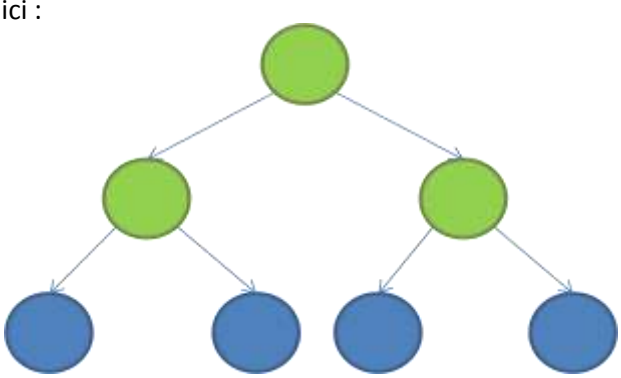
UVOD

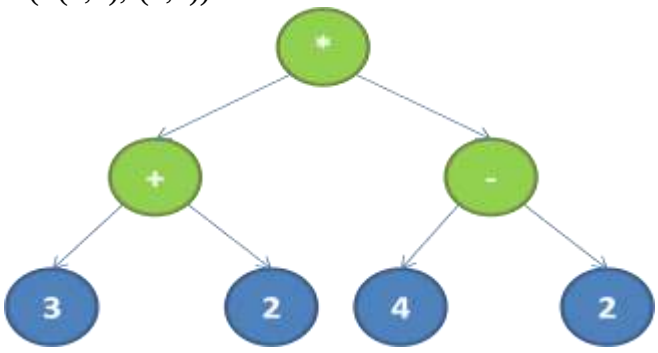
- Oblik notacije koji se koristi u logici, aritmetici i algebri. Njena je odlika da se operatori pisu levo od svojih operanada. Ako je arnost operatora fiksirana , sintaksa koja se dobija je nedvosmislena i bez zagrada. Poljski logičar Jan Lukašijevič je izumeo ovu notaciju oko 1920, pa se ponekad ova notacija zove “Poljska notacija”.
- Više se ne koristi puno u logici, ali je našla primenu u računarstvu.

ALGORITAM

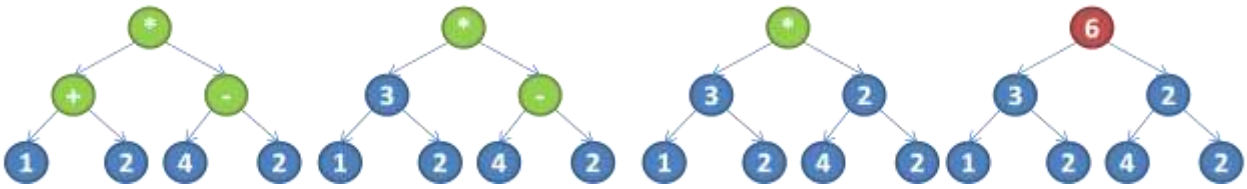
Ako zelenim krugovima obeležimo znakove (+, -, *, /), a plavim brojeve, raspored elemenata u stablu bi izgledao kao na sledećoj slici :



Kao što se vidi na prethodnoj slici, u korenu i unutrašnjim čvorovima se nalaze znakovi, dok se u listovima nalaze brojevi. Ovakvo stablo je poznato pod nazivom drvo izraza (eng. expression tree).
Tako bi na primer izraz *(+(3,2),-(4,2)) smestili na drvo izraza na sledeći način :



Računanje ovog izraza bi uradili tako što bi došli do listova i onda izračunali vrednost tog jednog izraza , a rezultat smestili u čvor čija su deca ti listovi. To primenimo na svim čvorovima dok u korenu drveta ne dobijemo broj.
Na sledećoj slici možemo videti kako bi ovaj algoritam radio na primeru : *(+(1,2),-(4,2)) .



PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define LEN 15 Maksimalna dužina broja
#define MAX 100 Maksimalna dužina izraza
#define MAXDEPTH 30 Najveća dubina stabla ( Najdublji nivo stabla )

typedef struct node
 {
  char *data; Podatak
  struct node *left; Levo dete
  struct node *right; Desno dete
 }NODE;

int GetType( char c )
 {
  if( ( c >= '0' && c <= '9' ) || c == '.')
    return 1; Ako je broj vrati 1
  else if( c == '+' || c == '-' || c == '*' || c == '/')
    return 2; Ako je znak vrati 2
  else if( c == '\n' )
    return -1; Ako je znak za novi red vrati -1
  return 0;  U ostalim slucajevima vrati 0
 }

char *GetMeData(char line[],int *index,int *TYPE) Funkcija za dobijanje stringa istog tipa podataka
 {
  int curr,type,i=0,j;

  char data[LEN],*ret;

  curr = *index; Postavljamo prvi indeks
  type = GetType(line[curr]); Uzimamo tip prvog karaktera
  *TYPE = type; Postavljamo tip koji vracamo u program na taj trenutni
  data[ i++ ] = line[ curr ]; Prvi karakter stringa koji se vraca postaje trenutni karakter stringa koji smo
poslali
  while( GetType( line[ ++curr ] ) == type) Dok je tip isti
   data[ i++ ] = line[ curr ]; Upisuj karaktere iz poslatog stringa u onaj koji se vraca
  data[ i ] = '\0'; Završi povratni string
  *index = curr;  Ažuriraj index
  ret = (char*) malloc( i * sizeof(char)); Alociraj memoriju za povratni pokazivač
  strcpy(ret, data);  Kopiraj podatke
  return ret;  Vrati string
 }
```

```
NODE* MakeNew( char line[],int *index,int *type) Funkcija pravi novi čvor
 {
  NODE *ret;
  ret = ( NODE* ) malloc( sizeof( NODE ) ); Alociranje memorije
  ret->data = GetMeData( line,index,type ); Dobijanje podatka
  ret->left = ret->right = NULL; Ažuriranje pokazivača
  return ret; Vraćanje vrednosti
 }

NODE *Populate( char line[] ) Funkcija pravi izrazno stablo
  {
    NODE *L[MAXDEPTH];
    int LB=0;
    int i=0,tip,t;
    NODE *ROOT = NULL;
    do{
        ROOT = MakeNew( line, &i, &tip );      Ucitavanje ROOT-a
        if(!GetType(ROOT->data[0])) Ako nije ni broj ni znak ni novi red
         {
          ROOT = NULL; ROOT se postavlja na NULL
          if( GetType( line[ i ]  ) < 0 ) Ako smo naisli na znak "novi red"
            return NULL; Vratiti NULL
         }
     }while(!ROOT); Prethnodni kod ponavljati dok se ne dobije ROOT
    L[LB]=ROOT;  Na nultom levelu postavljamo ROOT

    while( GetType( line[ i ] ) >= 0 )
     {
      while( t = GetType( line[ i ] ) == 0 )  Sve dok se ne dobije znak ili broj
       if( t == -1)  Ukoliko je znak za novi red
         break;  izlazimo  iz petlje
       else
         i++; Ako nije znak za novi red idemo na sledeći karakter
      if( t == -1 ) Ukoliko je znak za novi red
        break;  izlazimo  iz petlje
      else  Ako je znak ili broj
       {
         if(!ROOT->left) Ako ne postoji levo dete
          {
            ROOT->left = MakeNew( line, &i , &tip ); "Napravimo" levo dete
            if( tip == 2 ) Ako je znak
             {
               ROOT = ROOT->left; Udji u levo dete
               L[ ++LB ] = ROOT; Novi koren je na LB nivou
             }
            continue; Preskoči sve ispod
          }
```

```
        else if(!ROOT->right) Ako postoji levo, ali ne postoji desno dete
         {
          ROOT->right = MakeNew( line, &i , &tip ); "Napravimo" desno dete
          if( tip == 2 ) Ako je znak
           {
            ROOT = ROOT->right; Uđi u desno dete
            L[ ++LB ] = ROOT; Novi koren je na LB nivou
           }
          continue; Preskoči sve ispod
         }
        else
         {
          while( ROOT->left && ROOT->right ) Dok postoji i levo i desno dete
           {
            if( ! LB ) Ako smo na nultom nivou
              break; Izlazimo iz petlje
            ROOT = L[--LB];  smanjujemo nivo
           }
         }
      }
    }
   ROOT = L[0]; Pravi koren se nalazi na nultom nivou
   return ROOT;  Vratimo koren
 }

void LKD(NODE *ROOT) Ispis u Levi-Koren-Desni redosledu
 {
  if(ROOT) Ako postoji koren
   {
    if(ROOT->left != NULL ) Ako postoji levo dete
     {
      putchar('('); Otvori zagradu
      LKD(ROOT->left); Ispisi levo dete
     }
    else if (ROOT->left == NULL && GetType(ROOT->data[0]) == 2 )
     { Ako ne postoji levo dete, a u korenu je znak
      printf("Pogresan unos!\n"); Prijavi korisniku pogrešan unos

      exit(1);   Izađi
     }
    printf("%s",ROOT->data); Ispiši podatak u korenu
    if(ROOT->right  != NULL ) Ako postoji desno dete
     {
      LKD(ROOT->right); Ispisi desno dete
      putchar(')'); Zatvori zagradu
     }
    else if (ROOT->right == NULL && GetType(ROOT->data[0]) == 2 )
     { Ako ne postoji desno dete, a u korenu je znak
```

```
      printf("Pogresan unos!\n"); Prijavi korisniku pogrešan unos

      exit(1);   Izađi
     }
   }
 }
void LDK(NODE *ROOT) Ispis u Levi- Desni-Koren redosledu
 {
  if(ROOT)
   {
    if(ROOT->left != NULL ) Ako postoji levo dete
     {
      putchar('('); Otvori zagradu
      LDK(ROOT->left); Ispisi levo dete
     }
    else if (ROOT->left == NULL && GetType(ROOT->data[0]) == 2 )
     { Ako ne postoji levo dete, a u korenu je znak
      printf("Pogresan unos!\n"); Prijavi korisniku pogrešan unos

      exit(1);   Izađi
     }
    if(ROOT->right  != NULL ) Ako postoji desno dete
     {
      LDK(ROOT->right); Ispisi desno dete
      putchar(')'); Zatvori zagradu
     }
    else if (ROOT->right == NULL && GetType(ROOT->data[0]) == 2 )
     { Ako ne postoji desno dete, a u korenu je znak
      printf("Pogresan unos!\n"); Prijavi korisniku pogrešan unos

      exit(1);   Izađi
     }
   }
    printf("%s",ROOT->data); Ispiši podatak u korenu
   }
 }

void KLD(NODE *ROOT) Ispis u Koren-Levi-Desni redosledu
 {
  if(ROOT)
   {
    printf("%s",ROOT->data); Ispiši podatak u korenu
    if(ROOT->left != NULL ) Ako postoji levo dete
     {
      putchar('('); Otvori zagradu
      KLD(ROOT->left); Ispisi levo dete
     }
    else if (ROOT->left == NULL && GetType(ROOT->data[0]) == 2 )
```

```
    { Ako ne postoji levo dete, a u korenu je znak
      printf("Pogresan unos!\n"); Prijavi korisniku pogrešan unos

      exit(1);   Izađi
    }
  if(ROOT->right  != NULL ) Ako postoji desno dete
    {
      LDK(ROOT->right); Ispisi desno dete
      putchar(')'); Zatvori zagradu
    }
  else if (ROOT->right == NULL && GetType(ROOT->data[0]) == 2 )
    { Ako ne postoji desno dete, a u korenu je znak
      printf("Pogresan unos!\n"); Prijavi korisniku pogrešan unos

      exit(1);   Izađi
    }
  }
}

char* sracunaj(char operacija, char *operand1, char *operand2)
 { Funkcija koja računa vrednost izraza zadatog preko znaka i dva stringa, vraća string
  float a,b,rez;
  char *ret;
  a = atof( operand1 ); Konvertujemo prvi operand (string) u broj (float)
  b = atof( operand2 ); Konvertujemo drugi operand (string) u broj (float)
  switch(operacija)
   {
    case '+' : rez = a + b; break; Ako je operacija + saberi brojeve a i b
    case '-' : rez = a - b; break; Ako je operacija - odutzmi brojeve a i b
    case '*' : rez = a * b; break; Ako je operacija * pomnoži brojeve a i b
    case '/' :  Ako je operacija /
        {
          if(b) Ako je b različito od 0
            rez = a / b; Podeli a sa b
          else Inače
            {
              printf("Rezultat : Deljenje sa nulom nije dozvoljeno!!\n"); Prijavi grešku

              exit(1); Izađi
            }
        } break;
    default : printf("NEPOZNATO!"); Ako dođe do greske u programu
   }
  ret = (char*) malloc(LEN * sizeof(char)); Alociranje memorije
  if( !( ret ) ) Ako dođe do greške pri alokaciji
   {
    printf("Greska pri alociranju memorije!\n"); Prijavi grešku
    exit(1);    Izađi
```

```
  }
 sprintf(ret, "%.3f",rez );  Prabacujemo broj u string
 return ret;  Vraćamo rezultat
}

char *Izracunaj(NODE *ROOT)
{
 char *ret;
 if( ( ROOT->left ) && ( ROOT->right ) ) Ako postoje levo i desno dete
   ret = sracunaj( ROOT->data[0],Izracunaj( ROOT->left ),Izracunaj( ROOT->right ));
  Rezultat je računanje izraza sa operandima u levom tj. desnom detetu , a operacijom u korenu
 else Ako ne postoje deca
   ret = ROOT->data; rezultat je koren
 return ret;  Vrati rezultat
}

int main()
{
 char line[MAX];
 NODE *ROOT = NULL;
 printf("Insert expression in prefix notation : \n");
 gets(line); Učitavanje stringa
 line[strlen(line)]='\n';
 ROOT = Populate( line ); Punjenje stabla
 if(ROOT) Ako postoji koren
  {
    printf("Infix : ");
    LKD(ROOT); Odštampaj u Levi-Koren-Desni redosledu
    putchar('\n');

    printf("Postfix : ");
    LDK(ROOT); Odštampaj u Levi-Desni-Koren redosledu
   putchar('\n');

    printf("Prefix : ");
    KLD(ROOT); Odštampaj u Koren-Levi-Desni redosledu
    putchar('\n');

    printf("Rezultat : ");
    KLD(ROOT);
    printf(" = %s\n",Izracunaj(ROOT)); Računanje izraza
  }
 else
  {
   printf("Pogresan unos!\n");
  }
 return 0;
}
```

DODATAK

1. Program bez komentara

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define LEN 15
#define MAX 100
#define MAXDEPTH 30

typedef struct node
 {
  char *data;
  struct node *left;
  struct node *right;
 }NODE;

int GetType( char c )
 {
  if( ( c >= '0' && c <= '9' ) || c == '.')
    return 1;
  else if( c == '+' || c == '-' || c == '*' || c == '/')
    return 2;
  else if( c == '\n' )
    return -1;
  return 0;
 }

char *GetMeData(char line[],int *index,int *TYPE)
 {
  int curr,type,i=0,j;

  char data[LEN],*ret;

  curr = *index;
  type = GetType(line[curr]);
  *TYPE = type;
  data[ i++ ] = line[ curr ];
  while( GetType( line[ ++curr ] ) == type)
   data[ i++ ] = line[ curr ];
  data[ i ] = '\0';
  *index = curr;
  ret = (char*) malloc( i * sizeof(char));
  strcpy(ret, data);
  return ret;
 }
```

```
NODE* MakeNew( char line[],int *index,int *type)
 {
  NODE *ret;
  ret = ( NODE* ) malloc( sizeof( NODE ) );
  ret->data = GetMeData( line,index,type );
  ret->left = ret->right = NULL;
  return ret;
 }
NODE *Populate( char line[] )
  {
    NODE *L[MAXDEPTH];
    int LB=0;
    int i=0,tip,t;
    NODE *ROOT = NULL;

    do{
        ROOT = MakeNew( line, &i, &tip );
        if(!GetType(ROOT->data[0]))
          {
           ROOT = NULL;
           if( GetType( line[ i ] ) < 0 )
             return NULL;
          }
      }while(!ROOT);
    L[LB]=ROOT;

    while( GetType( line[ i ] ) >= 0 )
     {
      while( t = GetType( line[ i ] ) == 0 )
       if( t == -1)
         break;
       else
         i++;
      if( t == -1 )
        break;
      else
        {
          if(!ROOT->left)
           {
            ROOT->left = MakeNew( line, &i , &tip );
            if( tip == 2 )
             {
               ROOT = ROOT->left;
               L[ ++LB ] = ROOT;
             }
            continue;
```

```
          }
         else if(!ROOT->right)
          {
            ROOT->right = MakeNew( line, &i , &tip );
            if( tip == 2 )
             {
               ROOT = ROOT->right;
               L[ ++LB ] = ROOT;
             }
            continue;
          }
         else
          {
            while( ROOT->left && ROOT->right )
             {
              if(!LB)
                break;
              ROOT = L[--LB];
             }
          }
       }
    }
   ROOT = L[0];
   return ROOT;
  }
void LKD(NODE *ROOT)
 {
  if(ROOT)
   {
    if(ROOT->left != NULL )
      {
       putchar('(');
       LKD(ROOT->left);
      }
    else if (ROOT->left == NULL && GetType(ROOT->data[0]) == 2 )
      {
       printf("Pogresan unos!\n");
       system("pause");
       exit(1);
      }
    printf("%s",ROOT->data);
    if(ROOT->right  != NULL )
      {
       LKD(ROOT->right);
       putchar(')');
      }
```

```
    else if (ROOT->right == NULL && GetType(ROOT->data[0]) == 2 )
      {
       printf("Pogresan unos!\n");
       system("pause");
       exit(1);
      }
   }
 }
void LDK(NODE *ROOT)
 {
  if(ROOT)
   {
    if(ROOT->left != NULL )
      {
       putchar('(');
       LDK(ROOT->left);
       putchar(',');
      }
    else if (ROOT->left == NULL && GetType(ROOT->data[0]) == 2 )
      {
       printf("Pogresan unos!\n");
       system("pause");
       exit(1);
      }

    if(ROOT->right  != NULL )
      {
       LDK(ROOT->right);
       putchar(')');
      }
    else if (ROOT->right == NULL && GetType(ROOT->data[0]) == 2 )
      {
       printf("Pogresan unos!\n");
       system("pause");
       exit(1);
      }
    printf("%s",ROOT->data);
   }
 }

void KLD(NODE *ROOT)
 {
  if(ROOT)
   {
    printf("%s",ROOT->data);
    if(ROOT->left != NULL )
```

```
    {
      putchar('(');
      KLD(ROOT->left);
      putchar(',');
    }
  else if (ROOT->left == NULL && GetType(ROOT->data[0]) == 2 )
    {
      printf("Pogresan unos!\n");
      system("pause");
      exit(1);
    }
  if(ROOT->right  != NULL )
    {
      KLD(ROOT->right);
      putchar(')');
    }
  else if (ROOT->right == NULL && GetType(ROOT->data[0]) == 2 )
    {
      printf("Pogresan unos!\n");
      system("pause");
      exit(1);
    }
 }
}

char* sracunaj(char operacija, char *operand1, char *operand2)
 {
  float a,b,rez;
  char *ret;
  a = atof( operand1 );
  b = atof( operand2 );
  switch(operacija)
   {
    case '+' : rez = a + b; break;
    case '-' : rez = a - b; break;
    case '*' : rez = a * b; break;
    case '/' :
        {
          if(b)
            rez = a / b;
          else
            {
              printf("Deljenje sa nulom nije dozvoljeno!!");
              exit(1);
            }
        } break;
```

```
    default : printf("NEPOZNATO!");
   }
  ret = (char*) malloc(LEN * sizeof(char));
  sprintf(ret, "%.3f",rez );
  return ret;
 }

char *Izracunaj(NODE *ROOT)
 {
  char *ret;
  if( ( ROOT->left ) && ( ROOT->right ) )
    ret = sracunaj( ROOT->data[0],Izracunaj( ROOT->left ),Izracunaj( ROOT->right ));
  else
    ret = ROOT->data;
  return ret;
 }

int main()
 {

  char line[MAX];
  NODE *ROOT = NULL;
  char c[LEN];
  printf("Insert expression in prefix notation : \n");gets(line);
  line[strlen(line)]='\n';
  ROOT = Populate( line );
  if(ROOT)
    {
      printf("Infix : ");
      LKD(ROOT);
      putchar('\n');

      printf("Postfix : ");
      LDK(ROOT);putchar('\n');

      printf("Prefix : ");
      KLD(ROOT);putchar('\n');

      strcpy(c,Izracunaj(ROOT));
      printf("Rezultat : %s\n",c);
    }
  else
     printf("Pogresan unos!\n");
 return 0;
  }
```