Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare

FORMA POSTFIXATĂ A UNEI EXPRESII

Introducere

- Scrierea postfixată (forma poloneză inversă) a fost realizată de matematicianul polonez Jan Lukasiewicz (1878 – 1956)
- Algoritmul de transformare a unei expresii matematice din forma infixată în forma postfixată a fost elaborat de informaticianul olandez Edsger Dijkstra (1930 – 2002)

Introducere

- Transformarea unei expresii aritmetice din forma infixată în forma postfixată necesită utilizarea unei stive
- După aceea, se evaluează expresia în forma postfixată
- Această operație este necesară la proiectarea compilatoarelor

Notația infixată

- Expresiile aritmetice sunt scrise în mod uzual cu operatorii (+, -, * sau /) așezați între cei doi operanzi (numere sau simboluri care înlocuiesc numerele)
- Această notație se numește **infixată**, din cauză că operatorul apare între operanzi
- Exemple: 2+2, 4/7, A+B, A/B

Notația postfixată

- În notația **postfixată**, operatorul urmează cei doi operanzi
- A+B devine AB+
- A/B devine AB/
- Orice expresie infixată, oricât de complexă, poate fi transcrisă în notație postfixată

TABLE 4.2 Infix and Postfix Expressions

Infix	Postfix
A+B–C	AB+C–
A*B/C	AB*C/
A+B*C	ABC*+
A*B+C	AB*C+
A*(B+C)	ABC+*
A*B+C*D	AB*CD*+
(A+B)*(C–D)	AB+CD–*
((A+B)*C)–D	AB+C*D–
A+B*(C–D/(E+F))	ABCDEF+/–*+

Notația prefixată

- Pe langă notațiile infixată și postfixată, mai există și notația **prefixată**, în care operatorii se scriu înaintea operanzilor
- +AB în loc de AB+
- Funcțional, această notație e similară cu cea postfixată

Evaluarea unei expresii infixate

- Se citește expresia de la stânga la dreapta
- Atunci când se citește suficient pentru a evalua doi operanzi și un operator, se efectuează calculul și se înlocuiesc cei doi operanzi și operatorul dintre ei cu rezultatul gasit
- Procesul continuă, mergând de la stânga spre dreapta și evaluând, acolo unde este posibil, până la sfârșitul expresiei

TABLE 4.3 Evaluating 3+4–5

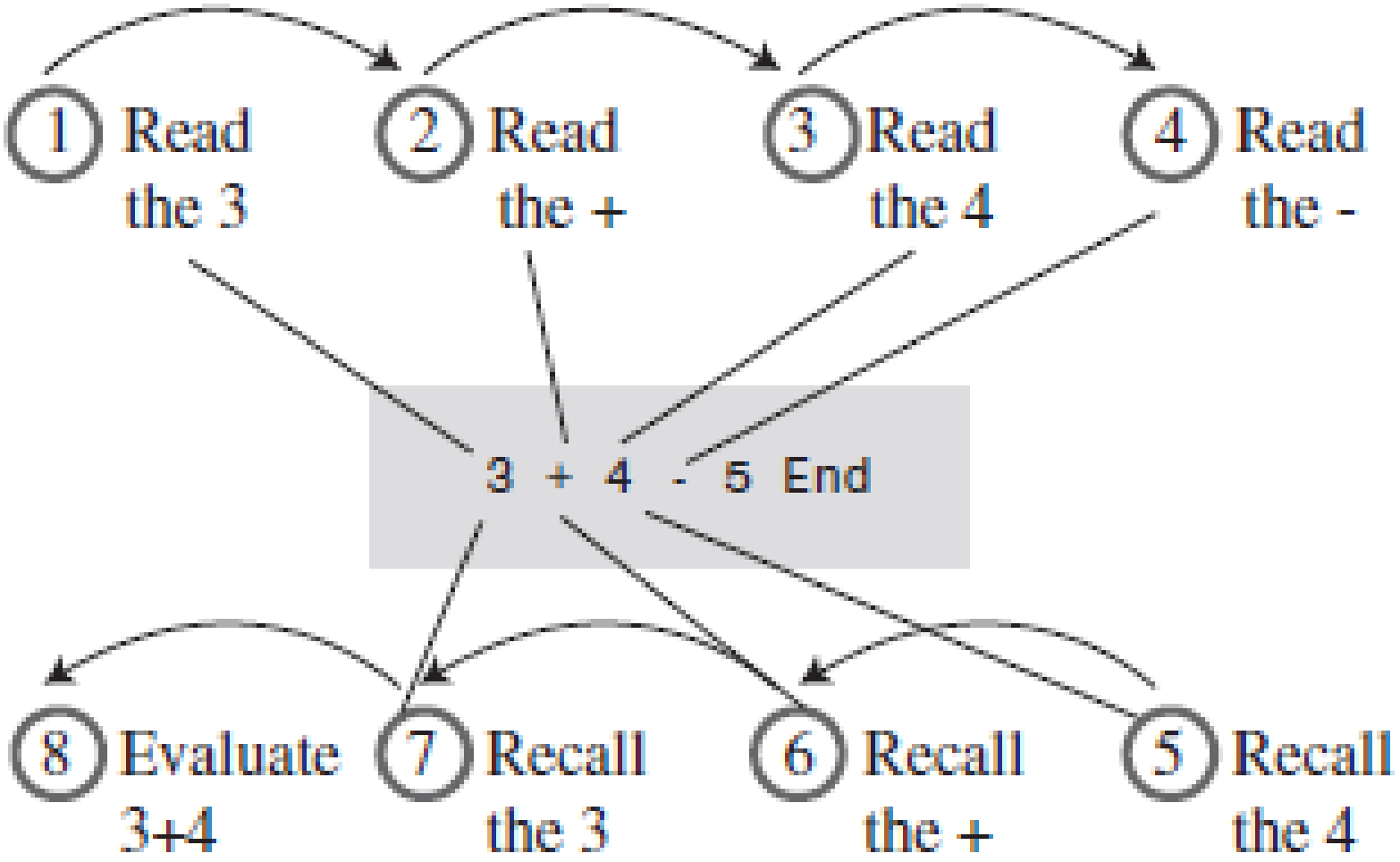
Item Read	Expression Parsed So Far	Comments
3	3	
+	3+	
4	3+4	
–	7	When you see the –, you can evaluate 3+4.
	7–	
5	7–5	
End	2	When you reach the end of the expression, you can evaluate 7–5.

Observații

- Nu putem evalua 3+4 până când nu vedem care este operatorul care urmează imediat după 4
- Dacă este vorba de un operator * sau /, trebuie să așteptăm înainte de a aplica efectul semnului + până când se evaluează * sau /

Observații

- Operatorul care urmează după 4 este -, care are aceeași precedență cu +, deci atunci când îl întâlnim, știm că putem evalua 3+4, care este 7
- Rezultatul 7 va înlocui secvența 3+4
- Se va evalua apoi 7-5, atunci când ajungem la sfârșitul expresiei


1 Read the 3
2 Read the +
3 Read the 4
4 Read the -
3 + 4 - 5 End
8 Evaluate 3+4
7 Recall the 3
6 Recall the +
5 Recall the 4

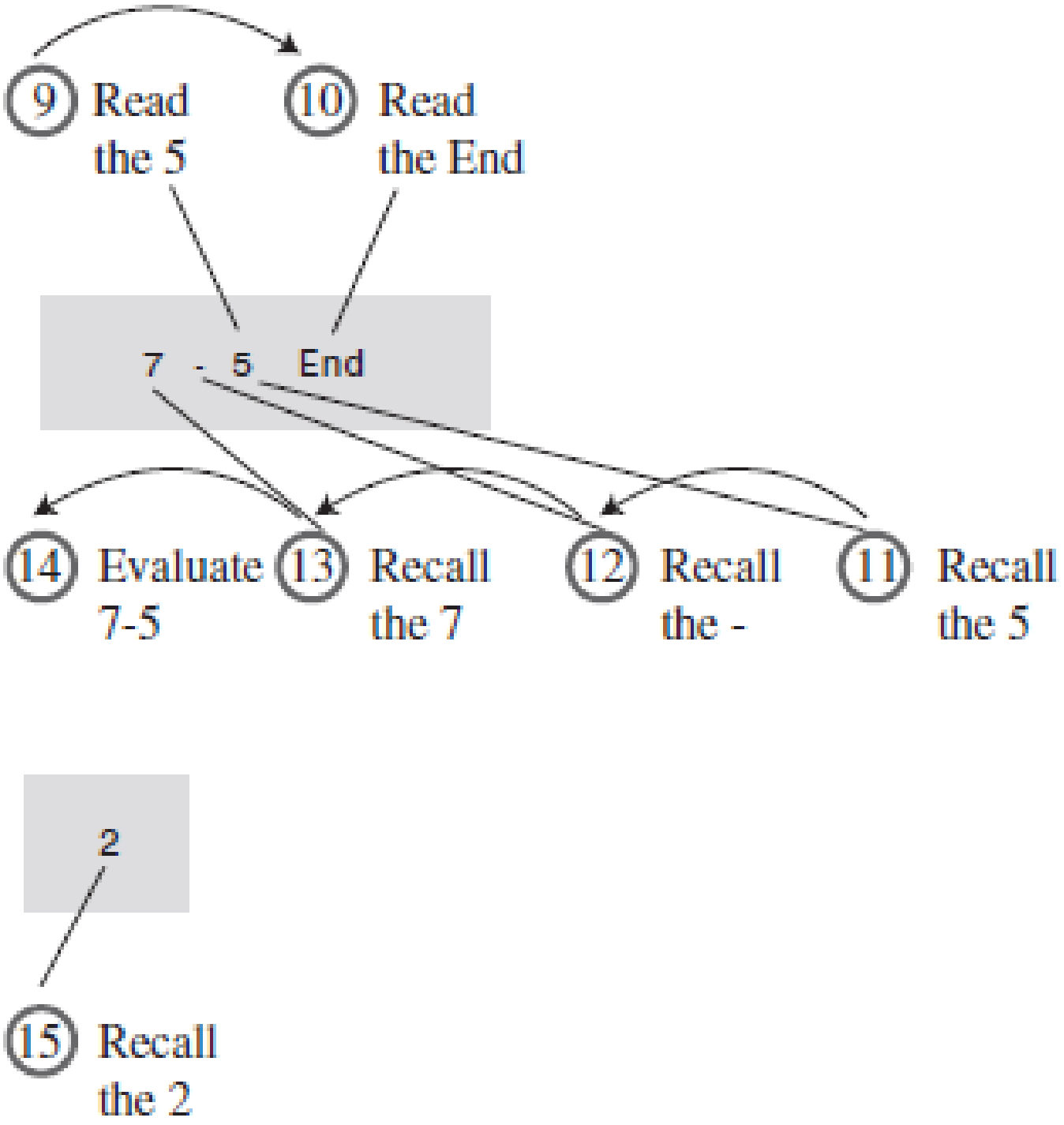

9 Read the 5
10 Read the End
7 - 5 End
14 Evaluate 7-5
13 Recall the 7
12 Recall the -
11 Recall the 5
2
15 Recall the 2

TABLE 4.4 Evaluating 3+4*5

Item Read	Expression Parsed So Far	Comments
3	3	
+	3+	
4	3+4	
*	3+4*	You can't evaluate 3+4 because * is higher precedence than +.
5	3+4*5	When you see the 5, you can evaluate 4*5.
	3+20	
End	23	When you see the end of the expression, you can evaluate 3+20.

Observații

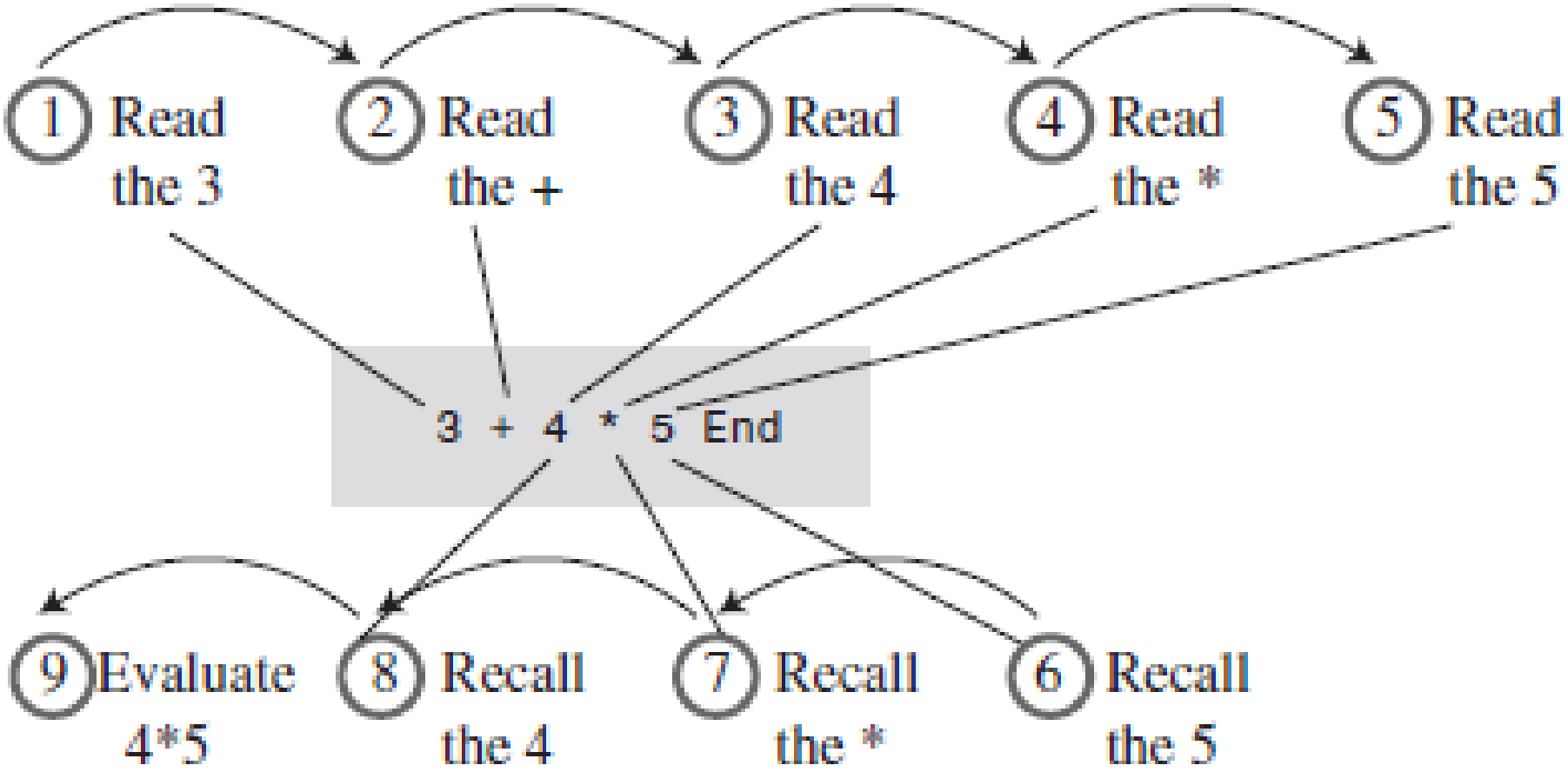
- Nu se poate însuma cu 3 decât atunci când cunoaștem rezultatul înmulțirii 4*5, deoarece înmulțirea are o precedență mai mare decât adunarea
- Toate înmulțirile sau împărțirile trebuie efectuate înaintea oricăror adunări sau scăderi, ce excepția cazului în care parantezele schimbă această ordine

Observații

- Trebuie să ne asigurăm, atunci când ajungem la o combinație operand-operator-operand, cum este A+B, că operatorul aflat la dreapta lui B nu are precedența mai mare decât +
- Dacă operatorul respectiv are o precedență mai mare, nu putem încă să efectuăm adunarea

Observații

- După ce l-am citit pe 5, înmulțirea poate fi efectuată, din cauză că are prioritate maximă
- Nu contează dacă un alt operator * sau / mai urmează după 5
- Nu putem efectua încă adunarea, până când nu știm ce se află la dreapta lui 5
- Când observăm că, după 5, expresia se termină, putem continua efectuând adunarea


1 Read the 3
2 Read the +
3 Read the 4
4 Read the *
5 Read the 5
3 + 4 * 5 End
9 Evaluate 4*5
8 Recall the 4
7 Recall the *
6 Recall the 5

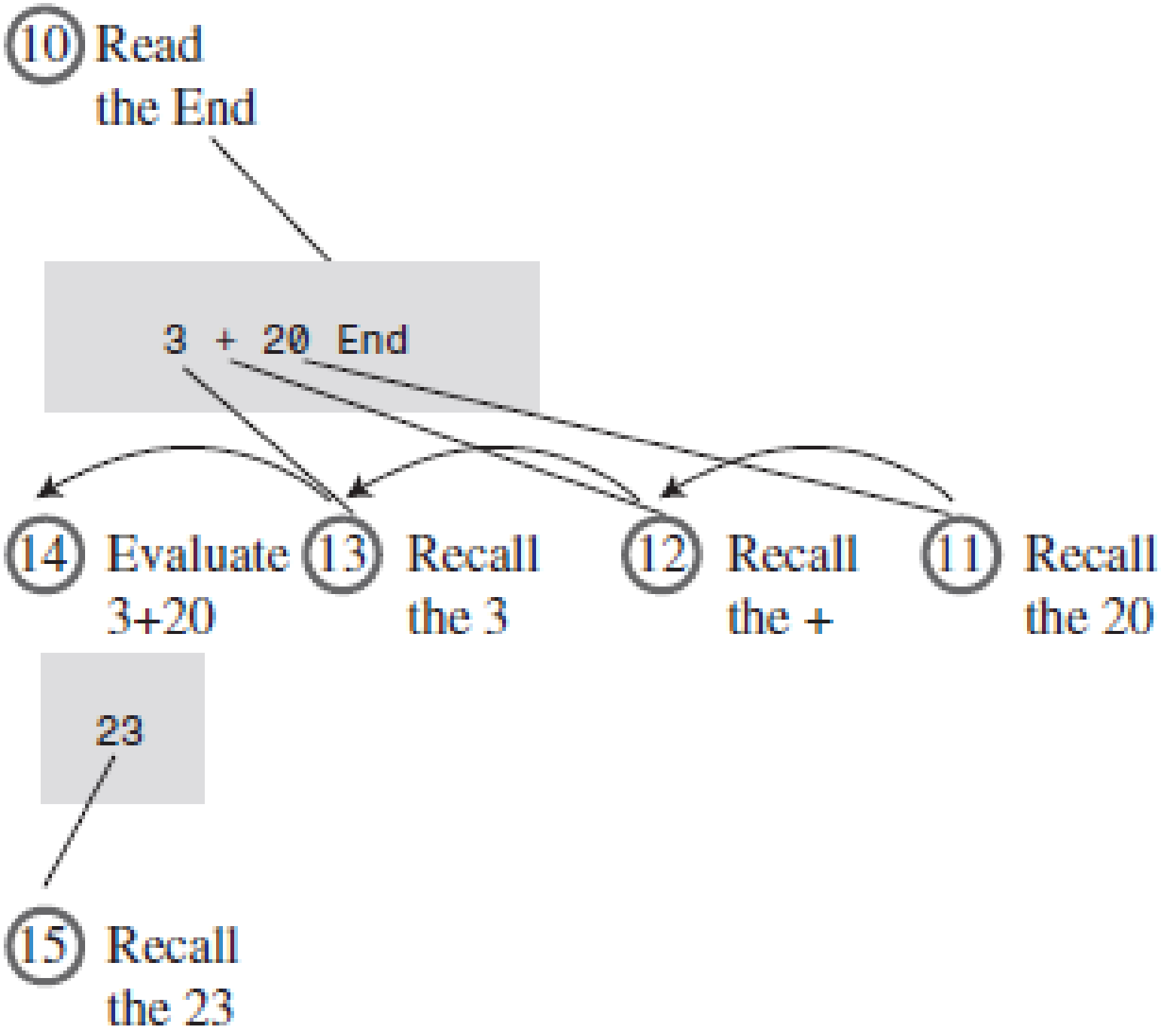

10 Read the End
3 + 20 End
14 Evaluate 3+20
13 Recall the 3
12 Recall the +
11 Recall the 20
23
15 Recall the 23

TABLE 4.5 Evaluating 3*(4+5)

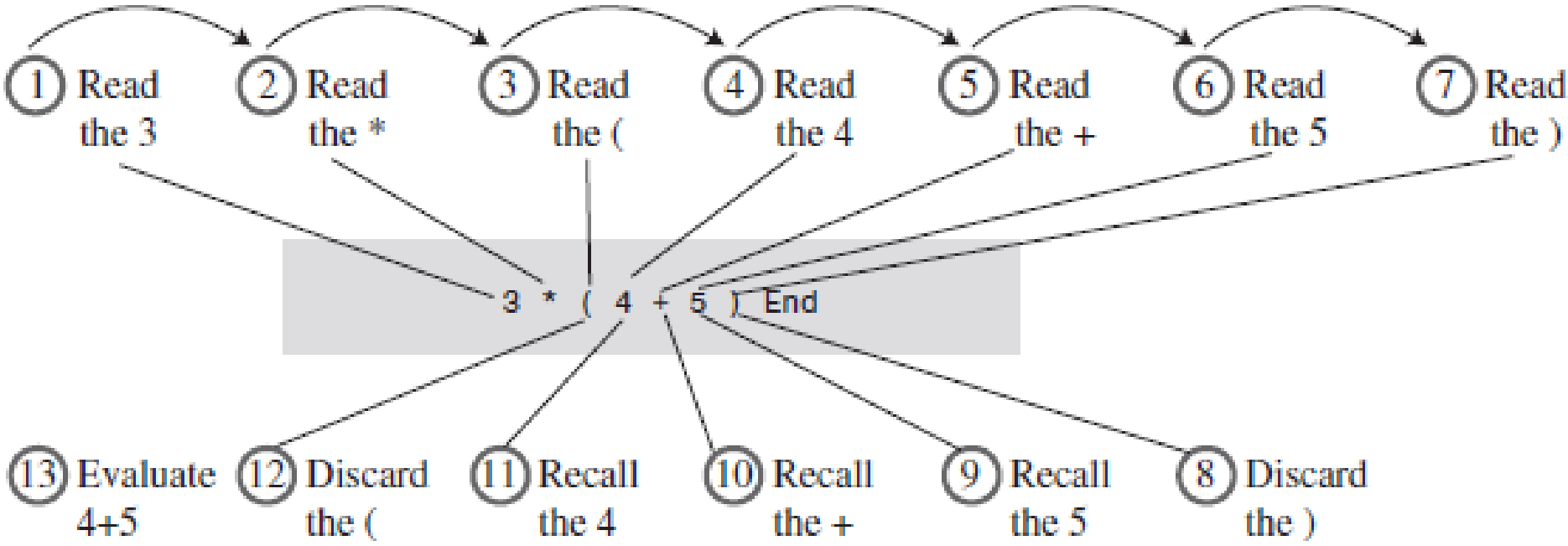
Item Read	Expression Parsed So Far	Comments
3	3	
*	3*	
(	3*(	
4	3*(4	You can't evaluate 3*4 because of the parenthesis.
+	3*(4+	
5	3*(4+5	You can't evaluate 4+5 yet.
)	3*(4+5)	When you see the), you can evaluate 4+5.
	3*9	After you've evaluated 4+5, you can evaluate 3*9.
	27	
End		Nothing left to evaluate.

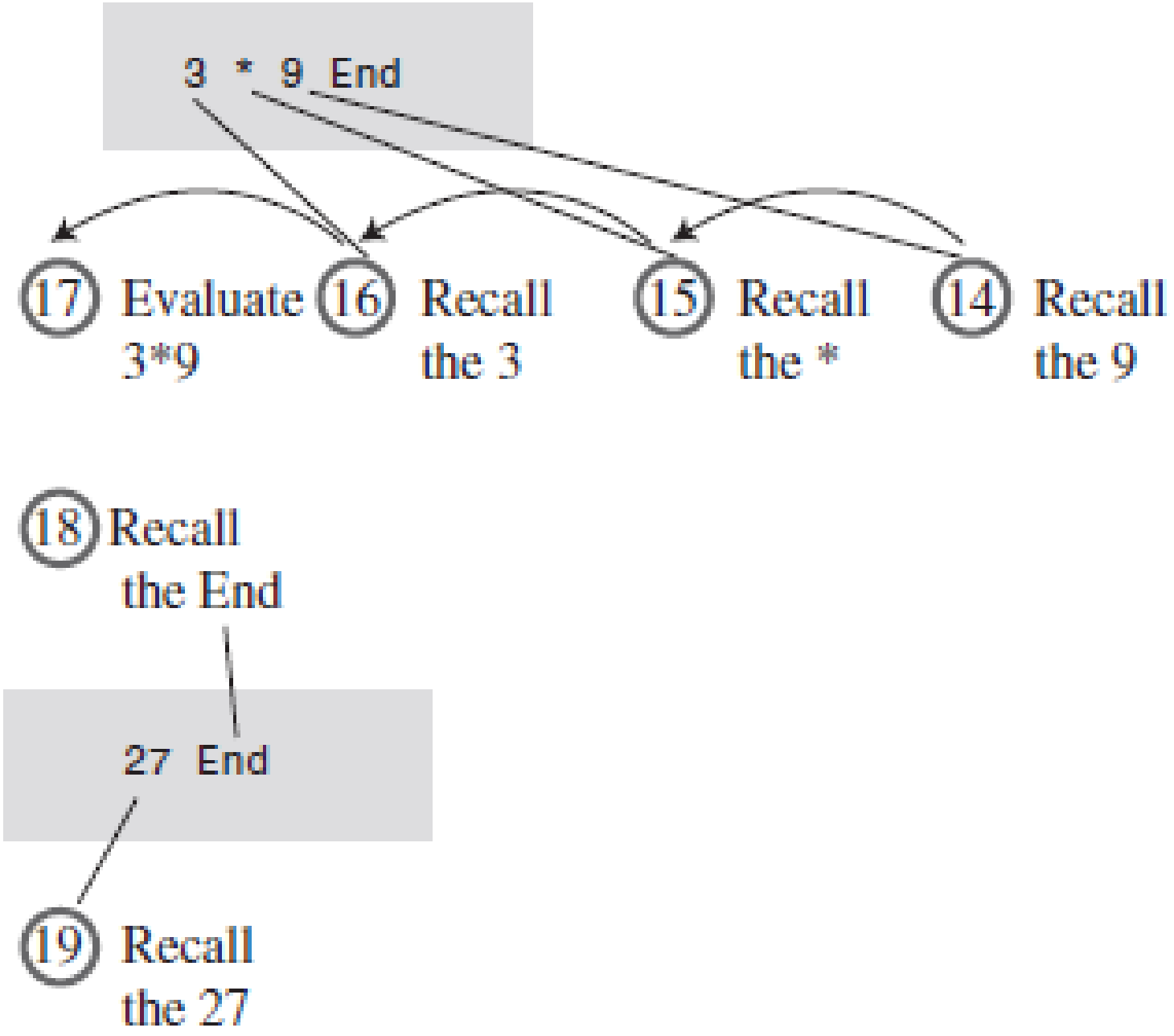
Observații

- Parantezele se utilizează pentru a modifica precedența obișnuită a operatorilor
- În expresia 3*(4+5), fără paranteze, se efectuează mai întâi înmulțirea
- Cu paranteze, adunarea este cea care se va efectua prima

Observații

- Nu se poate evalua nimic până când nu se întâlnește paranteza închisă
- Se poate calcula 3*4 imediat după ce îl întâlnim pe 4, dar parantezele au o precedență mai mare decât operatorii * și /
- Putem evalua tot ce este cuprins între paranteze, înainte de a putea utiliza rezultatul ca operand în alte calcule


1 Read the 3
2 Read the *
3 Read the (
4 Read the 4
5 Read the +
6 Read the 5
7 Read the)
3 * (4 + 5) End
13 Evaluate 4+5
12 Discard the (
11 Recall the 4
10 Recall the +
9 Recall the 5
8 Discard the)


3 * 9 End
17 Evaluate 3*9
16 Recall the 3
15 Recall the *
14 Recall the 9
18 Recall the End
27 End
19 Recall the 27

Observații

- La evaluarea expresiilor aritmetice în notație infixată, ne deplasăm atât înainte cât și înapoi în cadrul expresiei
- Deplasările înainte (de la stânga la dreapta) au ca scop citirea operanzilor și a operatorilor
- Când avem suficientă informație pentru a aplica un operator, ne deplasăm înapoi, amintindu-ne doi operanzi și un operator, pentru a efectua calculele aritmetice

Observații

- Uneori, trebuie să amânăm aplicarea unor operatori, dacă sunt urmați de operatori cu precedență mai mare sau de paranteze
- Atunci când apare această situație, trebuie mai întâi să aplicăm operatorul ulterior, de precedență mai mare
- După aceea, ne deplasăm înapoi (spre stânga) și aplicăm primul operator

Conversia formei infixate în notație postfixată

- Ideea nu este de a evalua expresia în formă infixată, ci de a rearanja operatorii și operanzii într-un format diferit: notația postfixată
- Expresia postfixată rezultată va fi evaluată ulterior
- Se citește forma infixată de la stânga spre dreapta, examinând fiecare caracter în parte

Observații

- Pe măsură ce ne deplasăm, vom copia operanzii și operatorii în șirul de ieșire, care reprezintă chiar notația postfixată
- Artificiul constă în a ști când să copiem fiecare element în parte
- Când caracterul din șirul infixat este un operand, îl vom copia imediat în șirul postfixat

Observații

- Ori de câte ori putem utiliza un operator pentru a evalua o parte a expresiei infixate (dacă am fi efectuat evaluarea expresiei în loc de conversia la forma postfixată), vom copia operatorul în șirul postfixat

TABLE 4.6 Translating A+B–C into Postfix

Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Comments
A	A	A	
+	A+	A	
B	A+B	AB	
–	A+B–	AB+	When you see the –, you can copy the + to the postfix string.
C	A+B–C	AB+C	
End	A+B–C	AB+C–	When you reach the end of the expression, you can copy the –.

TABLE 4.7 Translating A+B*C to Postfix

Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Comments
A	A	A	
+	A+	A	
B	A+B	AB	
*	A+B*	AB	You can't copy the + because * is higher precedence than +.
C	A+B*C	ABC	When you see the C, you can copy the *.
	A+B*C	ABC*	
End	A+B*C	ABC*+	When you see the end of the expression, you can copy the +.

TABLE 4.8 Translating A*(B+C) into Postfix

Character Read from Infix Expression	Infix Expression Parsed so Far	Postfix Expression Written So Far	Comments
A	A	A	
*	A*	A	
(	A*(	A	
B	A*(B	AB	You can't copy * because of the parenthesis.
+	A*(B+	AB	
C	A*(B+C	ABC	You can't copy the + yet.
)	A*(B+C)	ABC+	When you see the), you can copy the +.
	A*(B+C)	ABC+*	After you've copied the +, you can copy the *.
End	A*(B+C)	ABC+*	Nothing left to copy.

Observații

- La fel ca în procesul evaluării numerice, ne deplasăm atât înainte, cât și înapoi, prin expresia infixată, pentru a realiza conversia la notația postfixată
- Nu putem scrie un operator în șirul de ieșire (forma postfixată), dacă acesta este urmat de un operator cu precedență mai mare sau de o paranteză deschisă

Observații

- Dacă întâlnim o astfel de situație, operatorul de precedență mai mare sau operatorul din paranteză trebuie copiat în forma postfixată, înaintea operatorului de prioritate mai scăzută

Salvarea operatorilor într-o stivă

- Ordinea operatorilor este inversată atunci când trecem de la forma infixată la cea postfixată
- Din cauză că primul operator nu poate fi copiat în șirul de ieșire până când al doilea operator nu a fost copiat, operatorii apar la ieșire în ordine inversă față de cum au fost citiți din șirul infixat

TABLE 4.9 Translating A+B*(C–D) to Postfix

Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Stack Contents
A	A	A	
+	A+	A	+
B	A+B	AB	+
*	A+B*	AB	+*
(	A+B*(	AB	+*(
C	A+B*(C	ABC	+*(
–	A+B*(C–	ABC	+*(–
D	A+B*(C–D	ABCD	+*(–
)	A+B*(C–D)	ABCD–	+*(
	A+B*(C–D)	ABCD–	+*(
	A+B*(C–D)	ABCD–	+*
	A+B*(C–D)	ABCD–*	+
	A+B*(C–D)	ABCD–*+	

Observații

- În expresia infixată inițială, ordinea operatorilor este + * -, dar se inversează până la - * +, în forma postfixată rezultată
- Aceasta se întâmplă din cauză că operatorul * are o precedență mai mare decât +, iar -, fiind în paranteză, are precedență mai mare decât *

Observații

- Inversarea ordinii ne sugerează că este o idee bună să utilizăm o stivă pentru a memora operatorii, în timp ce așteptăm să îi putem utiliza efectiv
- Extragerea unor elemente din stivă ne permite ca, într-un fel, să ne deplasăm înapoi (de la dreapta spre stânga) prin șirul de intrare

Observații

- Nu examinăm întregul șir de intrare, ci doar operatorii și parantezele
- Aceștia au fost inserați în stivă atunci când s-a citit șirul de intrare, deci este simplu să ni-i amintim în ordine inversă, extrăgându-i efectiv din stivă
- Operanzii apar în aceeași ordine în cele două notații, deci ei pot fi scriși în șirul de ieșire imediat ce sunt întâlniți, nu este nevoie să-i memorăm într-o stivă

TABLE 4.10 Infix to Postfix Translation Rules

Item Read from Input (Infix)	Action
Operand	Write it to output (postfix)
Open parenthesis (	Push it on stack
Close parenthesis)	While stack not empty, repeat the following: Pop an item, If item is not (, write it to output Quit loop if item is (
Operator (`opThis`)	If stack empty, Push `opThis` Otherwise, While stack not empty, repeat: Pop an item, If item is (, push it, or If item is an operator (`opTop`), and If `opTop < opThis`, push `opTop`, or If `opTop >= opThis`, output `opTop` Quit loop if `opTop < opThis` or item is (Push `opThis`
No more items	While stack not empty, Pop item, output it.

Observații

- Simbolurile < și >= se referă la relația de precedență, nu la comparația unor valori numerice
- Operatorul `opThis` este cel care tocmai a fost citit din șirul de intrare, în timp ce `opTop` este cel care se află în vârful stivei

TABLE 4.11 Translation Rules Applied to A+B–C

Character Read from Infix	Infix Parsed So Far	Postfix Written So Far	Stack Contents	Rule
A	A	A		Write operand to output.
+	A+	A	+	If stack empty, push `opThis`.
B	A+B	AB	+	Write operand to output.
–	A+B–	AB		Stack not empty, so pop item.
	A+B–	AB+		`opThis` is –, `opTop` is +, `opTop>=opThis`, so output `opTop`.
	A+B–	AB+	–	Then push `opThis`.
C	A+B–C	AB+C	–	Write operand to output.
End	A+B–C	AB+C–		Pop leftover item, output it.

TABLE 4.12 Translation Rules Applied to A+B*C

Character Read From Infix	Infix Parsed So Far	Postfix Written So Far	Stack Contents	Rule
A	A	A		Write operand to postfix.
+	A+	A	+	If stack empty, push `opThis`.
B	A+B	AB	+	Write operand to output.
*	A+B*	AB	+	Stack not empty, so pop `opTop`.
	A+B*	AB	+	`opThis` is *, `opTop` is +, `opTop<opThis`, so push `opTop`.
	A+B*	AB	+*	Then push `opThis`.
C	A+B*C	ABC	+*	Write operand to output.
End	A+B*C	ABC*	+	Pop leftover item, output it.
	A+B*C	ABC*+		Pop leftover item, output it.

TABLE 4.13 Translation Rules Applied to A*(B+C)

Character Read From Infix	Infix Parsed So Far	Postfix Written So Far	Stack Contents	Rule
A	A	A		Write operand to postfix.
*	A*	A	*	If stack empty, push `opThis`.
(	A*(	A	*(	Push (on stack.
B	A*(B	AB	*(	Write operand to postfix.
+	A*(B+	AB	*	Stack not empty, so pop item.
	A*(B+	AB	*(	It's (, so push it.
	A*(B+	AB	*(+	Then push `opThis`.
C	A*(B+C	ABC	*(+	Write operand to postfix.
)	A*(B+C)	ABC+	*(	Pop item, write to output.
	A*(B+C)	ABC+	*	Quit popping if (.
End	A*(B+C)	ABC+*		Pop leftover item, output it.

Algoritmul de transformare în forma postfixată

- Se utilizează o stivă, în care sunt stocați temporar operatorii expresiei
- Fiecare operator are atribuită o prioritate

(	1
)	2
+ -	3
* /	4

Algoritmul de transformare în forma postfixată

- 1. Se inițializează stiva și structura de stocare a expresiei în forma postfixată, care este o coadă
- 2. Atât timp cât nu s-a ajuns la sfârșitul expresiei în forma infixată:
  - 2.1. Se citește următorul element (operand/operator) din expresia infixată

Algoritmul de transformare în forma postfixată

- 2.2. Dacă elementul este operand, acesta se adaugă în coadă
- 2.3. Dacă elementul este operatorul **(**, atunci acesta se introduce în stivă
- 2.4. Dacă elementul este operatorul **)**, atunci se transferă operatorii din stivă în coadă, până la întâlnirea operatorului **(** în stivă; operatorul **(** se extrage din stivă, fără a fi transferat în coadă

Algoritmul de transformare în forma postfixată

- 2.5. Altfel, nu este unul din operatorii **(** sau **)**:
  - 2.5.1. Atât timp cât prioritatea operatorului din vârful stivei este mai mare decât prioritatea operatorului curent, se trece operatorul din vârful stivei în coadă
  - 2.5.2. Se introduce operatorul curent în stivă
- 3. Se trec toți operatorii rămași pe stivă în forma postfixată

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>

struct NodStiva{
        char opr;
        NodStiva* next;
};
```

```
struct NodCoada{
        int opd;       //valoarea numerica a operandului
        char opr;      //simbolul asociat operatorului
        NodCoada* next;
};

/*Cele doua campuri int si char se exclud reciproc prin
initializarea cu valoarea zero a campului care nu trebuie
considerat in cadrul operatiei de extragere din coada
sau afisare a cozii*/
```

```
NodStiva* push(NodStiva* vf, char c){
NodStiva* nou = (NodStiva*) malloc(sizeof(NodStiva));
        nou->opr = c;
        nou->next = vf;
        return nou;
}
NodStiva* pop(NodStiva* vf, char *c){
        if (vf) {
                *c = vf->opr;
                NodStiva* t = vf;
                vf = vf->next;
                free(t);
                return vf;
        }
        return vf; }
```

```
NodCoada* put(NodCoada* c, int v, char o) {
NodCoada* nou=(NodCoada*) malloc(sizeof(NodCoada));
        nou->opd = v;
        nou->opr = o;
        nou->next = NULL;
        if (!c)
                return nou;
        else {
                NodCoada* t = c;
                while(t->next)
                        t = t->next;
                t->next = nou;
                return c;
        }
}
```

```
int prioritate(char c) {
        switch(c) {
                case '(':
                        return 1;
                case ')':
                        return 2;
                case '+':
                case '-':
                        return 3;
                case '*':
                case '/':
                        return 4;
                default:
                        return 5;
        }
}
```

```
int main() {
        NodStiva* stack = NULL;
        NodCoada* queue = NULL;
        char ExprInfix[100], SubExpr[100], o;
        int vb, vb_op = 0;

        printf("Introduceti expresia matematica in
                                forma infixata: ");
        scanf("%s", ExprInfix);
```

```
        /*algoritmul de transformare infixata -> postfixata
cu operanzi intregi fara semn*/
        int i = 0;
        //extragere expresie operand/operator
        while (ExprInfix[i] != '\0') {
                int k = 0;
                if (ExprInfix[i] > 47 && ExprInfix[i] < 58) {
//codurile ascii ale cifrelor de la 0 la 9 sunt de la 48 la 57
                while (ExprInfix[i] > 47 && ExprInfix[i] < 58) {
                                SubExpr[k] = ExprInfix[i];
                                k++;
                                i++;
                        }
                        SubExpr[k] = '\0';
                        vb = 1;
        }
```

```
else {
        SubExpr[k] = ExprInfix[i];
        SubExpr[k+1] = '\0';
        i++;
        vb = 0;
        }
if (vb) {
        o = 0;
        queue = put(queue, atoi(SubExpr), o);
        }
/*Valorile operanzilor se obtin prin aplicarea functiei
standard de conversie ASCII-to-int, respectiv atoi*/
```

```
else {
        if (SubExpr[0] == '(' ){
                                stack = push(stack, SubExpr[0]);
                        }
        else {
                 if (SubExpr[0]== ')' ) {
                                stack = pop(stack, &o);
                                while (o != '(' ) {
                                        queue = put(queue, 0, o);
                                        stack = pop(stack, &o);
                                        }
                                }
```

```
else {
        if (prioritate(SubExpr[0]) < 5) {
        if (stack) {
while (stack &&
prioritate(stack->opr) > prioritate(SubExpr[0])) {
                                stack = pop(stack, &o);
                                queue = put(queue, 0, o);
                                }
                        }
        stack = push(stack, SubExpr[0]);
        }
```

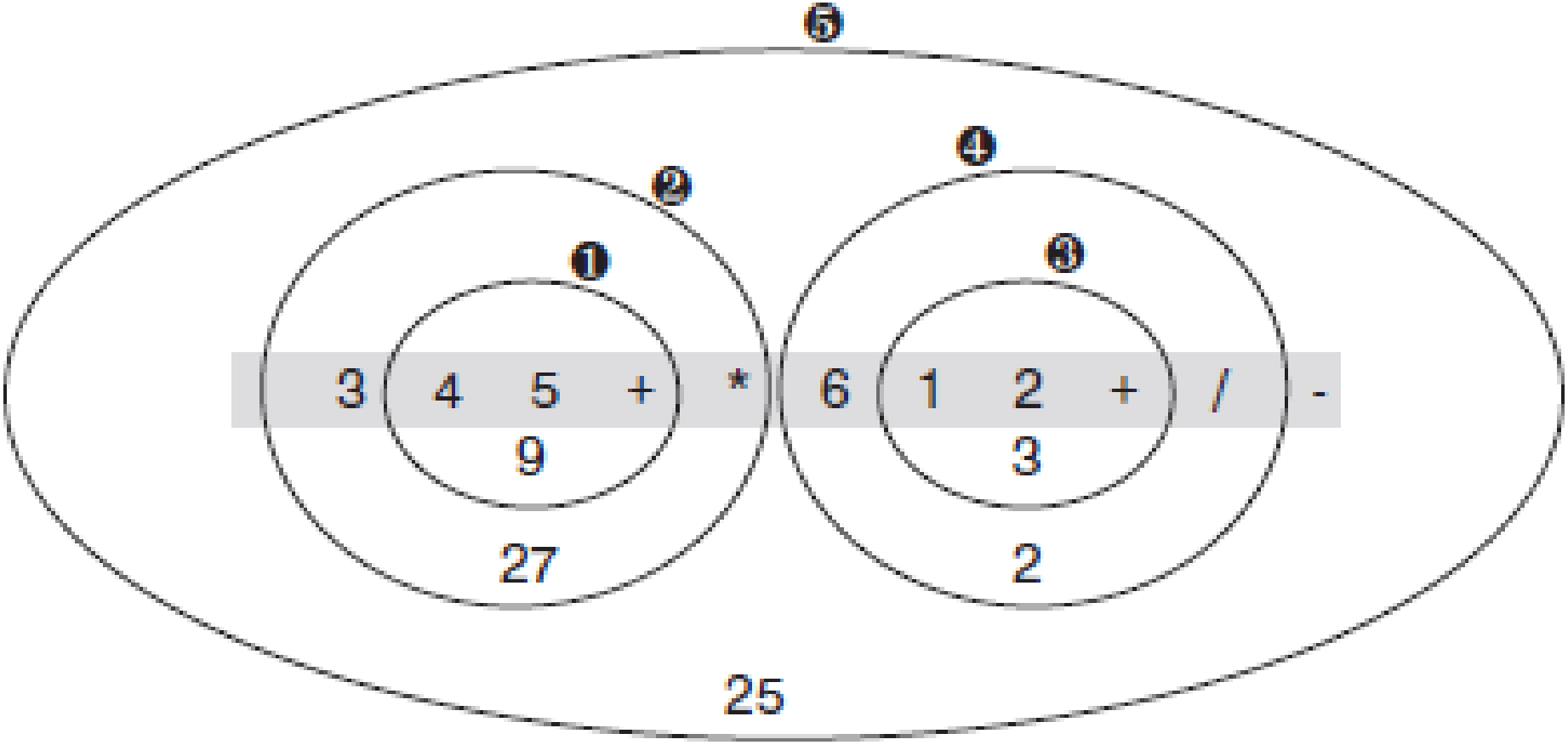
```
else {
        printf("Operator incorect introdus!");
        vb_op = 1;
                                        }
                                }
                        }
                }
        }
```

/*Daca in cadrul sirului de caractere introdus de la tastatura se identifica un operator sau simbol care nu se afla in lista precizata in functia prioritate, atunci variabila booleana ***vb_op*** este setata pe valoarea **1**. Expresia matematica in scriere postfixata este afisata doar pentru ***vb_op = 0****/

```
while (stack) {
        stack = pop(stack, &o);
        queue = put(queue, 0, o);
    }
    //afisarea expresiei in scriere postfixata
    NodCoada* t;
    if (!vb_op) {
        t = queue;
        while (t) {
            if (t->opd) printf("%d ", t->opd);
            else printf("%c ", t->opr);
            t = t->next;
        }
    }
```

```
//dealocare memorie alocata pentru coada
        while (queue) {
                t = queue;
                queue = queue->next;
                free(t);
        }
        getch();
}
```

Evaluarea expresiilor postfixate



Evaluarea expresiilor postfixate

- Se începe cu primul operator din stânga și se încercuiește, împreună cu cei mai apropiați doi operanzi aflați la stânga sa
- Se aplică operatorul asupra celor doi operanzi, efectuând calculul propriu-zis, și se scrie rezultatul în interiorul cercului

Evaluarea expresiilor postfixate

- Ne deplasăm spre dreapta, la operatorul imediat următor și încercuim operatorul, cercul anterior desenat și operandul aflat la stânga acestuia
- Se aplică operatorul între rezultatul încercuit anterior și noul operand, iar noul rezultat obținut se scrie în cercul nou desenat
- Procesul continuă până când se aplică toți operatorii

Reguli de evaluare a expresiilor postfixate

- De fiecare dată când întâlnim un operator, îl aplicăm asupra ultimilor doi operanzi întâlniți
- Aceasta ne sugerează ideea de a memora operanzii într-o stivă

Reguli de evaluare a expresiilor postfixate

- Este o situație inversă celei din algoritmul de conversie din forma infixată în cea postfixată, în care operatorii erau cei memorați în stivă
- La sfârșit, extragem ultimul element din stivă pentru a găsi valoarea expresiei

TABLE 4.14 Evaluating a Postfix Expression

Item Read from Postfix Expression	Action
Operand	Push it onto the stack.
Operator	Pop the top two operands from the stack and apply the operator to them. Push the result.

Algoritmul de evaluare a unei expresii postfixate

- 1. Se inițializează stiva
- 2. Atât timp cât nu s-a ajuns la sfârșitul expresiei postfixate, stocată într-o coadă
  - 2.1. Se extrage următorul element din expresia postfixată
  - 2.2. Dacă elementul extras este valoare, atunci acesta se pune pe stivă

Algoritmul de evaluare a unei expresii postfixate

- 2.3. Altfel, dacă elementul este operator, atunci:
  - 2.3.1. Se extrage din vârful stivei elementul y
  - 2.3.2. Se extrage din vârful stivei elementul x
  - 2.3.3. Se efectuează operația x operator y
  - 2.3.4. Se pune rezultatul pe stivă

- 3. Ultima valoare extrasă de pe stivă reprezintă rezultatul evaluarii expresiei

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct NodStiva{
        int val;
        NodStiva* next;
};
struct NodCoada{
        int val;
        char op;
        NodCoada* next;
};
```

```
NodStiva* push(NodStiva* vf, int v) {
NodStiva* nou = (NodStiva*) malloc(sizeof(NodStiva));
        nou->val = v;
        nou->next = vf;
        return nou;
}
NodStiva* pop(NodStiva* vf, int *v) {
        if (vf) {
                *v = vf->val;
                NodStiva* t = vf;
                vf = vf->next;
                free(t);
                return vf;
        }
        return vf;
}
```

```
NodCoada* put(NodCoada* c, int v, char o) {
NodCoada* nou=(NodCoada*) malloc(sizeof(NodCoada));
        nou->val = v;
        nou->op = o;
        nou->next = NULL;
        if (!c)
                return nou;
        else {
                NodCoada* t = c;
                while (t->next)
                        t = t->next;
                t->next = nou;
                return c;
        }
}
```

```
NodCoada* get(NodCoada* c, int *v, char* o) {
        if (c) {
                *v = c->val;
                *o = c->op;
                NodCoada* t = c;
                c = c->next;
                free(t);
                return c;
        }
        return c;
}
```

```
int main() {
        NodStiva* stack = NULL;
        NodCoada* queue = NULL;
        int opd;
        char opr, opt = 'd';

        //initializarea scrierii postfixate
        while (opt == 'd') {
                int i;
                printf("Operand/Operator?(1/0) ");
                scanf("%d", &i);
```

```
if (i==1) {
                printf("Valoarea operandului: ");
                scanf("%d", &opd);
                opr = 0;
                queue = put(queue, opd, opr);
                }
else
                if (i==0) {
                        printf("Simbol operator:('+'/'-'/'*'/'/') ");
                        opr = getche();
                        printf("\n");
                        opd = 0;
                        queue = put(queue, opd, opr);
                        }
```

```
else
        printf("Eroare la introducere!\n");
        printf("Continuati?(d/n) ");
        opt = getche();
        printf("\n");
}
```

```
//algoritmul de evaluare a expresiei
while (queue) {
        queue = get(queue, &opd, &opr);
        if (!opr) {
                stack = push(stack, opd);
        }
        else {
                int opd1, opd2, rez;
                stack = pop(stack, &opd2);
                stack = pop(stack, &opd1);
```

```
switch(opr) {
        case '+':
                rez = opd1 + opd2;
                break;
        case '-':
                rez = opd1 - opd2;
                break;
        case '*':
                rez = opd1 * opd2;
                break;
        case '/':
                rez = opd1 / opd2;
                break;
        default:
                printf("Operator gresit!\n");
    }
```

```
        stack = push(stack, rez);
                }
        }

stack = pop(stack, &opd);
printf("Rezultatul evaluarii expresiei aritmetice
                                este: %d", opd);
getch();
}
```