TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

Algoritmikeelte analüüs

Ringi pindaladega arvutamise keel

Priit Rebina
107362IASMM

Tallinn 2011

Sisukord

Sissejuhatus

Tänapäeva maailmas on välja töödeldud väga palju erinevaid programmeerimise keeli, millest osad on spetsiaalkeeled, mis tegelevad konkreetsete ette määratud ülesannetega. Oma programmeerimiskeele loomiseks on olemas erinevaid tüüpe teeke, mis lihtsustavad mingisse vahekeelde kompileerimise protsessi.

Näiteks kasutades GNU kompilaatori parserit Bison on võimalik genereeritud lähtekoodi kompileerida erinevate operatsioonisüsteemide all, mis toetavad C või C++. Bisoni sisendiks on fail, mis sisaldab loodava keele leksika ja süntaksi kirjeldust, samuti kasutatavaid funktsioone ja struktuure. Kasutades sellist lähenemist on nii mõnelgi juhul võimalik kiirendada tarkvara arenduse protsessi, kuna põhiline vaev kulub süntaksi ja leksika kirjelduseks, mitte kogu parseri loomiseks. Samuti teatud protsesside testimiseks vajalike sisendandmete loomiseks on võimalik sellist lähenemist kasutades imiteerida soovitavat tulemust.

Ringi pindaladega arvutamine

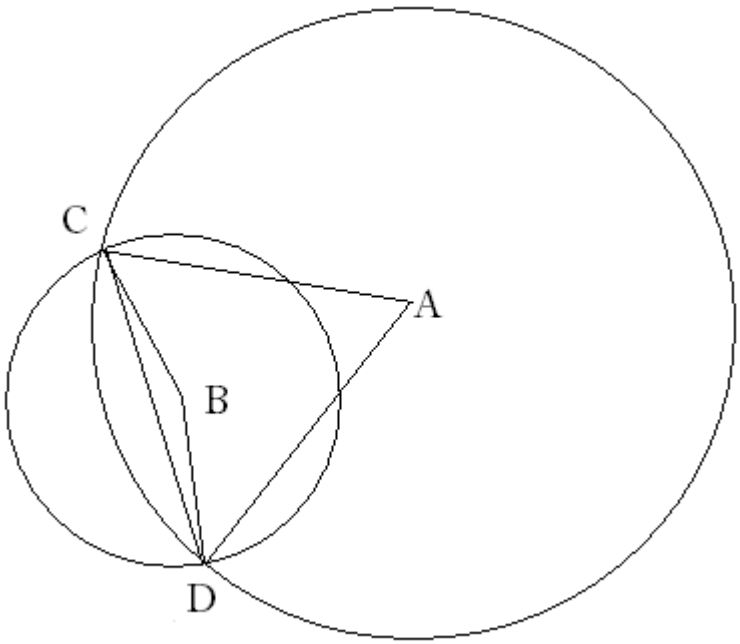
Ringi pindaladega arvutuste tegemiseks on vajalik teada ringist mõningaid andmeid. Arvustuste lihtsustamiseks tuleks teada ringi keskkoha koordinaate ning raadiust. Raadius on võimalik leida ka keskpunkti koordinaadi, ning ringjoonel paikneva punkti kauguse arvutuse teel. Selleks $r = \sqrt{((x1 - x0)^2 + (y1 - y0)^2}$, kus ringi keskpunkt on A(x0,y0) ja raadiuse koordinaat R(x1,y1). Ringide ühisosa leidmiseks tuleb leida mõlema ringi sektori pindalad, ning neid sektoreid ühendavate kolmnurkade pindalad. Üldine valem oleks kujul: väiksema ringi pindala – väiksema ringi sektor(CBD) + väiksema ringi kolmnurk(BCDB) + suurema ringi sektor(CAD) – suurema ringi kolmnurk(ACDA) (vt. Joonis 1.). Asendades vastavad valemis on võimalik saada lihtsustatud valem:

$$S = \pi r^2 - r^2 \arccos\left(\frac{R^2 - r^2 - d^2}{2dr}\right) + R^2 \arccos\left(\frac{R^2 + d^2 - r^2}{2dR}\right) - \frac{rR}{2}\sqrt{4 - \frac{(d^2 - R^2 - r^2)^2}{r^2 R^2}}$$

r = väiksema ringi raadius

R = suurema ringi raadius

d = kahe ringi keskpunktide vaheline kaugus



Joonis 1.

Keele kirjeldus

Ringi pindaladega arvutuste tegemiseks panin paika teatud üldised reeglid. Esiteks ei ole võimalik teha arvutusi üle kahe ringi pindaladega, kuna kasutusel on koordinaatteljestikul paiknemise süsteem ja üle selle oleks läinud arvutuste reeglid tunduvalt mahukamaks. Teiseks on ära määratud kindel andmete sisestuse viis, mis eemaldab võimalikud sisestusvead.

Programmeerimiskeele kirjeldamiseks on vajalik välja tuua kasutatava leksika. Ringi pindaladega arvutamise keeles on lubatud kasutada järgnevaid tähemärke:

1) Inglise keele tähestik a…z ja A…Z
2) Täisarvulised numbrid 0…9
3) Keelele omased märgid + - / * ; ( )

Leksikat kasutades on lubatud kasutada vastava süntaksiga lauseid:

1) exp '+' exp
2) exp '-' exp
3) exp '/' exp
4) exp '*' exp
5) exp

Kus exp võib olla

1) VAR
2) VAR '(' signedValue ';' signedValue ')' '(' signedValue ';' signedValue ')'

Kirjelduses kasutatav muutuja VAR on ringi nimetus, *signedValue* on ringi üks koordinaat punkt, mis võib omada ka negatiivset väärtust. NUM muutuja tüübiks on täisarv (*integer*), komaga arvu sisestamisel väljastab kompilaator hoiatuse, ning teostab arvutuse täisarvuks ümardatult.

signedValue:

1) NUM
2) '-' NUM

Süntaksist lähtuvalt on sisestatavate muutujate tüübid ringi pindala arvutamiseks vajalikud koordinaadid kujul: *„ringi nimi“* **(** *„keskpunkti x-telje koordinaat“* **;** *„keskpunkti y-telje*

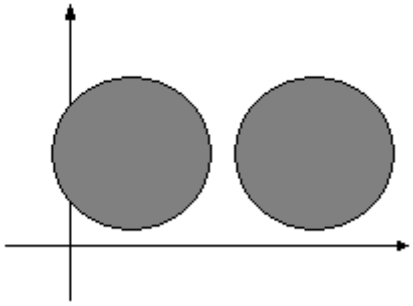
koordinaat“ **)(** *„ringjoonel asuv suvaline täisarvuline x-telje koordinaat“* **;** *„ringjoonel asuv suvaline täisarvuline y-telje koordinaat“* **)**

Sisestades ainult ringi nime VAR, lisatakse vastav muutuja koos ringi koordinaat punktide algkoordinaatidega mällu ja kuvatakse kasutajale väärtused (0;0)(0;0). Sisestades täielikult mingi ringi koos kõigi punktidega, näiteks: p(0;1)(3;3), kuvatakse kasutajale teade salvestamise tulemusest. Nagu süntaksi kirjeldusest näha on võimalik teostada tehteid:

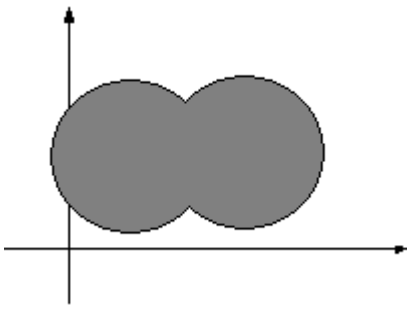
1) eelsalvestatud andmete ja uue deklaratsiooni käigus (s + p(1;2)(4;3)).
2) Ainult eelsalvestatud andmete vahel (s + p).
3) Ainult uute ringi andmete sisestamise käigus (s(3;3)(1;2) – k(8;2)(4;2)).

Arvutustehetest on lubatud kasutada + - / * ning tegevused peavad toimuma kahe ringi koordinaatide vahel.

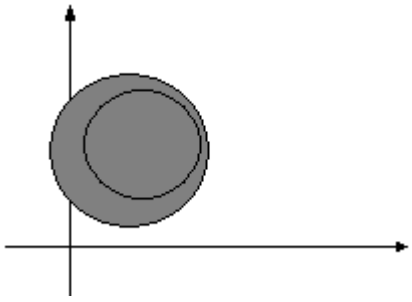
„ + „ märk teostab kahe ringi pindaladega kaetud ala pindala. Ringide mitte kattuvuse korral koordinaat teljestikul arvutatakse kahe ringi pindala summa (vt. joonis 2.). Ringide osalise kattuvuse kogu hõivatava pinna ala (vt. joonis 3.). Ühe ringi paiknemisel teise ringi sees tagastatakse suurema pindala (vt. Joonis 4.).



Joonis 2.

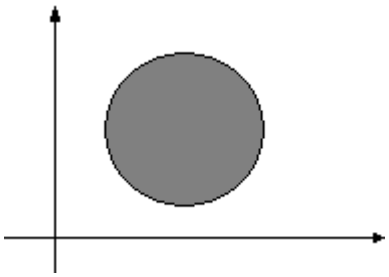


Joonis 3.

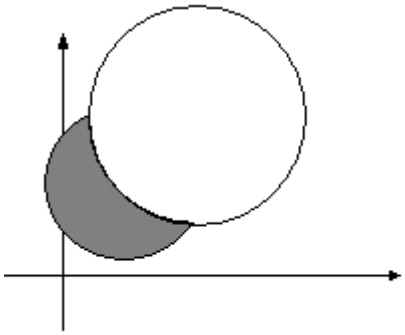


Joonis 4.

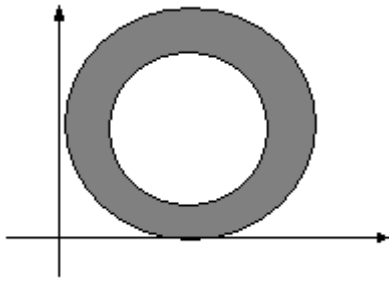
„ − „ märk teostab pindalade lahutus tehte. Esimese ringi pindala ilma teise ringi kattuva osata. Ringide mitte kattuvuse korral tagastatakse ainult esimese ringi pindala (vt. Joonis 5.). Ringide kattuvuse korral tagastatakse esimese ringi pindala ilma teise ringi osata (vt. Joonis 6.) Ühe ringi paiknemisel teise sees tagastatakse esimese ringi pindala lahutatult teise ringi pindalast, kui esimene on suurem (vt. Joonis 7.) vastasel juhul tagastatakse 0, kuna esimene ring ei paista välja (vt. Joonis 8.)



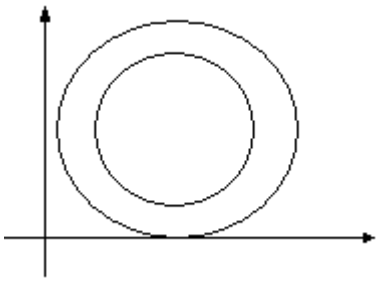
Joonis 5.



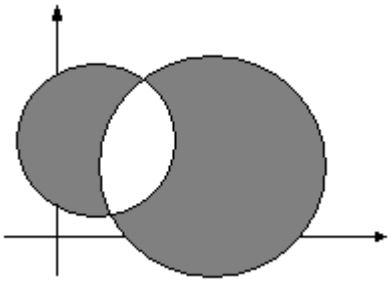
Joonis 6.



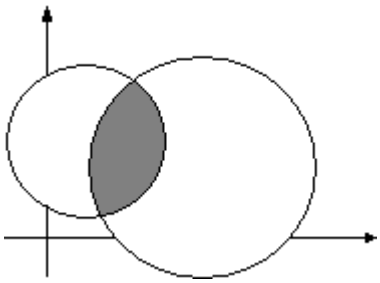
Joonis 7.



Joonis 8.



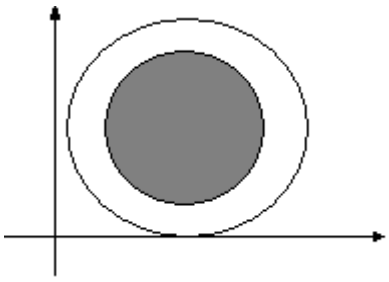
Joonis 9.



Joonis 10.

„ / „ märk teostab pindade lahutust ilma ühise osata. Ringide mitte kattuvuse korral tagastatakse mõlema pindala summa, kuna ühisosa puudub (vt. Joonis 2.). Ringide kattuvuse korral teostatakse liitmine ilma ühisosata (vt. Joonis 9.). Ringide teineteise sees asetuse korral tagastatakse sõltumata järjekorrast ja asetusest suurema pindala ilma ühisosata ehk väiksemata (vt. Joonis 7.). Ühe suuruste ringide korral on tulemuseks 0.

„ * „ märk teostab ringi pindalade ühisosade pindalade leidmist. Ringide mitte kattuvuse korral on tulemuseks 0. Ringide kattuvuse korral tagastatakse ainult ühisosa pindala (vt. Joonis 10.). Ringide teineteise sees paiknemisel arvutatakse väiksema pindala, kuna see ongi ühisosa (vt. Joonis 11.).



Joonis 11.

Näited keele kasutamisest

s(1;2)(4;2) s inserted m(5;2)(2;3) m inserted m+s m(5;2)(2;3) s(1;2)(4;2) 52.64	Sisend andmed Arvutuse teostamine. Kasutajale muutujate kuvamine. Tulemus.
m-k(4;4)(2;2) m(5;2)(2;3) k inserted 16.38	Erinevat tüüpi andmete sisestamine.
h(4;7)(3.2;4) warning: Value not integer h inserted h/s h(4;7)(3;4) s(1;2)(4;2) 58.80	Hoiatuse kuvamine koma arvu korral. Hoiatuse tagajärjel ümardatud väärtus.
n(4;a)(4;9) syntax error	Vea tekitamine. Programmi töö lõpetamine vea ilmnemise korral.

Lähtekood

```
%{
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>

struct symrec
{
  char *cName;
  int x1;
  int x2;
  int y1;
  int y2;
  struct symrec *pNext;
};

struct symrec *getsym(char const *);
struct symrec *putsym(char const *);
double SumOfArea(char const *, char const *);
double FirstWithoutTheSecond(char const *, char const *);
double WithoutIntersection(char const *, char const *);
double Intersection(char const *, char const *);
void releaseMemory();
int yylex (void);
void yywarning(const char *);
void yyerror(const char *);
%}

%union
{
   char* chr;
   int val;
   double dVal;
   struct symrec *var;
}

%token <val> NUM
%token <var> VAR
%type <val> signedValue
%type <var> exp
%type <dVal> calculations
%left '-' '+'
%left '*' '/'
%left NEG

%%
input: /*tyhi rida*/
  | input line
;

line: '\n'                    {printf("tyhi\n");}
```

```
  | exp '\n'
  | calculations '\n'           {printf("%4.2lf\n", $1);}
;

calculations: exp '+' exp       {$$ = SumOfArea($1->cName,$3-
>cName);}
   | exp '-' exp                {$$ = FirstWithoutTheSecond($1-
>cName,$3->cName);}
   | exp '/' exp                {$$ = WithoutIntersection($1-
>cName,$3->cName);}
   | exp '*' exp                {$$ = Intersection($1->cName,$3-
>cName);}
;

exp: VAR
{printf("%s(%d;%d)(%d;%d)\n",
$1->cName,

$1->x1,

$1->y1,

$1->x2,

$1->y2);}
   | VAR '(' signedValue ';' signedValue ')'
         '(' signedValue ';' signedValue ')'    {$1->x1 = $3;
                                                 $1->y1 = $5;
                                                 $1->x2 = $8;
                                                 $1->y2 = $10;
                                                 printf("%s
inserted\n", $1->cName);}
;

signedValue: NUM                                {$$ = $1;}
   | '-' NUM %prec NEG                             {$$ = - $2;}
;
%%

struct symrec *sym_table;

struct symrec *getsym(char const *s_name)
{
  struct symrec *ptr = sym_table;
  while(ptr != NULL)
  {
    if(strcmp(ptr->cName, s_name) == 0)
      return ptr;
    ptr = ptr->pNext;
  }
  return NULL;
}

struct symrec *putsym(char const *s_name)
{
```

```
  struct symrec *ptr = (struct symrec *)malloc(sizeof(struct
symrec));
  ptr->cName = (char *)malloc(strlen(s_name)+1);
  ptr->x1 = 0;
  ptr->y1 = 0;
  ptr->x2 = 0;
  ptr->y2 = 0;
  strcpy(ptr->cName, s_name);
  ptr->pNext = sym_table;
  sym_table = ptr;
  return ptr;
}

//
// Calculates the area covered by two circles.
//
double SumOfArea(char const* cFirstName, char const* cSecondName)
{
    struct symrec *sfirstPoint = getsym(cFirstName);
    struct symrec *sSecondPoint = getsym(cSecondName);

    double firstRadius = sqrt(pow((sfirstPoint->x2 - sfirstPoint-
>x1),2)
                          +pow((sfirstPoint->y2 - sfirstPoint-
>y1),2));
    double secondRadius = sqrt(pow((sSecondPoint->x2 - sSecondPoint-
>x1),2)
                          +pow((sSecondPoint->y2 - sSecondPoint-
>y1),2));

    double firstArea = pow(firstRadius,2) * 3.14;
    double secondArea = pow(secondRadius,2) * 3.14;
    double AB = sqrt(pow((sSecondPoint->x1 - sfirstPoint->x1),2)
                +pow((sSecondPoint->y1 - sfirstPoint->y1),2));
    if( AB == 0)
    {
         return (firstArea <= secondArea ? secondArea : firstArea);
    }
    else if( AB >= secondRadius + firstRadius)
    {
         double pindala = firstArea + secondArea;
         return pindala;
    }
    else
    {
         double intersection = Intersection(cFirstName,
cSecondName);
         double pindala = firstArea + secondArea - intersection;
         return pindala;
    }
}

//
// Calculates the area covered by first circle without the second
circle coverage.
//
```

```
double FirstWithoutTheSecond(char const* cFirstName, char const*
cSecondName)
{
    struct symrec *sfirstPoint = getsym(cFirstName);
    struct symrec *sSecondPoint = getsym(cSecondName);
    double firstRadius = sqrt(pow((sfirstPoint->x2 - sfirstPoint-
>x1),2)
                           +pow((sfirstPoint->y2 - sfirstPoint-
>y1),2));
    double secondRadius = sqrt(pow((sSecondPoint->x2 - sSecondPoint-
>x1),2)
                           +pow((sSecondPoint->y2 - sSecondPoint-
>y1),2));
    double firstArea = pow(firstRadius,2) * 3.14;
    double secondArea = pow(secondRadius,2) * 3.14;
    double AB = sqrt(pow((sSecondPoint->x1 - sfirstPoint->x1),2)
                 +pow((sSecondPoint->y1 - sfirstPoint->y1),2));

    if( AB == 0)
    {
         return (secondArea <= firstArea ? firstArea - secondArea :
0);
    }
    else if( AB >= secondRadius + firstRadius)
    {
         return firstArea;
    }
    else
    {
        double intersection = Intersection(cFirstName, cSecondName);
        double pindala = firstArea - intersection;
        return pindala;
    }
}


//
// Calculates the area covered by two circles without the
intersection area.
//
double WithoutIntersection(char const* cFirstName, char const*
cSecondName)
{
    struct symrec *sfirstPoint = getsym(cFirstName);
    struct symrec *sSecondPoint = getsym(cSecondName);
    double firstRadius = sqrt(pow((sfirstPoint->x2 - sfirstPoint-
>x1),2)
                           +pow((sfirstPoint->y2 - sfirstPoint-
>y1),2));
    double secondRadius = sqrt(pow((sSecondPoint->x2 - sSecondPoint-
>x1),2)
                           +pow((sSecondPoint->y2 - sSecondPoint-
>y1),2));
    double firstArea = pow(firstRadius,2) * 3.14;
    double secondArea = pow(secondRadius,2) * 3.14;
    double AB = sqrt(pow((sSecondPoint->x1 - sfirstPoint->x1),2)
```

```
                +pow((sSecondPoint->y1 - sfirstPoint->y1),2));

    if( AB == 0)
    {
         return (secondArea <= firstArea ? firstArea - secondArea
                                         : secondArea - firstArea);
    }
    else if( AB >= secondRadius + firstRadius)
    {
         return (firstArea + secondArea);
    }
    else
    {
         double sumOfArea = SumOfArea(cFirstName,cSecondName);
         double intersection = Intersection(cFirstName,
cSecondName);
         double pindala = sumOfArea - intersection;
         return pindala;
    }
}

//
// Calculates the two cirles intersection area.
//
double Intersection(char const* cFirstName, char const* cSecondName)
{
    struct symrec *sfirstPoint = getsym(cFirstName);
    struct symrec *sSecondPoint = getsym(cSecondName);

    double firstRadius = sqrt(pow((sfirstPoint->x2 - sfirstPoint-
>x1),2)
                           +pow((sfirstPoint->y2 - sfirstPoint-
>y1),2));
    double secondRadius = sqrt(pow((sSecondPoint->x2 - sSecondPoint-
>x1),2)
                           +pow((sSecondPoint->y2 - sSecondPoint-
>y1),2));

    double firstArea = pow(firstRadius,2) * 3.14;
    double secondArea = pow(secondRadius,2) * 3.14;

    double AB = sqrt(pow((sSecondPoint->x1 - sfirstPoint->x1),2)
                +pow((sSecondPoint->y1 - sfirstPoint->y1),2));

    double smallerRadius;
    double biggerRadius;

    if (firstRadius < secondRadius)
    {
        smallerRadius = firstRadius;
        biggerRadius = secondRadius;
    }
    else
    {
        smallerRadius = secondRadius;
        biggerRadius = firstRadius;
```

```
    }

    if( AB == 0)
    {
         return (firstArea <= secondArea ? firstArea : secondArea);
    }
    else if( AB >= firstRadius + secondRadius)
    {
         return 0;
    }
    else
    {
        double pindala = (3.14*pow(smallerRadius,2))
                        - (pow(smallerRadius,2) * (acos(0.5*(
pow(biggerRadius,2) - pow(smallerRadius,2) -
pow(AB,2))/(AB*smallerRadius))))
                        + (pow(biggerRadius,2) * (acos(0.5*(
pow(biggerRadius,2) - pow(smallerRadius,2) +
pow(AB,2))/(AB*biggerRadius))))
                        - (0.5 * biggerRadius * smallerRadius *
sqrt(4 - pow((pow(AB,2) - pow(biggerRadius,2) -
pow(smallerRadius,2)),2) / (pow(biggerRadius,2) *
pow(smallerRadius,2))));
        return pindala;
    }
}


//
// Free allocated structs and elements memory
//
void releaseMemory()
{
     struct symrec *ptrNext;
     struct symrec *ptr = sym_table;
     ptrNext = ptr;
     while(ptrNext != NULL)
     {
          ptrNext = ptr->pNext;
          free(ptr->cName);
          ptr->cName = NULL;
          free(ptr);
          ptr = ptrNext;
     }
}


#include <ctype.h>
int yylex(void)
{
  int c;
  do
    c= getchar();
  while(c == ' ' || c == '\t');

  if(isdigit(c) || c == '.')
```

```
  {
    ungetc(c, stdin);
    double dNumber;
    scanf("%lf", &dNumber);
    if ((dNumber - (int)dNumber) != 0)
       yywarning("Value not integer");

    yylval.val = (int)dNumber;
    return NUM;
  }

  if(isalpha(c))
  {
    int length = 20;
    int i = 0;
    char *s = (char *)malloc(length);
    do
    {
      if (i == length -1)
      {
           length = length * 2;
         s = (char *)realloc(s, length);
      }
      s[i] = c;
      c= getchar();
      i++;
    }while(isalpha(c));
    ungetc(c, stdin);
    s[i]='\0';
    struct symrec *ptr = getsym(s);
    if(ptr == NULL)
      ptr = putsym(s);

    free(s);
    yylval.var = ptr;
    return VAR;
  }

  if(c == EOF)
  {
       releaseMemory;
       return 0;
  }
  return c;
}
void yywarning(const char *s)
{
  printf("warning: %s\n", s);
}

void yyerror(const char *s)
{
  printf("%s\n", s);
  releaseMemory;
}
int main(void)
```

```
{
  sym_table = NULL;
  return yyparse();
}
```