Станка Матковић и Мијодраг Ђуришић

ПРОГРАМИ ЗА ИЗРАЧУНАВАЊЕ ВРЕДНОСТИ ЦЕЛОБРОЈНИХ АРИТМЕТИЧКИХ ИЗРАЗА

Аритметички изрази су саставни део сваког програмског језика. Као и остале конструкције програмских језика они се рачунару задају у облику текста, тако да се поставља питање на који начин се могу израчунати њихове вредности. У овом тексту описујемо израчунавање „уобичајених“, целобројних аритметичких израза, односно израза који садрже четири основне аритметичке операције:

$*$ (множење), : (целобројно дељење),

$+$ (сабирање) и $-$ (одузимање) (где су $*$ и : већег приоритета од $+$ и $-$)

целе бројеве и заграде „(“ и „)“.

У програмима који следе приказано је израчунавање вредности синтаксно исправних аритметичких израза, јер је проблем синтаксне исправности израза посебан проблем.

Дефинишимо типове података које користимо у програмима:

```
type niz_brojeva = array[1..100] of integer;
    niz_znakova = array[1..100] of char;
```

Израз чију вредност рачунамо налази се у низу типа `niz_znakova` iz дужине n. Променљиве iz и n третирамо као глобалне у свим програмима који следе.

1. Израчунавање вредности аритметичког израза свођењем на израз без заграда

Знајући да заграде мењају приоритет операција, проблем сводимо на једноставнији проблем израчунавања вредности аритметичког израза без заграда на следећи начин.

Уколико постоје заграде у изразу, налазимо позицију прве затворене заграде k и њој одговарајуће отворене заграде p (последње отворене заграде која претходи затвореној загради на позицији k). Израз између позиција $p+1$ и $k-1$ не садржи заграде.

Израчунамо вредност израза без заграда између позиција $p+1$ и $k-1$.

Упишемо израчунату вредност уместо дела израза чију смо вредност израчунали, почев од позиције p, уз одговарајуће сажимање низа iz.

Описани поступак се понавља све док у низу iz постоје заграде, а кад заграде не постоје, онда се израчуна вредност израза без заграда.

Израчунавање вредности аритметичког израза свођењем на израз без заграда реализујемо следећим сегментом програма.

```
repeat
    ima_zagrada:=nadji_zagrade(p,k);
    br:=izraz_bez_zagrada(p,k);
    if ima_zagrada then upisi_broj_u_niz_znakova(br,p,k)
  until not ima_zagrada;
writeln(' Vrednost izraza je :  ',br);
```

Функција `nadji_zagrade(var p,k:integer)` враћа логичку вредност `true` уколико заграде постоје, `false` иначе. Вредности параметара k и p представљају редом, позиције прве затворене и одговарајуће отворене заграде у изразу. Ако заграда нема вредност променљиве, p је 0, а k је $n+1$.

```
function nadji_zagrade(var p,k:integer):boolean;
  begin
    k:=1;p:=0;
    while (k<=n) and (iz[k]<>')') do
      begin
        if iz[k]='(' then p:=k;
        k:=k+1
      end;
    nadji_zagrade:=k<=n
  end;
```

Функција `izraz_bez_zagrada(p,k:integer)` израчунава вредност израза без заграда који се налази у низу iz почев од позиције $p+1$ до позиције $k-1$.

Основна идеја је да се прво изврше операције множења и дељења, да се редослед извршавања операција сабирања и одузимања чува у низу операцијских знакова `oznak` и да се бројеви над којима треба применити те операције чувају у одговарајућем редоследу у низу бројева `faktor`. На основу низова `oznak` и `faktor` једноставно се израчунава вредност израза.

Бројеви израза су задати знаковима, па их морамо превести у нумерички облик. То реализујемо функцијом `broj`. Функција `broj` помера индекс t низа iz на последњу цифру броја.

```
function broj(var nz:niz_znakova; var t:integer):integer;
var promena,br:integer;
begin
  promena:=1;
  if nz[t]='-' then promena:=-1;
  if (nz[t]='+') or (nz[t]='-') then t:=t+1;
  br:=0;
  while (t<=n) and (nz[t]>='0') and (nz[t]<='9') do
    begin
      br:=10*br+ord(nz[t])-ord('0');
      t:=t+1;
    end;
  t:=t-1;
```

```
  broj:=promena*br
end;
```

Јасно је да израз без заграда почиње бројем, па на почетку функције први број садржан у изразу уписујемо у низ `faktor`. Затим, редом, анализирамо знакове низа iz до краја израза без заграда (односно до позиције k).

Уколико је текући знак ($iz[t]$) „+“ или „−“ (операција нижег приоритета) уписујемо га у низ операцијских знакова (`oznak`).

Ако је текући знак „∗“ или „:“, како је израз исправан, после операције у изразу без заграда следи број, који одређујемо функцијом `broj` и одмах примењујемо текућу операцију ($iz[t]$) редом, на број који се налази на крају низа `faktor` и новодобијени број. Резултат уписујемо у низ `faktor` уместо његовог последњег члана.

Ако је текући знак цифра, одређујемо нови број и додајемо га на крај низа `faktor`.

Када дођемо до краја израза без заграда, у низу `oznak` налазе се операцијски знаци „+“ и „−“ које редом примењујемо на чланове низа `faktor`.

```
function izraz_bez_zagrada(p,k:integer):integer;
var i,t,rez:integer;
    faktor:niz_brojeva;
    oznak:niz_znakova;
begin
  t:=p+1;i:=1;
  faktor[i]:=broj(iz,t);
    t:=t+1;
    while t<k do
    begin
      if(iz[t]='+') or (iz[t]='-')
        then oznak[i]:=iz[t]
        else if (iz[t]='*') or (iz[t]=':')
          then begin
            t:=t+1;
            if iz[t-1]='*'
              then faktor[i]:= faktor[i]*broj(t)
              else faktor[i]:= faktor[i] div broj(t);
          end
          else begin
            i:=i+1;
            faktor[i]:=broj(iz,t);
          end;
      t:=t+1;
    end;
    rez:=faktor[1];
    t:=1;
    while t<i do
```

```
    begin
      if oznak[t]='+' then rez:=rez+faktor[t+1]
        else rez:=rez-faktor[t+1];
      t:=t+1
    end;
  izraz_bez_zagrada:=rez
end;
```

Процедура `upisi_broj_u_niz_znakova(br,p,k:integer)` уписује цео број br у низ знакова iz уместо знакова који се налазе од позиције p до позиције k.

Уколико је $br < 0$, у зависности од вредности знака у низу iz пре позиције p ($iz[p-1]$), уписујемо одговарајући знак на позицију $p-1$ или p. Од позиције k ка позицији p уписујемо цифре броја br, од цифре најмање тежине ка цифри највеће тежине. Неискоришћени део низа iz између позиција p и k бришемо.

```
procedure upisi_broj_u_niz_znakova(br,p,k:integer);
var i:integer;
  begin
    if br=0 then iz[k]:='0'
      else begin
        if br<0
        then begin
          if p=1
            then begin iz[p]:='-'; p:=p+1; end
            else if iz[p-1]='-'
              then iz[p-1]:='+'
              else if iz[p-1]='+'
                then iz[p-1]:='-'
                else begin
                  iz[p]:='-';
                  p:=p+1;
                end;
          br:=-br;
        end;
        while br<>0 do
          begin
            iz[k]:=chr(br mod 10 +ord('0'));
            br:=br div 10;
            if br<>0 then k:=k-1
          end
      end;
  n:=n-k+p;
  for i:=p to n do
    iz[i]:=iz[k+i-p]
end;
```

2. Израчунавање вредности израза коришћењем постфиксног записа

Постфиксни запис израза омогућава једноставно израчунавање вредности израза, јер не садржи заграде и у њему су све операције једнаког приоритета па се извршавају редом (с лева у десно).

2.1. Израчунавање вредности постфиксног израза

Нека је постфиксни израз дат низом знакова pf дужине n, тако да су операнди и операцијски знаци, операнди, као и операцијски знаци међусобно, раздвојени бланко симболом, написати програм којим се израчунава његова вредност.

Пример: Инфиксном запису израза $((2+3)*5-7*3):(4-3*2)$ одговара постфиксни запис $2\ 3+5*7\ 3*-4\ 3\ 2*-:$, а вредност израза је -2.

Основна идеја је да се бројеви редом уписују у низ (a), до појаве операције. Затим се та операција примењује, редом на последња два члана низа a ($a[i-1]$ и $a[i]$), а резултат се уписује на место првог операнда ($a[i-1]$). Приметимо да се извршавањем сваке операције смањује број елемената низа a. Поступак се наставља до краја постфиксног израза (pf). Јасно је да ће вредност постфиксног израза бити једини члан низа a ($a[1]$).

```
function vrednost_postizraza(var pf:niz_znakova;n:integer):integer;
  var t,i:integer;
    a:niz_brojeva;
  begin
  t:=1;i:=0;
    while t<=n do
      begin
        if ((pf[t]>='0') and (pf[t]<='9')) or
          ((t<n) and (pf[t] in ['-','+']) and (pf[t+1]<>' '))
            then begin
              i:=i+1;
              a[i]:=broj(pf,t);
            end
          else if (pf[t]<>' ')
            then begin
              case pf[t] of
                '+':a[i-1]:=a[i-1]+a[i];
                '-':a[i-1]:=a[i-1]-a[i];
                '*':a[i-1]:=a[i-1]*a[i];
                ':':a[i-1]:=a[i-1] div a[i]
              end;
              i:=i-1;
            end;
          t:=t+1
        end;
      vrednost_postizraza:=a[1];
    end;
```

2.2. Превођење израза из инфиксног у постфиксни запис

Вредност инфиксног аритметичког израза можемо израчунати коришћењем претходног алгоритма, уколико израз из инфиксног записа (iz) преведемо у одговарајући постфиксни запис (pf), па на pf применимо претходни алгоритам.

У инфиксном запису операцијски знак се налази између првог и другог операнда тако да, када формирамо одговарајући постфиксни запис потребно је запамтити операцијски знак све док у постфиксни запис не упишемо одговарајући други операнд. Операцијске знаке памтимо у низу `oznak` дужине `br_op`.

Анализирамо елемент по елемент низа iz на следећи начин.

Ако је текући елемент прва цифра или знак неког броја, у низ pf уписујемо све знаке који чине тај број.

Ако је текући елемент операцијски знак, његов први операнд је већ уписан у низ pf. Да ли у постфиксни запис можемо уписати операцијски знак са краја низа `oznak` (`oznak[br_op]`) зависи од тога да ли смо до краја уписали његов други операнд. Наиме, у случају да је тај операцијски знак „+“ или „−“, а текући знак „∗“ или „:“ (операција већег приоритета), други операнд операције `oznak[br_op]` је сложен, тј. није још увек уписан у постфиксни запис, па операцијски знак `oznak[br_op]` не можемо уписати. У случају да је `oznak[br_op]` већег или једнаког приоритета од текуће операције, други операнд операције `oznak[br_op]` је уписан, па `oznak[br_op]` можемо уписати у постфиксни израз и избрисати га из низа `oznak` ($br\_op = br\_op - 1$). У сваком случају текући операцијски знак уписујемо у низ `oznak`, јер још увек није формиран његов други операнд.

Описани поступак се односи на израз без заграда. Како заграде мењају приоритет операција, када је отворена заграда текући елемент у низу iz, уписујемо је у низ `oznak`, да би, кад наиђемо на затворену заграду, знали које операције из низа `oznak` треба уписати у постфиксни запис. По упису одговарајућих операцијских знакова у постфиксни запис, заграду уклањамо из низа `oznak`. Значи, заграде посматрамо као операције одговарајућег приоритета, што регулишемо функцијом `prioritet`.

Кад дођемо до краја низа iz, све операције из низа `oznak`, редом, од последње ка првој, уписујемо на крај постфиксног записа.

```
function prioritet(op1,op2:char):boolean;
{funkcija vraca true ako je op1 veceg ili jednakog prioriteta od op2
inace vraca false}
begin
  if op1='('
    then prioritet:=false
    else if op2='(' then prioritet:=false
      else if op2=')'
        then prioritet:=true
        else prioritet:=(op1 in['*',':'])  or
          ((op1 in ['+','-'])and(op2 in ['+','-']));
end;
```

```
procedure ifpf(var pf:niz_znakova;var m:integer);
var oznak:niz_znakova;
  i,br_op:byte;
  prvi_broj:boolean;
begin
  m:=0; br_op:=0; prvi_broj:=true; i:=1;
  while i<=n do
    if ((iz[i]>='0')and(iz[i]<='9'))or
      (prvi_broj and (iz[i] in['+','-']))
      then begin
        m:=m+1;pf[m]:=iz[i];
        i:=i+1;
        while(i<=n)and(iz[i]>='0')and(iz[i]<='9') do
          begin
            m:=m+1;pf[m]:=iz[i];
            i:=i+1;
          end;
          m:=m+1;pf[m]:=' ';
          prvi_broj:=false;
      end
      else begin
        while (br_op>0) and prioritet(oznak[br_op],iz[i]) do
          begin
            m:=m+1;pf[m]:=oznak[br_op];m:=m+1;pf[m]:=' ';
            br_op:=br_op-1
          end;
        if iz[i]=')'
          then br_op:=br_op-1
          else begin
            br_op:=br_op+1; oznak[br_op]:=iz[i];
          end;
        prvi_broj:=iz[i]='(';
        i:=i+1;
      end;
  for i:=br_op downto 1 do
  begin m:=m+1;pf[m]:=oznak[i];m:=m+1;pf[m]:=' '; end;
end;
```

Идеју коју смо користили у овом решењу можемо применити и за директно рачунање вредности инфиксног израза, без превођења у постфиксни запис, уз мале модификације.

3. Израчунавање вредности инфиксног израза коришћењем граматике за опис израза

Инфиксни аритметички израз може се формално дефинисати следећом граматиком, коришћењем Бекусове нотације:

<израз>::=[<а_оп>] <терм> {<а_оп> <терм>}
<терм>::= <фактор> {<м_оп> <фактор>}
<фактор>::= <број> |'(' <израз> ')'
<број>::= <цифра> {<цифра>}
<а_оп>::='+' | '−'
<м_оп>::='∗' | ':'
<цифра>::='0' | '1' | '2' | . . . | '9'

На основу наведене граматике израчунавање вредности аритметичког израза можемо решити и техником рекурзивног спуста, која се иначе користи за проверу синтаксне исправности конструкција одређених формалних језика.

За сваки од сложених појмова који се користе у дефиницији израза направљена је одговарајућа функција. Израз чију вредност рачунамо записан је у низу знакова iz дужине n, а променљива i садржи текућу вредност индекса низа iz (на почетку $i = 1$). Израчунавање вредности израза почиње позивом функције `izraz`.

```
var iz:niz_znakova;
  i,n:integer;
function term:integer; forward;
function faktor:integer; forward;

function izraz:integer;
var a:integer;
  op:char;
begin
  op:='+';
  if iz[i] in ['+','-']
    then begin op:=iz[i]; i:=i+1; end;
  a:=term;
  if op='-' then a:=-a;
  while (i<n) and ((iz[i]='+')or (iz[i]='-')) do
  begin
    op:=iz[i];
    i:=i+1;
    if op='+' then a:=a+term
      else a:=a-term;
  end;
  izraz:=a;
end;
```

```
function term:integer;
var a:integer;
  op:char;
begin
  a:=faktor;
  while(i<n)and((iz[i]='*')or (iz[i]=':'))  do
  begin
    op:=iz[i];
    i:=i+1;
    if op='*' then a:=a*term
      else a:=a div term;
  end;
  term:=a;
end;

function broj:integer;
var a:integer;
begin
  a:=0;
  while (i<=n) and (iz[i]>='0')and(iz[i]<='9')do
  begin
    a:=a*10+ord(iz[i])-ord('0');
    i:=i+1;
  end;
  broj:=a;
end;

function faktor:integer;
begin
  if iz[i]='('
    then begin
      i:=i+1;
      faktor:=izraz;
      i:=i+1;
    end
  else faktor:=broj;
end;
```

У овом тексту су дате само неке од идеја за израчунавање вредности инфиксног аритметичког израза. Због једноставности кода радили смо искључиво са целобројним изразима али се програми лако могу проширити за рад са реалним бројевима. Такође, у израз се могу уврстити и друге, прецизно дефинисане (приоритетом) операције. У програмима није анализиран случај дељења нулом.