Univerzitet u Sarajevu
Prirodno-matematički fakultet
Odsjek za matematiku

Dokumentacija za Projekat 1 iz predmeta Strukture podataka i algoritmi

Student:

Edib Šupić

Interfejs klase **Polinom** dat je na slici 1. Klasa pored privatnog atributa *vector<double>* **skup**, koji čuva polinom, sadrži i privatni atribut *mutable int* **monomCount** u kojem se cuva broj nenultih koeficijenata polinoma. Također vidimo da klasa sadrži i 3 privatne metode: *void* **inputFun**(*string* **s1**), *void* **checkSkup**() i *void* **nonZero**() *const. Funkcija void* **checkSkup**() modificira vektor **skup** tako da zadnji elemenat bude nenulti, jer se ostale funkcije oslanjaju na tu činjenicu **.** Funckija *void* **nonZero**() *const* broji nenulte koeficijente polinoma i rezultat smješta u **monomCount**, u nastavku ćemo vidjeti zbog čega. Implementacija pomenutih funkcija je trivijalna i neće se spominjati. Što se tiče funkcije *void* **inputFun**(*string* **s1**) njena uloga će biti objašnjena kada dođemo do operatora **>>** jer je ona pomoćna funkcija pomenutog operatora.

```
1  class Polinom {
2  private:
3          mutable int monomCount = 0;
4          vector<double> skup;
5          void inputFun( string s1 );
6          void checkSkup();
7          void nonZero() const;
8  public:
9          Polinom() { skup.resize(1); skup[0]=0;}
10         Polinom(int n, int k=1) : monomCount(1) {
11                 if (k == 0 || n < 0) throw invalid_argument("Neispravan stepen");
12                 skup.resize(n+1);
13                 skup[n]=k;
14         }
15         Polinom( vector<double> v ) : skup(v) { checkSkup(); nonZero(); }
16         double& operator[]( int i ) { return skup[i]; }
17         const double& operator[]( int i ) const { return skup[i]; }
18         unsigned int size() const { return skup.size(); }
19         double operator()( double x ) const;
20         Polinom operator^( int n ) const;
21         double NulaPolinoma( double a, double b, double e=1e-7 ) const;
22         friend Polinom operator+( const Polinom &p1, const Polinom &p2 );
23         friend Polinom operator-( const Polinom &p1, const Polinom &p2 );
24         friend Polinom operator*( const Polinom &p1, const Polinom &p2 );
25         friend bool operator<( const Polinom &p1, const Polinom &p2 );
26         friend bool operator>( const Polinom &p1, const Polinom &p2 );
27         friend bool operator==( const Polinom &p1, const Polinom &p2 );
28         friend istream& operator>>( istream &flow, Polinom &p1 );
29         friend ostream& operator<<( ostream &flow, const Polinom &p1 );
30 };
```

Slika 1.

Klasa sadrži 3 konstruktora: **Polinom**(), **Polinom**(*int* **n,** *int* **k=**1) i **Polinom**(*vector<double>* **v**). **Polinom**() kreira nulu polinom i koristi se samo u specijalnim slučajevima. **Polinom**(*int* **n,** *int* **k=**1) kreira jednočlani polinom stepena **n** i koeficijenta **k**, pri čemu baca izuzetak za nepravilno unesene **n** i **k**, primjenu ovo konstruktora ćemo vidjeti u implementaciji operatora ***.** **Polinom**(*vector<double>* **v**) je uveden radi lakšeg testiranja klase i nije potreban sa njeno pravilno funkcionisanje. Pored traženih operatora implementirani su i operatori: **[], []** *const*, **+**, **-**, **<**, **>** i **==.** Operatori **[]**, **[]** *const* zajedno sa funkcijom *unsigned int* **size**() *const* omogućavaju objektima tipa **Polinom** da se ponašaju identično kao vektori u određenim situacijama, njihova implementacija je trivijalna. Operatori **+** i **–** su implementirani zbog operatora ***.** Operator **+** (slika 2.) pravi pomoćni **Polinom** i u njega kopira veći od polinoma **p1** i **p2** nakon čega sabira koeficijente jedan po jedan, po završetku poziva funkcije **checkSkup**() i **nonZero**() kako bi doveo **skup** u ispravan oblik i postavio **monomCount**. Potreba za ovim pozivima dolazi iz činjenice da se koeficijenti mogu poništiti i u situaciji kad se svi poništе vraća nula polinom, u suprotom vraća pomoćni **Polinom**.

```
1  Polinom operator+(const Polinom &p1, const Polinom &p2) {
2          Polinom tempP;
3          if (p1.size() < p2.size()) {
4                  tempP = p2;
5                  for (unsigned int i=0; i<p1.size(); i++) tempP[i] += p1[i];
6          }
7          else {
8                  tempP = p1;
9                  for (unsigned int i=0; i<p2.size(); i++) tempP[i] += p2[i];
10         }
11         tempP.checkSkup();
12         tempP.nonZero();
13         if (tempP.size() == 0) return Polinom();
14         return tempP;
15 }
```

Slika 2.

Operator **–** kopira polinom **p2** u pomoćni **Polinom**, mijenja predznake koeficijenata, a zatim sabira pomoćni **Polinom** sa polinomom **p1**. Operator **<** (slika 3.) je uveden kako bi se niz **Polinoma** mogao sortirati uz pomoć **sort**() funkcije iz biblioteke *algorithm* i on mijenja funkciju koja prima niz **Polinoma** i sortira ih po veličini. Poređenje se vrši koeficijent po koeficijent za polinome istog stepen, a za polinome različitih stepena porede se veličine vektora **skup**. Ovdje vidimo zbog čega je uvedena funkcija **checkSkup**().

```
1  bool operator<(const Polinom &p1, const Polinom &p2) {
2          if (p1[p1.size()-1] * p2[p2.size()-1] < 0) {
3                  if (p1[p1.size()-1] < 0) return true;
4                  else return false;
5          }
6          else if (p1.size() < p2.size()) return true;
7          else if (p1.size() > p2.size()) return false;
8          else {
9                  for (unsigned int i=0; i<p1.size(); i++) {
10                         if (p1[i] < p2[i]) return true;
11                         else if (p1[i] > p2[i]) return false;
12                 }
13         }
14         return false;
15 }
```

Slika 3.

Operatori **>** i **==** implementirani su radi kompletnosti. Operator **>** implementiran je identično kao i operator **<** dok je operator **==** definisan preko operatora **<** i **>**. Kada smo objasnili uloge dodatnih funkcija i operatora možemo preći na implementacije traženih funkcionalnosti klase. Od traženih operatora definisali smo: **()**, *****, **^**, **>>** i **<<.** Operator **()** računa vrijednost polinoma u datoj tački i njegova implementacija je trivijalna. Operator ***** (slika 4.) množi dva polinoma strategijom podijeli pa vladaj. Prvo ćemo objasniti ideju, a zatim implementaciju. Neka su A i B polinomi koje množimo, tada možemo polinom A rastaviti na dva polinoma. Polinom stepena manjih od n/2(A0) i polionom stepena većih od n/2(A1), gdje je n stepen polinoma A. Ako iz polinoma A1 izvučemo x^(n/2) dobijamo da je A = A0 + A1*x^(n/2), na isti način definišemo i B = B0 + B1*x^(n/2). Sada je A*B = A0*B0 + A0*B1*x^(n/2) + A1*B0*x^(n/2) + A1*B1*x^n. Dobijamo 4 potproblema veličine n/2 koje možemo rekurzivno riješiti. Međutim lako se dokaže da ovakav pristup ne ubrzava vrijeme izvršavanja u odnosu na brute force pristup. Ideja je da ne rješavamo 4 potproblema već 3. Neka je Y = (A0+A1)*(B0+B1), U = A0*B0 i Z = A1*B1 tada A0*B1 + A1*B0 možemo računati kao Y – U – Z = A0*B1 + A1*B0 što je očigledno tačno. Sada rješavamo samo 3 potproblema veličine n/2 i dobijamo traženo vrijeme izvršavanja n^log3. Sada ćemo objasniti implementaciju. Rekurzivni poziv operatora ***** završava se kad jedan od polinoma **p1** ili **p2** postanu nula

polinomi i tada je rezultat nula polinom ili kada jedan od polionma **p1** ili **p2** postane jednočlnai polinom i tada je rezultat proizvod svih koeficijenata drugog polionma sa koeficijentom jednočlanog. Zahvaljujući činjenici da se polinom čuvao kao vektor množenje nekog **Polinoma** sa jednočlnaim polinomom ekvivalento je pomjeranju svih koeficijenata za n mjesta u desno gdje je n stepen jednočlanog polinoma i množenjem sa k gdje je k koeficijent jednočlanog polinoma. U opštem slučaju tražimo prvo sredine **Polinoma** A i B i pravimo polinome **A0**, **A1**, **B0** i **B1** kao što je objašnjeno u ideji, a zatim računamo **Y** = (**A0** + **A1**)*(**B0** + **B1**), **U** = **A0***B0** i **Z = A1***B1**. Konačni rezulatat dobijamo računanjem izraza **U** + (**Y – U – Z**)***x** + **Z*x*x**, gdje je **x** jednočlani polinom stepena n/2 i koeficijenta 1.

```
1  Polinom operator*(const Polinom &p1, const Polinom &p2) {
2          if (p1.monomCount == 0 || p2.monomCount == 0) return Polinom();
3          else if (p1.monomCount == 1) {
4                  Polinom tempP(p1.size()+p2.size()-2);
5                  for (unsigned int i = p1.size()-1; i < tempP.size(); i++) {
6                          tempP[i] = p2[i - p1.size() + 1] * p1[p1.size() - 1];
7                  }
8                  tempP.monomCount = p2.monomCount;
9                  return tempP;
10         }
11         else if (p2.monomCount == 1) {
12                 Polinom tempP(p1.size()+p2.size()-2);
13                 for (unsigned int i = p2.size()-1; i < tempP.size(); i++) {
14                         tempP[i] = p1[i - p2.size() + 1] * p2[p2.size() - 1];
15                 }
16                 tempP.monomCount = p1.monomCount;
17                 return tempP;
18         }
19         else {
20                 int midP1 = p1.size() / 2, midP2 = p2.size() / 2;
21                 int midP = min(midP1, midP2);
22                 Polinom A0(midP - 1), A1(p1.size() - midP - 1);
23                 Polinom B0(midP - 1), B1(p2.size() - midP - 1);
24                 for (unsigned int i = 0; i < A0.size(); i++) A0[i] = p1[i];
25                 A0.nonZero();
26                 for (unsigned int i = 0; i < A1.size(); i++) A1[i] = p1[i + midP];
27                 A1.nonZero();
28                 for (unsigned int i = 0; i < B0.size(); i++) B0[i] = p2[i];
29                 B0.nonZero();
30                 for (unsigned int i = 0; i < B1.size(); i++) B1[i] = p2[i + midP];
31                 B1.nonZero();
32                 Polinom Y = (A0+A1)*(B0+B1);
33                 Polinom U = A0*B0;
34                 Polinom Z = A1*B1;
35                 Polinom x(midP);
36                 return U + (Y - U - Z)*x + Z*x*x;
37         }
38 }
```

Slika 4.

Ovdje vidimo razlog za implementaciju operatora **+** i **–** kao i konstruktora jednočlanog polinoma. Operator **^** rekurzinvno računa stepen polinoma uz pomoć operatora ***** za n>0 u suprotom baca izuzetak. Ideja je da polinom stepena n možemo računati kao proizvod dva polinoma stepena n/2 i njegova implementacija je trivijalna. Operator **>>** učitava uneseni polinom u *string* i poziva funkciju **inputFun**() koja obavlja provjeru unosa. Sada ćemo objasniti funkciju **inputFun**(). Iz stringa prvo brišemo prazna mjesta radi jednostavnosti i čistimo **skup**. Polinom se rastavlja na monome pri čemu je monom niz znakova između dva + i/ili -. Uz pomoć regularnih izraza provjerava se format monoma, regularni izrazi neće

biti objašnjavani, nakon što se utvrdi format monoma iz njega se čitaju stepen i koeficijent i spremaju u **skup** na odgovarajuće mjesto. Ukoliko je stepen monoma veći od veličine **skupa**, veličina skupa se postavlja tako da je uneseni stepen zadnji u **skupu**. Primjetimo da ovo može biti veoma neefikasno ukoliko se polinom unosi od najmanjeg stepena prema najvećem jer se za svaki pročitani monom **skup** mora ponovo alocirati i moraju se kopirati koeficijenti. Međutim ovo je veoma efikasno ako se prvo unese najveći stepen jer tada imamo samo jednu alokaciju **skupa**. Ukoliko jedan od monoma nije ispravnog formata **skup** se briše i baca se izuzetak. Na kraju se pozivaju funkcije **nonZero**() i **checkSkup**() zbog situacija kada se neki monomi ponište. Operator **<<** je realiziran preko if petlji koje uzimaju u obzir sve varijacije prilikom ispisa polinoma kako bi se osigurao tačan i prirodan ispis polinoma. Polinom se ispisuje koeficijent po koeficijent počeviši od zadnjeg. Metoda *double* **NulaPolinoma**(*dobule* **a**, *double* **b**, *double* **e**) *const* (slika 5.) implementirana je kao binarna pretraga na segmentu [**a**,**b**]. Nakon što ustanovimo u kojoj od tačaka **a** ili **b** je funkcija negativna počinjemo sa binarnom pretragom. Ako je vrijednost funkcije u tački **mid** = (**a**+**b**)/2 veća od nule postavljamo tačku u kojoj je funkcija pozitivna na **mid** u suprotom postavljamo tačku u kojoj je funkcija negativna na **mid**. Proces ponavljamo dok ne dodjemo do tražene preciznosti. U slučaju da je vrijednost funkcije istog predznaka i u tački **a** i u tački **b** dolazi do izuzetka.

```
1  double Polinom::NulaPolinoma(double a, double b, double e) const {
2          if (((*this)(a) < 0 && (*this)(b) < 0) || ((*this)(a) > 0 && (*this)(b) > 0)) {
3                  throw invalid_argument("Polinom nema nula na intervalu [a,b]!");
4          }
5          double mid = (a+b)/2;;
6          if ((*this)(a) <= 0) {
7                  while (abs((*this)(mid)) > e) {
8                          if ((*this)(mid) < 0) a = mid;
9                          else b = mid;
10                         mid = (a+b)/2;
11                 }
12         }
13         else {
14                 while (abs((*this)(mid)) > e) {
15                         if ((*this)(mid) > 0) a = mid;
16                         else b = mid;
17                         mid = (a+b)/2;
18                 }
19         }
20         return mid;
21 }
```

Slika 5.

Funkcija *double* **PresjekPolinoma**(*const* **Polinom** &**p1**, *const* **Polinom** &**p2**, *dobule* **a**, *double* **b**, *double* **e**) definisana je identično kao **NulaPolinoma**. Prvo provjeravamo koja funkcija je manja u tački **a**, a zatim ukoliko je ta funkcija idalje manje u tački **mid** = (**a**+**b**)/2 postavljamo **a** = **mid** u suprotom postavljamo **b** = **mid**. Proces se ponavlja do željene preciznosti. Ovim smo ukratko objasnili implementacije svih traženih funkcionalnosti klase **Polinom**.

Reference:

http://www.cse.ust.hk/~dekai/271/notes/L03/L03.pdf

http://www.cplusplus.com/