea *adevărat* în cazul în care stiva este plină şi *fals* în caz contrar:

```
Subalgoritm StivăPlină(st,vf):
  dacă vf = DimStivă atunci
    StivăPlină ← adevărat
  altfel
    StivăPlină ← fals
  sfârşit dacă
sfârşit subalgoritm
```

În cele ce urmează valoarea din vârful stivei va fi referită prin `st[vf]`.

8.1.3. Introducerea unui element în stivă

Dacă dorim să adăugăm un element nou în stiva *st*, mai întâi vom creşte numărul elementelor *vf* a şirului care implementează stiva şi adăugăm pe această poziție valoarea de introdus *ce*. Înainte de adăugarea unui element trebuie să verificăm dacă stiva nu cumva este deja plină. Această verificare o realizăm în unitatea de program care apelează subalgoritmul de adăugare pentru a evita încercarea introducerii unui element nou în caz de stivă plină.

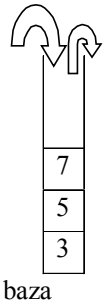
```
Subalgoritm AdăugareÎnStivă(st,vf,ce):
    vf ← vf + 1
    st[vf] ← ce
sfârşit subalgoritm
```

8.1.4. Extragerea unui element din stivă

În urma extragerii unui element dintr-o stivă obținem fie o stivă vidă (dacă am extras ultimul element), fie una care are un element mai puțin față de situația anterioară extragerii. De asemenea, în variabila *ce* vom avea conținutul acestui element îndepărtat din vârful stivei. Această operație nu poate fi efectuată decât dacă stiva nu este vidă (`vf` diferit de 0) condiție care se va verifica în unitatea de program apelantă al subprogramului `ExtragereDinStivă(st,vf,ce)`.

```
Subalgoritm ExtragereDinStivă(st,vf,ce):
  ce ← st[vf]
  vf ← vf - 1
sfârşit subalgoritm
```

Reprezentarea cea mai sugestivă a unei stive se face pe verticală, baza stivei fiind jos, iar vârful (capătul în care putem acționa) fiind sus, aşa cum se vede în figura alăturată:



Stiva o vom păstra într-un tablou unidimensional, baza stivei fiind capătul din stânga, iar vârful fiind reprezentat în capătul din dreapta.

Exemplu

Am declarat un şir având 8 elemente (indexate de la 0 la 7). Inițializarea stivei constă în inițializarea vârfului: *vf* = 0

vf

0	1	2	3	4	5	6	7

Introducem în stivă numărul *ce* = 5, acum *vf* = 1:

vf

0	1	2	3	4	5	6	7
	5						

Introducem în stivă *ce* = 7, acum *vf* = 2:

vf

0	1	2	3	4	5	6	7
	5	7					

Introducem în stivă numărul *nr* = 9, acum *vf* = 3:

vf

0	1	2	3	4	5	6	7
	5	7	9				

Extragem elementul din vârful stivei: obținem *ce* = 9 şi acum *vf* = 2:

vf

0	1	2	3	4	5	6	7
	5	7					

Observație

Conform unei astfel de implementări, valorile extrase din stivă nu se şterg fizic, ele nu vor mai fi accesate din cauza vârfului care a fost deplasat.

8.2. Evaluarea expresiilor aritmetice

Vom considera expresii aritmetice care conțin operatori binari şi variabile care au identificatori formați dintr-o singură literă mică a alfabetului englez. Abordarea acestei teme conduce spre studiul următoarelor trei aspecte:

a) Verificarea corectitudinii unei expresii

Valoarea unei expresii nu se poate calcula decât dacă expresia este corect scrisă. Pentru a decide dacă o expresie este corect scrisă vom verifica:

d

acă expresia nu conține alte caractere decât cele permise în specificarea problemei;

d

acă fiecare operator are doi operanzi (se consideră doar operatori binari);

d

acă numele fiecărui operand este corect (este format dintr-o literă mică);

d

acă numărul de paranteze deschise este egal cu numărul celor închise şi nu se închide nici una fără să fie deschisă.

b) Aducerea expresiei la forma poloneză postfixată

Forma poloneză a unei expresii aritmetice este o scriere fără paranteze a unei expresii aritmetice care, în scrierea clasică, poate să conțină şi paranteze. În forma poloneză operatorul este scris imediat *după* cei doi operanzi. Dacă unul din operanzi este la rândul lui o expresie (o subexpresie), atunci şi aceasta se scrie pe baza regulii de mai sus. Dăm câteva exemple:

Expresie aritmetică	Forma poloneză postfixată	Explicații

$a + b * c$	$abc*+$	Fie $b*c = x$. Expresia devine $ax+$ ($= a + b*c$).
$(a + b)/c(d – e)$	$ab + cde–*/$	Fie $a + b = x$. Expresia devine $xcde–*/$ Fie $d – e = y$. Expresia devine $xcy*/$ Fie $c*y = z$. Expresia devine $xz/$. Înlocuind valorile x, y, z obținem $(a + b)/c(d – e)$.

Algoritmul propus pentru transformarea unei expresii aritmetice (scrisă în forma clasică) în forma *poloneză postfixată* foloseşte două stive:

- *oper*, în care se vor introduce operatorii şi parantezele deschise;
- *polon*, în care se va obține expresia în formă postfixată.

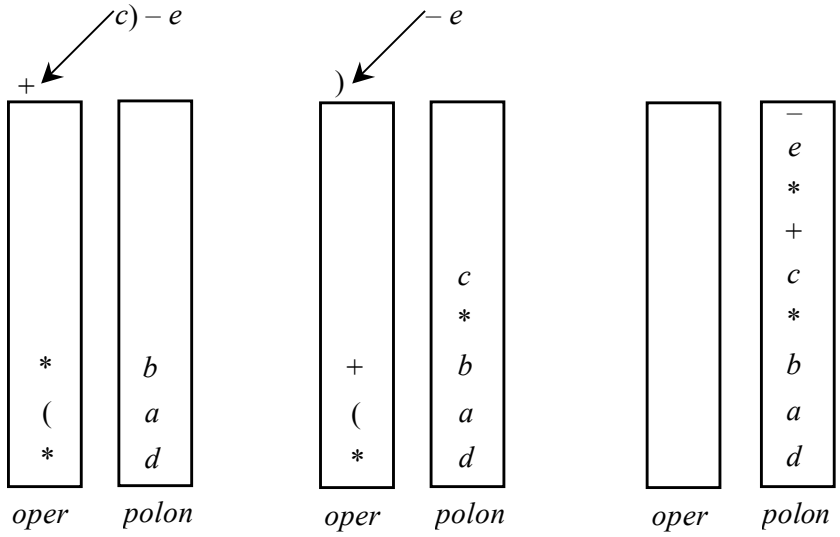
Se tratează rând pe rând fiecare caracter al expresiei, astfel:

- dacă este literă (operand) sau paranteză deschisă, se introduce în *polon*;
- dacă este operator, se introduce în *oper*, dar un operator cu prioritate mică nu poate fi introdus direct peste un operator cu prioritate mai mare. Într-un astfel de caz introducerea în *oper* se va face numai după ce în prealabil operatorii cu prioritate mai mare au fost extraşi din *oper* şi introduşi în *polon*;
- dacă avem caracter paranteză închisă, atunci se mută pe rând toți operatorii din *oper* în *polon* până la întâlnirea parantezei deschise corespunzătoare. Paranteza deschisă se elimină din *oper* fără a mai fi introdusă în *polon*.

Exemplu

Fie expresia: $d*(a*b + c) – e$.

Se introduc operatorii în *oper*, operanzii în *polon* până la întâlnirea semnului + care nu poate fi introdus peste * deoarece * are prioritate mai mare decât +. Operatorul * se mută în *polon*, tot acolo se introduce şi litera *c*. Apariția caracterului paranteză închisă are ca efect mutarea operatorului + în *polon* şi eliminarea parantezei deschise din *oper*.



În acest caz obținem expresia în formă poloneză postfixată în *polon*: *dab***c*+**e*–.

c) Evaluarea unei expresii date sub formă poloneză postfixată

Să considerăm că se citesc valori pentru variabilele *a*, *b*, *c*, *d* şi *e* (de exemplu, fie *a* = 1, *b* = 2, *c* = 7, *d* = 3, *e* = 5). Se construieşte şirul de valori (*v*) corespunzător expresiei. Acest şir va conține atâtea elemente câte caractere (litere şi operatori) există în expresie, iar valorile corespunzătoare pozițiilor ocupate de variabile vor fi valorile citite ale acestora, în rest, corespunzător caracterelor care reprezintă operatori, valoarea este 0.

indice	1	2	3	4	5	6	7	8	9
expresia	*d*	*a*	*b*	*	*c*	+	*	*e*	–
v	3	1	2	0	7	0	0	5	0

Pe măsură ce se efectuează câte o operație, din cele două şiruri se vor elimina elementele corespunzând operatorului şi operandului care îl precede, deoarece valoarea rezultată din operație îi va fi atribuită elementului de pe poziția corespunzătoare primului operand.

În exemplu prima operație care se efectuează este *ab**, rezultatul obținut este 1*2 = 2, valoare care va fi atribuită şirului *v* pe poziția corespunzătoare caracterului *a* (este primul operand al expresiei evaluate: *ab**). Şirurile devin:

indice	1	2	3	4	5	6	7
expresia	*d*	*a*	*c*	+	*	*e*	–

v	3	2	7	0	0	5	0

Următoarea operație de efectuat este *ac*+, adică 2 + 7 = 9. Structura celor două şiruri se modifică astfel:

indice	1	2	3	4	5
expresia	*d*	*a*	*	*e*	–
v	3	9	0	5	0

Acum urmează *da**, respectiv 3*9 = 27:

indice	1	2	3
expresia	*d*	*e*	–
v	27	5	0

În final *de*–, adică 27 – 5 = 22, deci se obține:

indice	1
expresia	*d*
v	22

Observație

În cazul unei expresii în care avem doi operatori consecutivi diferiți, dar ambii au prioritate maximă, ordinea acestora în forma poloneză posfixată trebuie inversată pentru a permite efectuarea operațiilor de la stânga la dreapta.

Exemplu

Fie datele de intrare: *a*b%c* şi *a* = 2, *b* = 5, *c* = 3. Forma postfixată corectă este *ab*c*%, iar valoarea corespunzătoare expresiei este 1. Dacă nu s-ar inversa operatorii, algoritmul de mai sus ar furniza expresia *abc*%* şi valoarea 4.

8.3. Probleme propuse

8.3.1. Conversie din baza 10 în altă bază

Să se convertească numărul natural *n*, dat în baza 10 într-o bază de numerație mai mică decât 37. Cifrele noului număr se vor păstra într-o stivă.

Date de intrare

Din fişierul **`DATE.IN`** se citesc două numere naturale. Primul reprezintă baza de numerație *b* în care se doreşte conversia, al doilea este numărul *n*, reprezentat în baza 10.

Date de ieşire
Fişierul de ieşire **DATE.OUT** va conţine o singură linie pe care se vor afla cifrele numărului rezultat din conversie.

Restricţii şi precizări
- $1 \le n \le 1000000000$;
- $2 \le b \le 36$.

Exemplu

DATE.IN	DATE.OUT
16 762	2FA

8.3.2. Tija cu bile

Pe o tijă verticală sudată la bază sunt dispuse *n* bile perforate în centru şi numerotate cu numere întregi oarecare. Se doreşte eliminarea bilei etichetate cu numărul *x*, folosindu-se o altă tijă verticală pe care pot fi mutate temporar bile.

Scrieţi un program care citeşte numerele asociate celor *n* bile, depune bilele pe tijă, apoi, folosind tija de manevră, elimină bila având eticheta numărul *x*.

Date de intrare
Prima linie a fişierului de intrare **TIJA.IN** conţine numărul *x* asociat bilei care trebuie eliminată de pe tijă. Pe următoarea linie se află numerele asociate bilelor. Aceste numere vor fi separate prin câte un spaţiu.

Date de ieşire
Fişierul de ieşire **TIJA.OUT** va avea următoarea structură:

Pe prima linie se va scrie mesajul: `'Se introduc pe tija: '`, urmat de valorile asociate bilelor în ordinea citirii din fişierul de intrare.

Pe a doua linie se va scrie mesajul: `'Se extrag: '`, urmat de valorile asociate bilelor care se extrag de pe tija 1 până la bila numerotată cu valoarea *x* inclusiv.

Pe a treia linie se va scrie mesajul `'Au ramas pe tija: '`, urmat de afişarea valorilor asociate bilelor rămase pe tijă după eliminarea bilei având ca etichetă numărul *x* şi după repunerea pe tija 1 a bilelor păstrate temporar pe tija 2. Afişarea se va face începând cu vârful tijei, mergând spre bază.

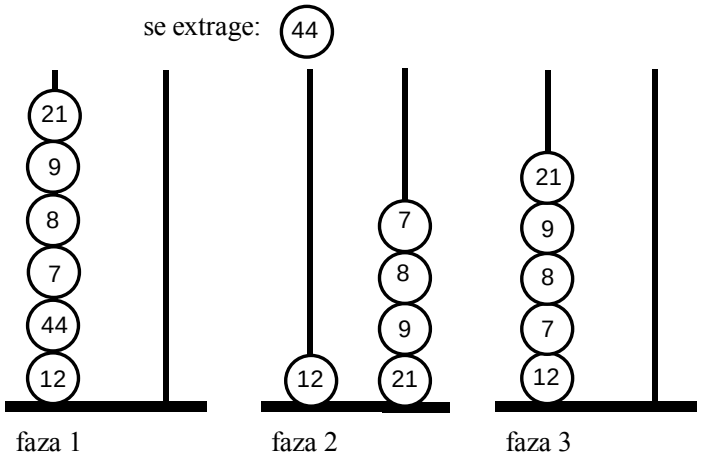
Restricţii şi precizări
- $1 \le n \le 100$;
- numerele de pe bile şi *x* sunt numere întregi.

Exemplu

TIJA.IN
```
44
12 44 7 8 9 21
```

TIJA.OUT
```
Se introduc pe tija: 12 44 7 8 9 21
Se extrag: 21 9 8 7 44
Au ramas pe tija: 21 9 8 7 12
```

Explicație



8.3.3. Tijă cu bile colorate

Pe o tijă verticală se găsesc amestecate *m* bile colorate, având *n* culori posibile. Să se separe aceste bile, punându-le pe *n* tije colorate, fiecare bilă de pe o anumită tijă având culoarea tijei. Bilele sunt numerotate de la 1 la *m*, iar culorile sunt codificate cu numere de la 1 la *n*.

Date de intrare

Pe prima linie a fişierului **BILA.IN** se află numărul de bile (*m*) şi numărul de culori (*n*). Pe următoarea linie se află şirul culorilor celor *m* bile, separate prin câte un spațiu.

Date de ieşire

Fişierul de ieşire **BILA.OUT** va conține *n* linii corespunzătoare fiecărei culori, având următorul conținut:

`'Tija de culoare: '`, urmat de un număr care reprezintă culoarea asociată tijei, apoi pe rândul următor va fi scris şirul numerelor de ordine ale bilelor aflate pe această tijă. Aceste numere vor fi separate prin câte un spațiu. Dacă pe o anumită tijă nu există bile, pe linia respectivă se va scrie: `'Tija vida!'`.

Restricții şi precizări

- $1 \le m \le 255$;
- $1 \le n \le 100$.

Exemplu

BILA.IN
```
7 4
2 2 1 3 1 3 1
```

BILA.OUT
```
Tija de culoare 1
3 5 7
Tija de culoare 2
1 2
Tija de culoare 3
4 6
Tija de culoare 4
Tija vida!
```

8.3.4. Valoarea unei expresii aritmetice

Se citeşte o expresie aritmetică în care se folosesc operatorii binari: +, –, *, /, %, unde prin operatorul „/” vom înțelege câtul împărțirii întregi al primului operand la al doilea, iar prin operatorul ,,%”, restul împărțirii întregi. Variabilele acestei expresii au identificatori formați dintr-o singură literă. Nu se folosesc constante.

Să se verifice dacă expresia este corect scrisă, iar în caz afirmativ, să se determine forma poloneză postfixată a acesteia şi să se calculeze valoarea expresiei pentru valori ale variabilelor citite ca date de intrare.

Date de intrare

Pe prima linie a fişierului de intrare **EXPRESIE.IN** se găseşte expresia. Pe următoarea linie se află valorile variabilelor separate prin câte un spațiu, în ordinea alfabetică a denumirii acestora şi nu neapărat în ordinea de apariție în expresie.

Date de ieşire

Fişierul de ieşire **EXPRESIE.OUT** va conține pe prima linie expresia aritmetică în forma poloneză postfixată, iar pe a doua linie se va afla valoarea expresiei pentru valorile variabilelor citite din fişierul de intrare.

În cazul în care expresia nu este corect scrisă, fişierul de ieşire va conține numai mesajul: `'Expresie incorectă.'`.

Restricții şi precizări

- expresia citită are cel mult 255 de caractere;
- toate valorile variabilelor sunt numere întregi;
- expresia nu conține operatori unari.

Exemple

EXPRESIE.IN
```
a+b
3 5
```

EXPRESIE.OUT
```
ab+
8
```

EXPRESIE.IN
```
a+b(c-d)
3 5 1 5
```

EXPRESIE.OUT
```
Expresie incorecta.
```

EXPRESIE.IN
```
c+b%(c-a)
1 4 3
```

EXPRESIE.OUT
```
cbca-%+
3
```

8.4. Soluțiile problemelor propuse

8.4.1. Conversie din baza 10 în altă bază

Trecerea unui număr întreg din baza de numerație 10 într-o altă bază se face prin împărțiri succesive ale numărului la noua bază, numărul fiind actualizat la fiecare pas cu câtul obținut.

Exemplu

Pentru a trece numărul 5198 din baza 10 în baza 8 avem următoarea succesiune de împărțiri:

Număr	Cât	Baza	Rest
5198 =	649 · 8 +		**6**
649 =	81 · 8 +		**1**
81 =	10 · 8 +		**1**
10 =	1 · 8 +		**2**
1 =	0 · 8 +		**1**
0			

Observăm că rezultatul dorit este construit prin culegerea resturilor în ordine inversă. În concluzie $5198_{10} = 12116_8$.

Pentru a realiza conversia unui număr dintr-o bază în alta, am putea folosi o variabilă în care am păstra puterea lui 10 care corespunde rangului acelei cifre, apoi, la fiecare pas am aduna la rezultat cifra înmulțită cu această putere.

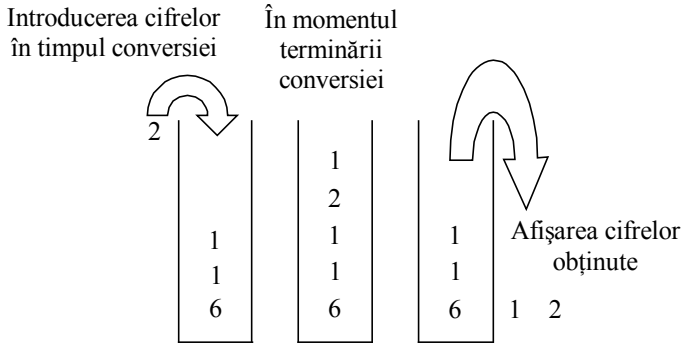
Exemplu

Rest	P	Rezultat
6	1	0 + 6 = **6**
1	10	6 + 10 = **16**
1	100	16 + 100 = **116**
2	1000	116 + 2000 = **2116**

1	10000	2116 + 10000 = **12116**

O soluție mult mai naturală ar consta în păstrarea cifrelor într-un vector, pe măsură ce acestea se obțin, afişarea cifrelor făcându-se, după terminarea algoritmului, în ordine inversă a obținerii lor. În funcție de cerință, din şirul cifrelor, parcurs de la sfârşit spre început s-ar putea construi şi numărul, folosind *schema lui Horner*.

Exemplu



O astfel de structură în care adăugăm şi extragem elemente numai la un capăt, se numeşte *stivă*. În cazul în care stiva este reprezentată folosind un şir, operațiile de adăugare şi extragere sunt uşor de realizat prin adăugarea unui element nou după ultimul existent, respectiv prin eliminarea ultimului element adăugat.

Exemplul analizat anterior a fost realizat pentru o bază de numerație mai mică decât 10. Să considerăm un exemplu nou unde baza de numerație să fie mai mare decât 10. Cifrele bazei 12 vor avea valorile: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11. Observăm că 10 şi 11 sunt numere formate din mai multe cifre, în concluzie nu pot fi utilizate ca *cifre* ale numărului rezultat. În locul lor vom folosi simboluri (litere ale alfabetului) conform unei convenții binecunoscute:

11 va fi simbolizat prin litera A;

12 va fi simbolizat prin litera B.

În cazul unor baze de numerație mai mari decât 12 vor apărea şi alte litere în funcție de numărul de valori care stau la baza sistemului de numerație respectiv. În enunț s-a specificat că baza de numerație este mai mică decât 37, deci cea mai mare poate fi 36 care necesită 36 de simboluri (de la cifra 0 la „cifra" 35. Pentru cifrele acestei baze vom folosi caracterele de la 0 la 9 (10 caractere) şi de la A la Z (26 caractere). Litera A corespunde „cifrei" 10 şi Z „cifrei" 35.

Pentru rezolvarea problemei enunțate folosim ca stivă un şir de caractere, deoarece cifrele numărului rezultat vor fi caractere (cifre sau litere).

Dacă restul împărțirii *r* este mai mic decât 10, caracterul cifră corespunzător va avea codul egal cu codul caracterului `'0'` plus restul obținut din împărțire.

caracter	`'0'`	`'1'`	`'2'`	`'3'`	`'4'`	`'5'`	`'6'`	`'7'`	`'8'`	`'9'`
cod	48	49	50	51	52	53	54	55	56	57

De exemplu, pentru *r* = 3 codul caracterului `'3'` este egal cu codul caracterului `'0'` plus `3` (*r*). În limbajul Pascal această expresie se scrie astfel:
`Chr(Ord('0')+r).`

În mod similar, dacă restul împărțirii este mai mare sau egal cu 10, acest surplus peste 10 este adunat la codul caracterului `'A'`, obținându-se codul literei corespunzătoare restului.

Exemplu

Pentru *r* = 14 surplusul este 4. Codul lui `'A'` este `65`, adunăm `4` şi obținem `69` care reprezintă codul caracterului `'E'`.

În algoritmul propus pentru rezolvarea acestei probleme am notat cu *nr* numărul întreg care trebuie convertit, cu *b* baza de numerație în care se face conversia, cu *rez* şirul de caractere în care obținem rezultatul conversiei.

```
Subalgoritm Conversie(nr,b,rez):
  rez  ← ''
  repetă
    cât ← [nr/b]                  { calculăm câtul şi restul împărțirii la bază }
    r ← rest[nr/b]              { din r construim cifra corespunzătoare în baza b }
    dacă r ∈ {0,1,...,9} atunci
      c ← Chr(Ord('0') + r)
    altfel
      c ← Chr(Ord('A') + r - 10)
    sfârşit dacă
    nr ← cât                    { schimbăm valoarea lui nr pentru pasul următor }
    rez ← c + rez                        { adăugăm cifra la rezultat (o alipim) }
  până când nr = 0
sfârşit subalgoritm
```

8.4.2. Tija cu bile

Din enunțul problemei se observă că se doreşte utilizarea a două structuri de tip stivă pentru fiecare din cele două tije. În această situație ne sunt extrem de utili subalgoritmii de introducere în stivă, respectiv extragere din stivă, subalgoritmi a căror utilizare uşurează mult întregul algoritm. Acești subalgoritmi au fost descrişi în partea teoretică a capitolului şi vor fi utilizați exact în aceeaşi formă şi în următoarele aplicații cu stive.

Pentru reprezentarea structurii de date de tip stivă vom declara un tip tablou unidi-

mensional având elemente numere întregi. Vârful stivei va fi considerat tot timpul indicele ultimului element al şirului.

Subalgoritmul de creare a stivei corespunzătoare tijei 1:

```
Subalgoritm CreareStivă(tija1,vf1):                    { DimStiva = 100 }
  vf1 ← 0                                              { iniţializare stiva 1 }
                          { citirea numerelor şi crearea stivei corespunzătoare tijei 1 }
  scrie 'Se introduc pe tija: '
  dim ← 0                                              { variabilă auxiliară }
  cât timp fişierul nu este gol şi (dim < DimStiva) execută:
    citeşte nr
    dim ← dim + 1
    scrie nr
    AdăugareÎnStivă(tija1,vf1,nr)
  sfârşit cât timp
  dacă fişierul nu este gol atunci      { nu am putut citi întregul fişier de intrare }
    scrie 'Stiva este plina!'
    Oprire forţată algoritm
  sfârşit dacă
sfârşit subalgoritm
```

Subalgoritmul în care se realizează căutarea bilei având eticheta *x* şi crearea stivei corespunzătoare tijei 2:

```
Subalgoritm CautăBila(tija1,vf1,tija2,vf2,x):
  vf2 ← 0                                              { iniţializare stiva 2 }
  scrie 'Se extrag: '            { extragerea de pe tija 1 a bilelor până la }
                              { întâlnirea bilei având ca etichetă numărul căutat }
  cât timp (vf1 > 0) şi (tija1[vf1] ≠ x) execută:
    ExtragereDinStivă(tija1,vf1,nr)
    scrie nr
    AdăugareÎnStivă(tija2,vf2,nr)
  sfârşit cât timp
  dacă tija1[vf1] = x atunci      { am ajuns la bila căutată care se elimină }
    ExtragereDinStivă(tija1,vf1,nr)
  altfel
    scrie 'Nu exista bila cu eticheta ',x
  sfârşit dacă
sfârşit subalgoritm
```

Bilele rămase pe tija 2 le vom muta înapoi pe tija 1:

```
Subalgoritm Mută(tija1,vf1,tija2,vf2,x):
                                  { bilele de pe tija 2 sunt puse la loc pe tija 1 }
```

```
  cât timp vf2 > 0 execută:
    ExtragereDinStivă(tija2,vf2,nr)
    AdăugareÎnStivă(tija1,vf1,nr)
  sfârşit cât timp
sfârşit subalgoritm
```

Pentru a putea afişa bilele de pe tija 1 conform cerinţei (de la vârf spre bază) va trebui să le mutăm din nou pe tija 2. De fiecare dată când vom lua în mână o bilă, vom scrie numărul corespunzător în fişierul de ieşire:

```
Subalgoritm Afişare(tija1,vf1):
  scrie 'Au ramas pe tija: '
  vf2 ← 0
  cât timp vf1 > 0 execută:
    ExtragereDinStivă(tija1,vf1,nr)
    scrie nr                          { scrierea în fişier a bilelor rămase }
    AdăugareÎnStivă(tija2,vf2,nr)
  sfârşit cât timp
sfârşit subalgoritm
```

8.4.3. Bile colorate

Rezolvarea acestei probleme necesită utilizarea unui *şir de n stive* în care vor fi păstrate numerele de ordine ale bilelor repartizate pe culori. Sunt folosite din nou procedurile clasice de introducere în stivă şi extragere din stivă.

Vom declara tipul `stiva` în felul următor:

```
type stiva=array[1..DimStiva] of Byte;
```

unde `DimStiva` este egal cu 255 (număr maxim de bile).

În subalgoritmii de rezolvare am notat cu *stive* un şir de stive, având cel mult 100 de elemente (numărul maxim de culori este 100). Vârfurile celor *n* stive sunt păstrate în şirul de numere naturale *vârfuri*. Stiva *st*, de vârf *vârf* cuprinde iniţial culorile tuturor bilelor citite. Numărul de ordine al bilei curente este *nr_bila*.

```
Subalgoritm Creare(st,vârf):      { se creează stiva iniţială cu toate bilele }
  vârf ← 0
  citeşte m,n                              { numărul bilelor şi culorilor }
  pentru i=1,m execută:
    citeşte culoare                                  { culoarea bilei }
    AdăugareÎnStivă(st,vârf,culoare)
  sfârşit pentru
sfârşit subalgoritm
```

Subalgoritmul de rezolvare a problemei:

```
Subalgoritm Distribuie(st,vârf,stive,vârfuri):
  pentru i=1,n execută:                    { inițializarea celor n stive }
    vârfuri[i] ← 0
  sfârşit pentru
        { luăm bilele de pe tija inițială şi le distribuim pe stive conform culorilor }
  nr_bilă ← m
  cât timp vârf > 0 execută:
    ExtragereDinStivă(st,vârf,culoare)
    AdăugareÎnStivă(stive[culoare],vârfuri[culoare],nr_bilă)
    nr_bilă ← nr_bilă - 1
  sfârşit cât timp
sfârşit subalgoritm
```

Afişarea stivelor le vom realiza cu ajutorul subalgoritmului `ExtragereDinStivă`. Subalgoritmul de afişare îl concepem pentru afişarea unei singure stive, şi îl vom apela din programul principal pentru cele *n* culori. Dacă vor fi stive vide (o anumită culoare lipseşte), în subalgoritmul `ExtragereDinStivă` vom afişa un mesaj corespunzător.

```
Subalgoritm AfişareStivă(st,vârf):
  cât timp vârf > 0 execută:
    ExtragereDinStivă(st,vârf,nr_bilă);
    scrie nr_bilă
  sfârşit cât timp
sfârşit subalgoritm
```

8.4.4. Valoarea unei expresii aritmetice

În prezentarea teoretică, susținută cu exemple, se conturează trei etape necesare evaluării unei expresii:

- verificarea corectitudinii expresiei;
- transformarea expresiei într-o formă fără paranteze;
- evaluarea propriu-zisă a expresiei.

Algoritmii de rezolvare ale celor trei subprobleme îi vom descrie separat în trei subalgoritmi.

a) Verificarea corectitudinii unei expresii

```
Subalgoritm Corectă(s):
  n ← lungimea şirului de caractere s
  p_deschise ← 0
```

```
  { variabilă auxiliară, la sfârşit valoarea ei se atribuie identificatorului de funcţie }
cor ← adevărat
dacă s[1]='(' atunci                    { primul caracter îl tratăm separat }
  p_deschise ← p_deschise + 1
sfârşit dacă
dacă s[1] = ')' atunci
  cor ← fals
sfârşit dacă
                                  { să nu înceapă sau să nu se termine cu operator }
dacă (s[1] este operator) sau (s[n] este operator) atunci
  cor ← fals
sfârşit dacă
i ← 2
cât timp (i ≤ n) şi cor execută:
  dacă s[i] nu este literă şi nu este operator şi nu este paranteză atunci
    cor ← fals
  sfârşit dacă


                        { dacă numele unei variabile este format din mai multe litere }
  dacă (s[i] este literă) şi (s[i-1] este literă) atunci
    cor ← fals
  sfârşit dacă
          { sau dacă există doi operatori ce se succed, expresia nu este corectă }
  dacă (s[i] este operator) şi (s[i-1] este operator) atunci
    cor ← fals
  sfârşit dacă
  dacă s[i] = '(' atunci          { se numără parantezele când se deschid }
    p_deschise ← p_deschise + 1
    { dacă înaintea unei '(' avem o variabilă sau ')', expresia nu este corectă }
    dacă (s[i-1] este literă) sau (s[i-1] = ')') atunci
      cor ← fals
    sfârşit dacă
  sfârşit dacă
  dacă s[i]=')' atunci                  { se închide o paranteză deschisă }
    dacă p_deschise ≤ 0 atunci          { dacă se închide una nedeschisă }
      cor ← fals
    altfel
      p_deschise ← p_deschise - 1
    sfârşit dacă
                          { dacă în faţa unei paranteze închise este un operator }
```

```
      dacă s[i-1] este operator atunci
        cor ← fals
      sfârşit dacă
    sfârşit dacă
    i ← i + 1
  sfârşit cât timp
  corectă ← cor
sfârşit subalgoritm
```

b) Aducerea expresiei la forma poloneză postfixată

Vom realiza o implementare care foloseşte cele două stive prezentate în suportul teoretic al lecţiei. În variabila *s* citim expresia dată cu paranteze, iar rezultatul se va construi în *rezult*, ambele variabile fiind de tip **string**. Cele două stive (*oper* şi *polon*) au vârfurile în *vfoper* şi *vfpolon*.

```
Subalgoritm Forma_postfixată(s,rezult):
  vfoper ← 0
  vfpolon ← 0
  mulţime ← {}          { formăm mulţimea numelor de variabile din expresie }
  pentru i=1,lungimea lui s execută: { prelucrăm fiecare caracter al expresiei }
    { caracterul literă sau paranteză deschisă se introduce în polon şi se adaugă la }
    { mulţimea de litere de care vom avea nevoie pentru a citi valorile variabilelor }
    dacă s[i] este literă atunci
      adaug(polon,vfpolon,s[i])                    { se adaugă în polon }
      mulţime ← mulţime ∪ s[i]
    sfârşit dacă
    dacă s[i] = '(' atunci
      adaug(oper,vfoper,s[i])                      { se adaugă în oper }
    sfârşit dacă
                                  { caracterul operator se introduce în oper }
                  { doi operatori consecutivi de aceeaşi prioritate se inversează }
    dacă (s[i] = '*') sau (s[i] = '/') sau (s[i] = '%') atunci
      dacă (vfoper ≠ 0) şi
             (prioritate(oper[vfoper]) = prioritate(s[i]) atunci
        Extrag(oper,vfoper,ce)
        Adaug(polon,vfpolon,ce)
      sfârşit dacă
      Adaug(oper,vfoper,s[i])
    sfârşit dacă
            { operatorii cu prioritate mică se mută din oper în polon până când }
          { în vârful stivei oper nu se va mai afla un operator '*', '/' sau '%' }
    dacă (s[i] = '+') sau (s[i] = '-') atunci
```

```
        { operatorii cu prioritate mai mare se scot din oper şi se introduc în polon }
      cât timp (vfoper ≠ 0) şi
          (Prioritate(oper[vfoper]) < prioritate(s[i])) execută:
        Extrag(oper,vfoper,ce)
        Adaug(polon,vfpolon,ce)
      sfârşit cât timp
      Adaug(oper,vfoper,s[i])              { se adaugă operatorul + sau – }
    sfârşit dacă
                { caracterul curent este paranteză închisă: se mută toţi operatorii }
       { din oper în polon până la întâlnirea parantezei deschise corespunzătoare }
                                    { paranteza deschisă se elimină din oper }
    dacă s[i] = ')' atunci          { expresia dintre două paranteze deschise }
                    { se mută în polon şi se elimină din oper paranteza deschisă }
      cât timp oper[vfoper] ≠ '(' execută:
        Extrag(oper,vfoper,ce)
        Adaug(polon,vfpolon,ce)
      sfârşit cât timp
      Extrag(oper,vfoper,ce)                          { elimină '(' }
    sfârşit dacă
  sfârşit pentru
                                  { tot ceea ce a rămas în oper se mută în polon }
  cât timp vfoper ≠ 0 execută:
    Extrag(oper,vfoper,ce)
    Adaug(polon,vfpolon,ce)
  sfârşit cât timp
sfârşit subalgoritm
```

c) Evaluarea unei expresii aflate în formă poloneză postfixată

Pentru început se construieşte un şir *v* din valorile variabilelor din expresie în ordinea în care apar în expresie. Poziţiile corespunzătoare operatorilor sau parantezelor se iniţializează cu valoarea 0.

Expresia fiind în formă poloneză postfixată, la fiecare pas căutăm primul operator pentru a efectua întâi acea operaţie.

În algoritm notăm cu *n* lungimea expresiei, deci şi a şirului *v*, iar în *rez* calculăm valoarea expresiei. Variabila logică *gata*, va primi valoarea *adevărat* în momentul în care s-a terminat calcularea valorii expresiei. După efectuarea unei operaţii se şterge unul dintre operanzi (de exemplu al doilea) precum şi operatorul efectuat, atât din expresie cât şi din şirul *v*. Valoarea calculată se aşează în şirul *v* pe poziţia corespunzătoare variabilei păstrate.

Când şirul evaluărilor s-a încheiat avem o singură variabilă a cărei valoare este valoarea expresiei date. Observăm că identificatorul variabilei nu are nici o semnificaţie pentru rezultat.

```
Subalgoritm valoare_expresie(s):                    { returnează un întreg }
  pentru i=1,n execută:
    dacă s[i] este literă atunci
      v[i] ← valoarea numerică a variabilei având identificatorul în s[i]
    altfel
      v[i] ← 0
    sfârşit dacă
  sfârşit pentru
  repetă          { căutăm primul operator pentru a efectua întâi acea operaţie }
    i ← 1
    cât timp (s[i] ≠ '+') şi (s[i] ≠ '*') şi (s[i] ≠ '-') şi
                 (s[i] ≠ '/') şi (s[i] ≠'%') si (i < n) execută:
      i ← i + 1
    sfârşit cât timp
                                                    { efectuăm operaţia }
    dacă s[i] = '+' atunci rez ← v[i-2] + v[i-1]
    altfel
      dacă s[i] = '-' atunci rez ← v[i-2] - v[i-1]
      altfel
        dacă s[i] = '*' atunci rez ← v[i-2]*v[i-1]
        altfel
          dacă s[i]='/' atunci rez ← [v[i-2]/v[i-1]]
          altfel
            dacă s[i] = '%' atunci rez ← rest[v[i-2]/v[i-1]]
            sfârşit dacă
          sfârşit dacă
        sfârşit dacă
      sfârşit dacă
    sfârşit dacă
    dacă (i = n) şi (s[i] este literă) atunci gata ← adevărat
                                       altfel gata ← fals
    dacă nu gata atunci
     { ştergem unul dintre operanzi şi operatorul efectuat din expresie şi din şirul v }
      se şterg din s două caractere începând cu poziţia i-1
                { valoarea calculată devine în şirul v valoarea variabilei păstrate }
      v[i-2] ← rez
                          { ştergem din v două elemente începând cu poziţia i-1 }
      pentru k=i-1,n execută:
        v[k] ← v[k+2]
        n ← n - 2
      sfârşit pentru
```

```
    sfârşit dacă
  până când gata
                { am rămas cu o singură variabilă care conține valoarea expresiei }
  valoare_expresie ← v[1]
sfârşit algoritm
```