Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Student: Ilija Bešlin, SV71/2021

Predmet: Objektno orijentisano programiranje 2

Broj projektnog zadatka: Individualni projekat

Tema projektnog zadatka: Implementacija igre ’Moj broj’

# **Rad U/I podsistema (učitavanje i ispis/upis)**

Pri pokretanju programa funkcijom ‘**loadRoundsFromFile**’ učitavamo vektor rundi iz ulaznog fajla koje ćemo kasnije koristiti u toku igre.

Na početku igre pomoću funkcije ’**clearOutFile**’ čisti se izlazni fajl “rezultati.txt” kako bi mogli upisati nove prikupljene podatke.

Nakon svake runde pomoću funkcije ‘**writeRoundToFile**’ upisujemo podatke prikupljene o toj rundi u izlazni fajl “rezultati.txt”.

U toku svake runde konzolno je prikazano trenutno stanje, ponuđeni brojevi, traženi broj i izraz(broj) koji je izračunao računar.

Nakon završene igre podatke o broju pobeda, kao i o konačnom pobedniku pišemo na kraju izlaznog fajla “rezultati.txt“ pomoću funkcije ’**writeResultToFile**’.

# **Spisak svih klasa, izuzetaka i slobodnih funkcija.**

Korištene klase: **Calculator, Round(**runda**), Game(**igra**)**

Unutar klase Calculator korištene su 2 pomoćne klase:

* **Token**(Predstavlja broj ili operaciju)
* **TokenStream**(Niz Tokena)

Klasa Calculator može baciti exception **invalid\_argument** ukoliko nije unet dobar izraz za računanje.

Korištene slobodne funkcije:

* **loadRoundsFromFile**
* **writeRoundToFile**
* **writeResultToFile**
* **clearOutFile**

# **Objašnjenje najbitnijih atributa, klasa i funkcija članica, slobodnih funkcija i izuzetaka.**

Klasa **Calculator** predstavlja implementiran kalkulator sa predavanja kao template koji prima argument T – sa kojim tipom podataka se radi.

Funkcije članice:

* T **calculate**(std::string izraz)
* Računa vrednost prosledjenog izraza i vraća dobijenu vrednost
* Baca exception **invalid\_argument** u slucaju lose unetog izraza(ukoliko je unet tip koji ne odgovara zadatom, fali zagrada, nije u dobrom formatu izraz)

Formalna gramatika

* T **number**() //N -> realan broj
* T **primary**() //P -> moze biti N ili (E)
* T **term**() //množi/deli
* T **expression**() //E

Unutrašnje pomoćne klase:

* **Token**(Predstavlja broj ili operaciju)
* Atribut ‘char kind’ predstavlja oznaku za broj(“8”) ili operaciju
* Ukoliko je u pitanju broj unutar atributa ‘T value’ se nalazi vrednost broja
* **TokenStream**(Niz Tokena)
* Atribut ‘Token buffer’ je korišten za čuvanje tokena
* Funkcija članica ’void putBack(Token token)’ vraća pročitani token nazad u buffer

Klasa **Round** predstavlja jednu odigranu igru

Atributi:

* brojRunde
* ponudjeniBrojevi
* trazeniBrojevi
* prviIgracNaPotezu
* izrazA
* dobijeniBrojA
* izrazB
* dobijeniBrojB
* izrazProg
* dobijeniBrojProg
* pobednik

Najvažnije funkcije članice:

* bool **validateInput**(std::string unos)
* Proverava unos korisnika
* vraca true/false u slucaju dobrog/loseg unosa
* void **findSolution**(std::vector<std::string>& ponudjeniBrojevi, int& trazeniBroj,

int\* pronadjenaVrednost, std::string\* pronadjenIzraz)

* Funkcija radi tako sto trazenjem azurira pokazivace unutar rekurzije
* Parametri:

1 - brojevi pomocu kojih racunamo

2 - broj koji treba da dobijemo

3 - pokazivac na trenutnu najblizu vrednost resenja

4 - pokazivac na izraz trenutne najblize vrednosti

Klasa **Game** predstavlja trenutnu igru

Atributi:

* Runde 🡪 niz rundi koje će se odigrati
* brojPobedaA
* brojPobedaB

Najvažnija funkcija članica je ‘void **start**()’ pomoću koje pokrećemo igru

# **Strukture argumenata komandne linije i primer korišćenja**

Argv[0] – filepath

Argv[1] – Naziv fajla

Koristimo da bi prosledili naziv ulaznog fajla iz kojeg učitavamo potrebne brojeve za kreiranje rundi i pokretanje igre

# **Strukture ulazne i izlazne datoteke**

Ulazna datoteka:

* Svaki red predstavlja jednu rundu
* Prvih 6 brojeva su ponudjeni brojevi, a 7. broj je trazeni
* Primer: 3 3 25 50 75 100 996

Izlazna datoteka:

* Niz podataka o svakoj odigranoj rundi
* Na kraju se nalazi rezultat odigrane igre
* Primer:

---------- RUNDA 1 ----------

Ponudjeni brojevi: 1 1 1 1 10 10

Trazeni broj: 106

Izraz igraca A: 10\*10

Dobijeni broj igraca A: 100

Razlika u odnosu na trazeni broj: 6

Izraz igraca B: (10+1)\*10-1-1-1

Dobijeni broj igraca B: 107

Razlika u odnosu na trazeni broj: 1

Pobednik: B

Izraz koji je pronasao racunar: ((1+10)\*10)-(1+(1+1))

Dobijeni broj racunara: 107

# **Opis algoritma za pronalaženje tačnog rešenja**

Korištena funkcija **findSolution** je navedena ranije.

Njoj prosledjujemo brojeve koje računamo, broj koji tražimo, pokazivač na najbližu vrednost rešenja i pokazivač na izraz najbliže vrednosti

**Algoritam**:

1. Pravimo sve parove učitanih brojeva (e1,e2) i primenjujemo operacije +, -, /, \* nad njima.
2. Računamo dobijenu vrednost i shodno rezultatu radimo sledeće

* Ukoliko je operacija ‘/’ i dobijemo decimalni broj, preskačemo ovaj slučaj
* Ukoliko je operacija ‘/’ i drugi broj e2 je jednak 1, preskačemo jer se ne menja vrednost
* Ukoliko je operacija ’-’ i dobijemo negativan broj, preskačemo jer možemo dobiti tu vrednost u drugom smeru oduzimanjem. Primer: ( 3-5 = -2 <=> -(5-3) = -2 )
* Ukoliko je operacija ‘\*’ i neki od brojeva je 1, preskačemo jer se ne menja vrednost
* Ukoliko je dobijeni broj jednak nuli, preskačemo
* Ukoliko je pronađena vrednost jednaka traženom broju prekidamo rekurziju i ažuriramo vrednosti pokazivača
* Ukoliko je pronađena vrednost bliža od prethodno pronađene vrednosti ažuriramo vrednosti pokazivača

1. Pravimo kopiju prosleđenog niza brojeva. Brišemo e1 i e2 iz niza, ali dodajemo novi izraz (e1 op e2) u niz

* Primer:

ponudjeni brojevi = (8 1 9 1 10 75)

e1 = 8, e2 = 9, op = ‘+’

novi niz = (1 1 10 75 (8+9))

1. Sada tražimo rešenje pomoću novodobijenog niza brojeva pozivom funkcije findSolution (rekurzija)

**Napomena**: Izbegavamo množenje i sabiranje 2 puta sa istim vrednostima

Uslovi za **izlaz iz rekurzije** su:

* Ukoliko je pronađena vrednost jednaka traženom broju
* Ukoliko je niz brojeva pomoću kojih računamo jednak 1

Ideja za poboljšanje algoritma:

* Skladištimo sve brojeve koje daju neki rezultat.

Primer 2\*1+3 = 5. Ovde brojevi 1,2 i 3 daju rezultat 5.

* Ovi isti brojevi mogu da daju isti rezultat ali na drugačije načine:

2/1+3, 2+3\*1, 2+3/1, 2\*3-1

* Čuvanjem ovih podataka unutar neke strukture(npr. Trie) lako ćemo proveravati da li su ti brojevi već korišteni za dobijanje određene vrednosti i moći da preskočimo te slučajeve što će znatno ubrzati algoritam.

# **Opis načina testiranja**

Napravljen novi solution u kojem se testira.

**Testiranje kalkulatora**:

* Par primera gde daje dobar rezultat
* Loš ulaz gde fali zagrada
* Loš ulaz gde nije dobar format unosa (fali ‘;q’ na kraju)
* Loš ulaz, ne prihvata slova
* Loš ulaz, int kalkulator ne prihvata decimalne vrednosti

**Testiranje programa za računanje izraza**:

* Nekoliko slučajeva gde nema rešenja (dobije se približna vrednost)
* Nekoliko slučajeva gde ima rešenja (dobije se tražena vrednost)

# **Uočeni problemi i ograničenja**

Ukoliko funkciji za računanje izraza prosledimo više ponuđenih brojeva, a nije moguće izračunati tačan broj može jako dugo trajati izvršavanje.

U našem slučaju gde imamo 6 ponuđenih brojeva moguće je naći tačno rešenje jako brzo ili tražiti malo duže ako se tačna kombinacija brojeva, zagrada i operacija ne nalazi pri početku.

Ukoliko su uneti takvi brojevi da se retko koji slučaj preskoči u 2. koraku algoritma , računanje može trajati duže.