Олександр Рудик

Зворотний польський запис

1. Означення і алгоритм породження

Звичною формою запису виразів є *інфіксна*, коли знак бінарної операції записують *між* позначеннями операндів цієї операції, наприклад, $a + b$. Розглянемо запис знаків операцій *після* позначень операндів, тобто *постфіксний* запис, наприклад, $a\ b\ +$. Такий запис має також назву *зворотного польського*, бо його запропонував польський логік Ян Лукасевич. Далі словосполучення: «зворотний польський запис» позначатимемо ЗПЗ. Позначення для функції традиційно записують *перед* аргументами. Природно такий запис назвати *префіксним*. При описі ЗПЗ переважно обмежуються перетворенням інфіксного запису у ЗПЗ. Далі розглянуто узагальнення — перетворення у ЗПЗ арифметичних виразів з традиційним записом функції: аргументи записано після позначення функції у дужках через кому.

Означення 1. *Зворотний польський запис (ЗПЗ) — це*:

- *або запис позначення сталої чи змінної;*
- *або скінченний запис такого вигляду: < ЗПЗ > < ЗПЗ > < знак бінарної операції >*;
- *або скінченний запис такого вигляду: n < ЗПЗ > < позначення функції n змінних >.*

Можна дати еквівалентне означення зворотного польського запису через *рекурсивно задану* операцію P перетворення звичайного інфіксного запису у зворотний польський запис.

Нехай:

x — позначення сталої чи змінної;

X , Y, X_1 , X_2, ... , X_n — інфіксні записи;

$\odot$ — знак бінарної операції (наприклад, один зі знаків арифметичних дій: +, – , · або /);

f — позначення функції n змінних.

Маємо:

(P1) $P(x) = x$;

(P2) $P(X \odot Y) = P(X)\ P(Y)\ \odot$;

(РЗ) $P(f(X_1, X_2, \ldots, X_n)) = P(X_1)\, P(X_2) \ldots P(X_n)\, f$.

Наприклад, $P(a \cdot b + c/d - e) = a\ b \cdot c\ d / + e -$.

Зворотний польський запис має чудові властивості, які перетворюють її на ідеальну проміжну ланку при трансляції коду програми.

Обчислення виразу, записаного в зворотному польському записі, можна проводити шляхом однократного перегляду ЗПЗ.

Зворотний польський запис виразу з арифметичними діями та піднесенням до степеня можна отримати, дотримуючись алгоритму, запропонованого Дейкстрою. Для цього запроваджують поняття стекового пріоритету символів (див. табл. 1, у якій ^ є позначенням для піднесення у степінь).

Таблиця 1

Пріоритет операцій для отримання зворотного польського запису

Пріоритет	**Операція**
0	(
1	)
2	+ або –
3	· або /
4	^

Проглядаючи послідовність символів інфіксного запису, операнди записуємо у вихідний файл у порядку зустрічі, а знаки операцій і дужки заносимо у стек, дотримуючись таких правил:

- якщо стек порожній, то символ записуємо у стек;
- символ «виштовхує» зі стека всі попередні символи з більшим або однаковим пріоритетом у вихідний файл;
- якщо черговий символ інфіксного запису є дужкою (, що відкриває, то заносимо її у стек;
- дужка, що закриває, «виштовхує» всі операції зі стека до найближчої дужки, що відкриває, а самі дужки у вихідний файл не записуємо;
- по завершенні перегляду інфіксного запису всі символи стеку записуємо у вихідний файл.

Приклад виконання такого алгоритму подано таблицею 2.

Таблиця 2

Процес отримання зворотного польського запису виразу (*a* + *b*) · (*c* + *d*) – *e*

Символ	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Введення	(	a	+	b	)	*	(	c	+	d	)	-	e	

Стан стека	(	(	+(	+(		*	(*	(*	+(*	+(*	*	-	-	
Виведення		a		b	+			c		d	+	*	e	-

Подамо програму мовою Turbo Pascal для отримання зворотного польського запису.

```
{$I-}
const v=['a'..'e','g'..'z','0'..'9','.',
         'A'..'E','G'..'Z'];
      ns_=1000;                          {розмір стеку}
var   fin,                               {вхідний файл}
      fout: text;                       {вихідний файл}
        ch: char;                     {зчитаний символ}
        ns,                {кількість елементів стеку}
        is: word;          {лічильник елементів стеку}
      code: integer;       {код перетворення
                             числа аргументів функції}
s: array [0..1000] of string;         {елементи стеку
                                  у зворотному порядку}
p: array [0..1000] of byte;   {пріоритет елементів s}

            {Визначення пріоритету бінарної операції}
function prior(ch: char): byte;  BEGIN
                        case ch of
                '(': prior:=0;
                ')': prior:=1;
            '+','-': prior:=2;
            '*','/': prior:=3;
                '^': prior:=4; end END;
                                                BEGIN
assign(fin, 'in.txt');    reset(fin);
assign(fout,'out.txt');  rewrite(fout);
ns:=0;
REPEAT read(fin,ch);
CASE ch of
' ':;                           {ігнорування прогалини}

','                                          : begin
            if s[ns]<>'(' then
               repeat write(fout,s[ns],' '); dec(ns)
                until s[ns]='('                  end;

                   {Запис числа або позначення змінної}
'a'..'e','g'..'z','.','0'..'9',
'A'..'E','G'..'Z':                           begin
                               write(fout,ch);
             repeat read (fin, ch);
                    if   ch in v then write(fout,ch);
              until not (ch in v);
             write (fout,' ');
             case ch of
```

```
              '+','-','*','/','^'           : begin
                 if ns=0 then                 begin
                    ns:=1;
                    s[1]:=ch;
                    p[1]:=prior(ch)             end
               else                          begin
       {виштовхування операцій з неменшим пріоритетом}
                   p[0]:=prior(ch);
                   while (ns>0) and
                         (p[ns]>=p[0]) do     begin
                     write(fout,s[ns],' ');
                     dec(ns)                    end;
                   inc(ns);
                   p[ns]:=p[0];
                   s[ns]:=ch                end end;
              ',',')'                       : begin
                 if s[ns]<>'(' then
                    repeat write(fout,s[ns],' ');
                           dec(ns)
                    until s[ns]='(';
                    if ch=')' then dec(ns)  end end
                                               end;
'('                                          : begin
                 inc(ns);  s[ns]:=ch;  p[ns]:=0 end;

          {бінарні операції, записані між операндами}
'+','-','*','/','^'                          : begin
        if   ns=0 then     begin
             ns:=1;
             s[1]:=ch;
             p[1]:=prior(ch) end
        else                                  begin
       {виштовхування операцій з неменшим пріоритетом}
             p[0]:=prior(ch);
             while (ns>0) and (p[ns]>=p[0]) do begin
                   write(fout,s[ns],' ');
                   dec(ns)                    end;
           inc(ns);
           p[ns]:=p[0];
           s[ns]:=ch                       end end;
           {початок запису функції у форматі:
         f<кількість аргументів>_<літери або цифри>}
'f','F'                                      : begin
           s[0]:=ch;
           repeat read(fin,ch);  s[0]:=s[0]+ch;
            until ch='_';
           val(copy(s[0],2,length(s[0])-2),p[0],code);
           p[0]:=p[0]+4;
           if code<>0 then           begin
             writeln(fout);
             writeln(fout,'See ',s[0]);
             close(fout);
             halt                      end;
```

```
            repeat read(fin,ch);  s[0]:=s[0]+ch;
             until ch='(';
            delete(s[0],length(s[0]),1);
            while (ns>0) and (p[ns]>=p[0]) do  begin
                  write(fout,' ',s[ns]);
                  dec(ns)                          end;
            inc(ns);  s[ns]:=s[0];  p[ns]:=p[0];
            inc(ns);  s[ns]:='(';   p[ns]:=0     end;
')'                                          : begin
            repeat write(fout,s[ns],' ');
                   dec(ns)
             until s[ns]='(';
            dec(ns)                          end END
UNTIL seekeoln(fin);
                                {вивільнення стеку}
for is:=ns downto 2 do write(fout,s[is],' ');
writeln(fout,s[ns]);  close(fout);  close(fin)   END.
```

2. Породження дерева і виконання обчислень

З кожним зворотним польським записом Z природним чином пов'язане деяке орієнтоване дерево $T(Z)$, в якому кожна вершина поставлена у взаємно однозначну відповідність до фрагменту зворотного польського запису таким чином:

1. Позначенню x сталої або змінної відповідає вершина ⓧ, з якої не виходить жодної дуги (таку вершину називають *листком*);
2. Бінарній операції ⊙ відповідає вершина, з якої виходить дві дуги до тих вершин Ⓧ і Ⓨ, що відповідають операндам X і Y цієї операції;
3. Функції f відповідає вершина ⓕ, з якої виходить дуга до вершини Ⓧ, що відповідає аргументу X цієї функції.

Інакше кажучи, відображення T, як і P, задано рекурсивно:

(T1) $T(x) =$ ⓧ;

⊙

(T2) $T(X\,Y\,\odot) =$ ↙ ↘ ;

Ⓧ Ⓨ

ⓕ

(Т3) $T(Xf) =$ ↙ .

Ⓧ

Зауваження 1. *Для функції n змінних правило* (Т3) *потрібно замінити на таке, у якому з вершини* ⓕ *виходить n дуг.*

Обчислення згідно зі зворотним польським записом Z має наочне тлумачення у термінах отриманого дерева $T(Z)$: поступова заміна листків і дуг, що ведуть у ці листки з *однієї* вершини на листок, якому приписують результат. Отже, для обчислення згідно зі зворотним польським записом достатньо здійснити таке.

1. Побудувати дерево згідно з правилами (Т1), (Т2) і (Т3).
2. Здійснити обхід листків побудованого дерева, виконуючи заміну доти, поки дерево не перетвориться на одноелементну множину.

Побудова в оперативній пам'яті ПК орієнтованого дерева, що відповідає ЗПЗ, з використанням динамічного розподілу пам'яті вимагає детального опису алгоритму, який українською мовою далі не подано. Зате подано прокоментований код програми мовою Turbo Pascal для обчислення арифметичного виразу сталих величин (присутні лише бінарні операції — арифметичні дії). Кінці дуг з однаковим кінцем поділено на ліві й праві нащадки за місцем в інфіксному записі (див. поля `l`, `r` типу `b`, запровадженого для опису вершин дерева). Для аналізу коду програми рекомендуємо будувати графічні ілюстрації дій щодо створення нових вершин і дуг дерева на етапі його побудови та перетворення дерева на етапі обчислення величини арифметичного виразу. Обхід листків розпочато з того, для якого шлях від кореня був весь час до лівого нащадка, поки не потрапили у листок.

Подану далі програму можна удосконалити на випадок використання функцій багатьох змінних після визначення послідовностей символів, що позначатимуть функції 1, 2, 3 і т. і. змінних. Якщо буде достатньо лише символів, то істотних змін структура програми щодо обчислення виразу не зазнає. Інакше доведеться або використовувати умовні оператори, або вкладати оператори case один в інший. При цьому тип поля `a` типу запису `b` потрібно буде змінити, наприклад, на `string`. Змін має зазнати:

- опис типу `b` щодо вказівників: або масив, або список. Вже не буде лише лівих чи правих нащадків, хоча порядок серед них запровадити потрібно;
- алгоритм обходу листків дерева.

```
{Створення бінарного дерева для виконання арифметичних дій
```

```
                    згідно зі зворотним польським записом}

type p=^b;  b=record       {вершина дерева)
              n: real;     {число}
              a: char;     {символ арифметичної дії (' ' для
числа)}
              l,r,         {вказівники на операнди  (NIL для
числа)}
              prev: p end; {вказівник на попередній запис}
var cp,                    {попередня вершина}
    cl,                    {її лівий  нащадок}
    cr: p;                 {її правий нащадок}
    i,                     {вхідний файл, що містить рядки-ЗПЗ}
    o: text;               {вихідний файл, що міститиме рядки-
результати}
    num: boolean;          {чи останнє дане є числом?}
     st: string;           {послідовність символів числа}
      s: char;             {останній зчитаний символ}
      r: real;             {останнє число}
   code: integer;          {код перетворення рядка}
BEGIN
assign(o,'TREE.SOL');  rewrite(o);
assign(i,'TREE.DAT');  reset(i);              while not eof (i)
do begin
new(cp);  cp^.prev:=nil;  cp^.a:=' ';
                                 num:=true;    while not eoln(i)
do begin
read(i,s);  while s=' ' do read(i,s);                      case
s of
'0','1','2','3','4','5','6','7','8','9','.':
begin
{Цифри числа}            st:='';  while (s<>' ') and not eoln(i)
do begin
                                                st:=st+s;
read(i,s) end;
{Знаходження числа}               val(st,r,code);  if code<>0
then begin
            writeln(o,'Є некоректний запис числа.');  close(o);
halt end;
{Продовжуємо заповнювати числами}                        if num
then begin
new(cl);  cl^.a:=' ';  cl^.l:=nil;  cl^.r:=nil;  cl^.prev:=cp;
cp^.l:=cl;
new(cr);  cr^.a:=' ';  cr^.l:=nil;  cr^.r:=nil;  cr^.prev:=cp;
cp^.r:=cr;
          cl^.n:=r;    cp:=cr
end
{Вставка вершини між коренем і cp}           else if cp^.a=' '
then begin
new(cr);  cr^.a:=' ';  cr^.l:=cp^.r;  cr^.prev:=cp;
cp^.r^.prev:=cr;
                       cp^.r:=cr;     cp:=cr;
```

```
new(cr);  cr^.a:=' ';  cr^.l:=nil;  cr^.r:=nil;  cr^.prev:=cp;
cp^.r:=cr;
new(cl);  cl^.a:=' ';  cl^.l:=nil;  cl^.r:=nil;  cl^.prev:=cr;
cr^.l:=cl;
          cl^.n:=r;    cp:=cr;
new(cr);  cr^.a:=' ';  cr^.l:=nil;  cr^.r:=nil;  cr^.prev:=cp;
cp^.r:=cr;
                       cp:=cr
end

else begin
{Новий корінь дерева}                cl:=cp;
new(cp);  cp^.a:=' ';  cp^.l:=cl;  cl^.prev:=cp;  cp^.prev:=nil;
new(cr);  cr^.a:=' ';  cp^.r:=cr;  cr^.prev:=cp;  cp:=cr;
new(cl);  cl^.a:=' ';  cp^.l:=cl;  cl^.prev:=cp;  cl^.r:=nil;
cl^.l:=nil;
new(cr);  cr^.a:=' ';  cp^.r:=cr;  cr^.prev:=cp;  cr^.r:=nil;
cr^.l:=nil;
                                                    cl^.n:=r;
cp:=cr end;
                                          num:=true
end;
'+','-','*',':','/':
begin
                     if num then begin
cp:=cp^.prev^.prev;  cp^.a:=s;
cp^.r^.n:=cp^.r^.l^.n;  num:=false;
dispose(cp^.r^.l);  cp^.r^.l:=nil;
dispose(cp^.r^.r);  cp^.r^.r:=nil  end else cp^.a:=s;
if cp^.prev<>nil then cp:=cp^.prev
end end end;

           {Обчислення виразу за створеним бінарним деревом}
repeat  while cp^.l^.l<>nil do cp:=cp^.l;
      if (cp^.r^.l=nil) and (cp^.r^.r=nil) and
         (cp^.l^.l=nil) and (cp^.l^.r=nil) then        begin
                                                 case cp^.a of
        '+':                      cp^.n:=cp^.l^.n+cp^.r^.n;
        '-':                      cp^.n:=cp^.l^.n-cp^.r^.n;
        '*':                      cp^.n:=cp^.l^.n*cp^.r^.n;
    ':','/': if cp^.r^.n<>0 then cp^.n:=cp^.l^.n/cp^.r^.n
                            else                        begin
        writeln(o,'Вираз не існує, бо є ділення на 0.');
                                        close(o);  halt end;
                                                 else begin
        writeln(o,'Є символ, що не є цифрою або знаком ',
              'арифметичної дії.');  close(o);  halt end end;
        dispose(cp^.l);  cp^.l:=nil;
        dispose(cp^.r);  cp^.r:=nil;
        if cp^.prev<>nil then cp:=cp^.prev               end
   else cp:=cp^.r;
until (cp^.r=nil) and (cp^.l=nil) and (cp^.prev=nil);
                                              {Запис відповіді}
```

```
writeln(o,cp^.n:20:4);  readln(i);  dispose(cp)          end;
                                     close(i);  close(o) END.
```