

Факултет Техничких Наука

Трг Доситеја Обрадовића 6,

Нови Сад 106314

Предметни Пројекат из Објектно Оријентисаног Програмирања 2

Слагалица „Мој Број“

Студент:

Душица Трбовић, SV 42/2021

Нови Сад, јануар 2023. год.

САДРЖАЈ

1.Опис пројектног задатка	3
2. Рад у/И подсистема	4
2.1. Читање 7 бројева из задате датотеке	4
2.2. Уписивање података о Игри у крајњу датотеку – резултати.txt	4
3. Списак класа и слободних функција	6
3.1. Класа Израз	6
3.2. Класа Рунда	7
3.3. Класа Рачунар	10
4. Начини тестирања	12
5. Закључак.....	13

1. ОПИС ПРОЈЕКТНОГ ЗАДАТКА

Тема пројектног задатка била је једна од игара из квиза „Слагалица“. Назив игре је Мој Број а задатак је да од 6 понуђених бројева такмичари пронађу тражено решење. У нашој верзији, одржава се полуфинале квиза и 2 такмичара играју 6 рунди. На крају свих 6 рунди се одређује победник. У случају да је једнак број освојених рунди, остаје нерешено.

Главне функционалности које су морале бити имплементирани су:

- Интеракција са корисницима кроз конзолу
- Рачунање израза
- Тражење најбољег решења
- Чување свих података у текстуалној датотеци

Рачунање израза је метода која може бити имплементирана на више начина, али је у складу са нашим потребама и ради оптимизације програма изабран одређен алгоритам.

Неки од одбачених алгоритама за рачунање израза јесте читање стринга са леве на десну страну и памћење операнда, и рачунања операција све док се не дође до краја стринга. Главна мана, уједно и разлог за одбацивање овог начина рачунања израза јесте шта урадити када наш израз има заграде? Још један од начина који је одбачен јесте уписивање израза у бинарно стабло, и пролазак кроз исто све док не останемо само са његовим кореном. Приликом рачунања на овај начин неопходна нам је рекурзија, је бисмо приком наилазак на заграде правили ново стабло чији би корен био чвор на коме се тренутно налазимо. Управо због ове рекурзије, начин је временски веома сличан предходном који смо навели, тако да смо и од њега одустали.

Коначно начин који је коришћен јесте превођење инфиксе нотације у постфиксну. На овај начин се решавамо заграда, и рачунање израза иде „по реду“, тј редом читајући операнде и операције са лева на десно.

2. РАД У/И ПОДСИСТЕМА

2.1. Читање 7 бројева из задате датотеке

Почетак рунде није могао да се изведе без исчитавања једне линије кода из текстуалне датотеке. Текстуална датотека која је креирана унапред, садржи 6 линија са по 7 бројева који су нам неопходни да би се игра Мој Број извршила. Линија улазне датотеке је направљена у фомату 4 једноцифрена броја, 2 двоцифрена при чему први из скупа {10,15,20,25} а други из скупа {25,50,75,100} и на крају се налази број максималне вредности 999, који представља тражено решење, сви подаци одвојени су карактером „;“ (тачка-зarez). Приликом покретања самог програма прослеђује се информација која линија из датотеке треба бити прочитана, на тај начин раздвајамо податке који су спојени у једној текстуалној датотеци.

```
std::string citanje_ulazne_datoteke(std::string ulazna_datoteka, int line) {
    std::fstream newfile;
    int line_now = 1;
    newfile.open(ulazna_datoteka, std::ios::in);
    if (newfile.is_open()) {
        std::string tp;
        while (getline(newfile, tp)) {
            if (line_now == line) {
                return tp;
            }
            line_now++;
        }
        newfile.close();
    }
}
```

Да бисмо били у могућности да креирамо бројеве од једне линије коју смо добили као повратну вредност предходне функције, била нам је неопходна функција за раздвајање стринга. Функција *parse_string* као повратну вредност враћа листу стрингова, које је добила раздвајањем једне линије из текстуалне датотеке по карактеру „;“.

```
std::list<std::string> parse_string(std::string text, std::string delimiter) {
    std::list<std::string> new_text;
    size_t pos = 0;
    std::string token;
    while ((pos = text.find(delimiter)) != std::string::npos) {
        token = text.substr(0, pos);
        new_text.push_back(token);
        text.erase(0, pos + delimiter.length());
    }
    new_text.push_back(text);
    return new_text;
}
```

2.2. Уписивање података о Игри у крајњу датотеку – резултати.txt

Након сваке рунде нови подаци су се уписивали у текстуалну датотеку. Пре самог уписа, неопходно је исцитати све податке који су се налазили у датотеци и запамтити их у једну променљиву типа стринг, како подаци о предходним рундама не би били обрисани. Затим за сваку рунду правимо нови стринг који ће представљати нови упис, тј нови ред у текстуалној датотеци. Стринг садржи податке о рунди, што је заправо: редни број рунде, бројеве који су били дати такмичарима, резултат који се тражио,

израз који је унео играч А, резултат израза А, одступање резултата А од траженог резултата, израз који је унео играч Б, резултат израза Б, одступање резултата Б од траженог резултата, победника рунде, израз који је компјутер пронашао, решење израза који је компјутер пронашао.

```
void upis_u_fajl(Runda runda, std::list<std::string> new_podaci, Racunar
racunar){

    std::vector<std::string> text = citanje_iz_fajla();
    std::string new_text = "RUNDA: " + int_to_str(runda.get_broj_runde()) +
";";

    for (const auto& str : new_podaci) {
        new_text = new_text + str + ";";
    }

    new_text = new_text + "PODACI O RUNDI: " +
runda.get_izraz_takmicar_a().get_izraz() + ";";
    new_text = new_text +
int_to_str(runda.get_izraz_takmicar_a().get_rezultat()) + ";";
    new_text = new_text + int_to_str(runda.get_odstupanje_a()) + ";";

    new_text = new_text + runda.get_izraz_takmicar_b().get_izraz() + ";";
    new_text = new_text +
int_to_str(runda.get_izraz_takmicar_b().get_rezultat()) + ";";
    new_text = new_text + int_to_str(runda.get_odstupanje_b()) + ";";

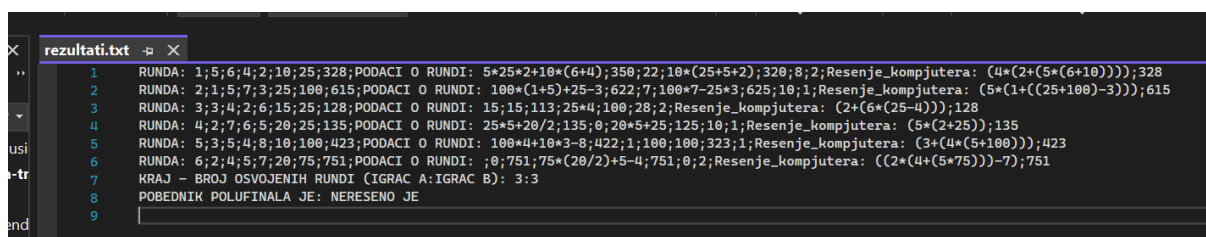
    new_text = new_text + int_to_str(runda.get_pobednik_runde()) + ";";
    new_text = new_text + "Resenje_kompjuteru: " +
racunar.get_izraz().get_izraz() + ";" +
int_to_str(racunar.get_izraz().get_rezultat());

    text.push_back(new_text);

    std::fstream newfile;
    newfile.open("rezultati.txt", std::ios::out);
    if (newfile.is_open())
        for (const auto& str : text) {
            newfile << str + '\n';
        }
    newfile.close();
}
```

На самом крају програма у текстуалну датотеку се уписују и подаци о крајњем резултату, тј број освојених рунди Играча А : Играча Б, као и коначан победник. Две нове линије се након креирања стринга додају у резултати.txt.

```
std::vector<std::string> text = citanje_iz_fajla();
std::string new_text = "KRAJ - BROJ OSVOJENIH RUNDI (IGRAC A:IGRAC B): " +
int_to_str(pobednik_a) + ':' + int_to_str(pobednik_b) + '\n' + "POBEDNIK
POLUFINALA JE: " + konacan_pobednik;
```



Слика 1 - Изглед текстуалне датотеке резултати.txt

3. СПИСАК КЛАСА И СЛОБОДНИХ ФУНКЦИЈА

Програм је подељен на три класе, од којих класа Израз представља помоћну класу. Преостале две класе су: Рунда и Рачунар.

3.1. Класа Израз

Класа садржи три атрибута израз у инфиксном облику, израз у постфиксном облику и резултат изрази. Поред гет и сет метода, још неке методе које су битне за ову класу су:

```
int prioritet(char q);
std::string convert_to_postfix();
std::string convert_to_infix();
int izracunaj_rezultat();
```

Метода *приоритет* одређује која операција има већи приоритет, тј помаже нам приликом пребацивања изрази у постфиксни облик и брисања заграда. Метода *convert_to_postfix* пребацује израз из инфиксног облика у постфиксни облик и олакшава рачунање изрази. Алгоритам за пребацивање јесте:

- Кренемо са читањем стринга у коме се налази израз, карактер по карактер
- Ако је скенирани знак операнд, преписујемо га у посебан стринг, који на крају прослеђујемо као повратну вредност
- Ако није:
 - Ако је приоритет скенираног оператора већи од приоритета оператора у стеку (или је стек празан), додајемо оператор на стек
 - У супротном, избацујемо све операторе који имају већи или једнак приоритет од скенираног оператора (што би за на пример оператор * били оператори +, - и /). Након избацивања можемо да убацимо тренутни оператор на стек. (У случају да наиђемо на заграду приликом избацивања оператора на стеку, стопирамо избацивање и убацимо тренутни оператор)
- Ако је скенирани знак „(“, стављамо га на стек.
- Ако је скенирани знак „)“, избацујемо један по један оператор и додајемо га на стринг који ће бити повратна вредност све док се не наиђе на „(“, отворена заграда се такође брише са стека у том моменту.
- Понављамо кораке 2 – 6 док се не скенира цео стринг који представља израз.

Infix Expression: (A/(B-C)*D+E)

Symbol Scanned	Stack	Output
((-
A	(A
/	(/	A
(((A
B	((AB
-	((-	AB
C	((-	ABC
)	(/	ABC-
*	(/*	ABC-/
D	(/*	ABC-/D
+	(/+	ABC-/D*
E	(/+	ABC-/D*E
)	Empty	ABC-/D*E+

Postfix Expression: ABC-/D*E+

Слика 2 - Алгоритам пребацивања инфиксног изрази у постфиксан

На сличан начин се постфиксни облик пребацује у инфиксни, водимо рачуна о предности оператора и убацујемо заграде у израз уколико су оне потребне. Последња метода у класи Израз је *izracunaj_rezultat*. Уз помоћу предходне методе која израз пребацује у постфиксни облик, пролазак кроз исти нам је олакшан, исто као и рачунање крајњег резултата. Метода ради тако што се операнди убацују на стек све док се не наиђе на оператор, у том моменту се рачуна вредност изрази (операнд_1 операција операнд_2) и након рачунања стек се празни а затим се резултат додаје на стек. Овај процес се понавља све док не завршимо пролазак кроз стринг који представља израз у постфиксној нотацији.

3.2. Класа Рунда

Атрибути класе Рунда су:

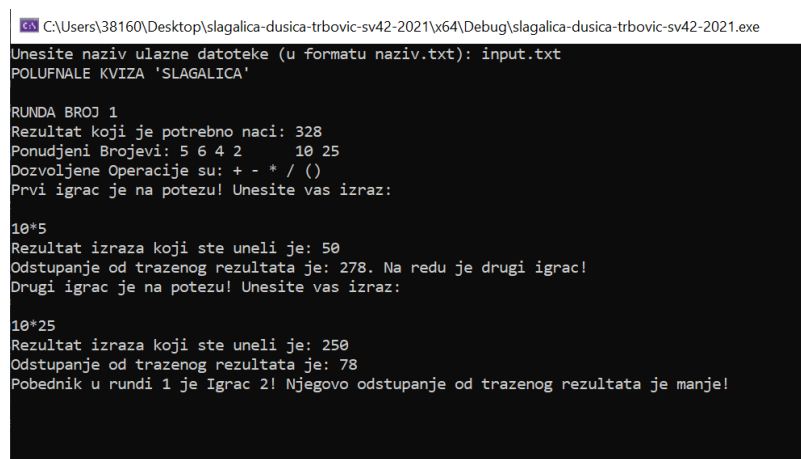
```
int broj_runde;  
int trazeni_rezultat;  
std::list<int> brojevi;  
int broj_x;  
int broj_y;  
  
Izraz izraz_takmicar_a;  
int odstupanje_a;  
Izraz izraz_takmicar_b;  
int odstupanje_b;  
  
int pobednik_runde;
```

Сваки од атрибута је јединствен за сваку рунду и неопходан је за упис који вршимо у крајњу текстуалну датотеку. Поред конструктора, деструктора и гет и сет метода, још неке од битнијих метода ове класе су:

```
void konzola_ispis();  
bool prover_a_unosa(std::string unos);
```

3.2.1. Метода „konzola_ispis()“

Приликом позива методе на екран се исписују општи подаци о рунди, тј број рунде, бројеви који су доступни кориснику за унос изрази и резултат који је потребно пронаћи.



```
C:\Users\38160\Desktop\slagalica-dusica-trbovic-sv42-2021\Debug\slagalica-dusica-trbovic-sv42-2021.exe  
Unesite naziv ulazne datoteke (u formatu naziv.txt): input.txt  
POLUFNALE KVIZA 'SLAGALICA'  
  
RUNDA BROJ 1  
Rezultat koji je potrebno naci: 328  
Ponudjeni Brojevi: 5 6 4 2 10 25  
Dovoljene Operacije su: + - * / ()  
Prvi igrac je na potezu! Unesite vas izraz:  
  
10*5  
Rezultat izraza koji ste uneli je: 50  
Odstupanje od traženog rezultata je: 278. Na redu je drugi igrac!  
Drugi igrac je na potezu! Unesite vas izraz:  
  
10*25  
Rezultat izraza koji ste uneli je: 250  
Odstupanje od traženog rezultata je: 78  
Pobednik u rundi 1 je Igrac 2! Njegovo odstupanje od traženog rezultata je manje!
```

Слика 3 - Испис података о рунди на конзолу

Након исписа података, на реду су играчи. Уколико је рунда парна први на потезу биће Играч Б, у супротном то је Играч А. Рунда се зауставља и други играч нема право на потез уколико је играч који је био први на реду унео тачан израз, тј израз чије решење се поклапа са траженим. Уколико ни један од играча није пронашао тачан израз, победник је онај чије је одступање од крајњег резултата мање, предност има играч који је био први на потезу.

```
RUNDA BROJ 3
Rezultat koji je potrebno naci: 128
Ponudjeni Brojevi: 3 4 2 6      15 25
Dozvoljene Operacije su: + - * / ()
Prvi igrac je na potezu! Unesite vas izraz:

15*(3+4)+25-2
Rezultat izraza koji ste uneli je: 128
Cestitamo. Pronasli ste tacno resenje! Pobedili ste u rundi 3

NASE RESENJE(racunar):
Izraz: (2+(6*(25-4)))=128
```

Слика 4 - Пример прекида рунде приликом уноса правилног израза

Приликом сваког уноса израза позива се метода за проверу истог:

```
std::cout << "Drugi igrac je na potezu! Unesite vas izraz: " << std::endl <<
std::endl;
std::string unos_b;
bool p = false;
while (p == false) {
    std::cin >> unos_b;
    p = provera_unosa(unos_b);
    if (p == false) {
        std::cout << "Unos nije odgovarajuci! Pokusajte ponovo!" << std::endl;
    }
}
```

На крају методе проглашава се победник рунде, док се у *main*-у позива рачунар да пронађе своје решење.

3.2.2. Метода „provera_unosa(std::string unos)“

Провера уноса служи како неко од играча не би унео неки од карактера који није дозвољен. Било то рандом карактер из асциј табеле или број који није понуђен у датој рунди. Такође на овај начин изегавамо могућност израза где би дошло до дељења са нулом.


```

RUNDA BROJ 2
Rezultat koji je potrebno naci: 615
Ponudjeni Brojevi: 1 5 7 3      25 100
Dozvoljene Operacije su: + - * / ( )
Drugi igrac je na potezu! Unesite vas izraz:

6*100+15
Unos nije odgovarajuci! Pokusajte ponovo!

```

Слика 5 - Пример погрешног уноса

Правимо листу свих могућих карактера, и након тога пролазимо кроз унети стринг, ради провере.

```

std::list<int> mogucnosti = {this->broj_x, this->broj_y};
for (auto it = this->brojevi.begin(); it != this->brojevi.end(); ++it) {
    mogucnosti.push_back(*it);
}

std::vector<std::string> mogucnosti_string = {};

for (auto it = mogucnosti.begin(); it != mogucnosti.end(); ++it) {
    std::stringstream stream;
    stream << *it;
    std::string res_str = stream.str();
    mogucnosti_string.push_back(res_str);
}

mogucnosti_string.push_back("+");
mogucnosti_string.push_back("-");
mogucnosti_string.push_back("*");
mogucnosti_string.push_back("/");

```

Упис се проверава кроз петљу, за почетак проверавамо за сваки карактер да ли је бар неки од оних који се налазе у листи могућности, ако није, упис није добар. Али ако јесте, пазимо на то да је сваки карактер коришћен тачно један пут, не рачунајући операнде. Уколико је све у реду, повратна вредност методе је *true*.

```

for (const auto& str : new_text) {
    std::vector<std::string> iskoriscene_vrednosti = {};
    int count = 0;
    if (str != " ") {
        for (const auto& str_mo : mogucnosti_string) {
            if (str_mo == str) {
                int count_ponavljanje = 0;
                for (const auto& str_po : iskoriscene_vrednosti) {
                    if (str_po == str) {
                        count_ponavljanje++;
                    }
                }
            }
        }
        if (str_mo != "+" && str_mo != "-" && str_mo != "*" && str_mo != "/") {
            if (count_ponavljanje == 0) {
                iskoriscene_vrednosti.push_back(str_mo);
                count++;
            }
        }
        else {
            count++;
        }
    }
}

```

3.3. Класа Рачунар

Класа Рачунар за атрибуте има:

```
Izraz izraz;  
int trazeni_rezultat;  
std::list<int> brojevi;  
int broj_x;  
int broj_y;
```

Поред основних гет и сет метода још неке од важнијих су:

```
void racunaj_izraze();  
void ispis_na_konzolu();
```

3.3.1. Метода „racunaj_izraze()“ – ПРОНАЛАЖЕЊЕ ТАЧНОГ РЕШЕЊА

Почетна идеја алгоритама била је слична brute-force алгоритму али уместо састављања свих могућих израза у постфиксној нотацији, решење тражимо тако што: имамо 6 понуђених бројева; додајмо један број са једне стране израза, израз од преосталих 5 бројева са друге стране израза, а у средину убацимо операцију и израчунајмо тај израз. Овакав алгоритам је рекурзиван је се за све изразе са десне стране примењује исти алгоритам и тако у круг. Након свих таквих комбинација, са леве стране би се додавали изрази са по 2 броја, а са десне са преостала 4; затим са 3 броја и тако даље.

У претходном алгоритму се одмах може приметити то да се један те исти израз може рачунати пуно пута (што зна да буде чест проблем у рекурзивним алгоритмима). Због тога овај метод може бити много спорији чак и од brute-force алгоритма. Да би се то избегло имамо идеју за нови алгоритам.

Сви израчунати изрази се могу памтити у једном или више низова, те би се изрази, уместо да се (рекурзивно) састављају из почетка, они само узимали из низа. У низу можемо да памтимо и више вредности од смог израза, као на пример решење израза, па тако нема потребе за понављањем рачунања истих израза, као што је у brute-force алгоритму на рачунање читавог израза.

Главни кораци приликом имплементације овог алгоритма су:

- У први низ убацимо 6 понуђених бројева.
- Затим се састављају сви изрази по 2 броја. Са првог низа се узима један број стављамо га са леве стране затим додајемо други број (не исти као и први) и на крају додајемо једну операцију. Сваки израз који смо добили додајемо у низ број два из уз израз памтимо и бројеве који се налазе у изразу, операције као и вредности зраза.
- Када смо склопили све изразе са по два операнда на реду су они са по три, спајамо први и други низ, на два начина. Један је да са леве стране стављамо бројеве док су са десне изрази, други ће бити обрнути. Сваки нови израз такође памтимо у ноив, трећи низ, и то на исти начин на који смо памтили и изразе са по два операнда.
- Извршавамо алгоритам све док не саставимо све могуће изразе, тј и оне са по 4, 5 и 6 операнада.
- Уколико смо у било ком тренутку пронашли тачно решење, алгоритам престаје са радом и на екран се исписује израз који даје тачно решење, наравно да је то потребно могли смо оставити алгоритам да пролази сваки пут, чува све тачне

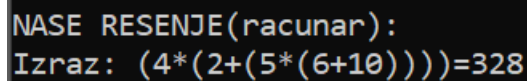
изразе и на крају методе исписује сва могућа решења, али у изради овог пројекта, за тим није било потребе.

Приликом креирања сваког изрази проверава се разлика траженог решења и решења тог изрази, уколико је она најмања до сада, ми памтимо тај израз и његов резултат, како бисмо у случају да не постоји решење, исписали најприближније њему.

```
if (abs(target_number - izraz.get_rezultat()) < min_razlika) {  
    this->izraz.set_izraz(izraz.convert_to_infix());  
    this->izraz.set_rezultat();  
    min_razlika = abs(target_number - izraz.get_rezultat());  
  
    if (min_razlika == 0)  
        return;  
}
```

3.3.2. Метода „ispis_na_konzolu()“

Метода служи за крајњи испис решења који је рачунар успео да пронађе. Уколико рачунар не пронађе тачно решење, исписаће оно које је најприближније траженом.



```
NASE RESENJE(racunar):  
Izraz: (4*(2+(5*(6+10))))=328
```

Слика 6 - Решење које Рачунар проналази након потеза Играча

4. НАЧИНИ ТЕСТИРАЊА

За тестирање користили посебан *main* у коме смо одмах доделили вредности и позивали компјутер да тражи изразе. На овај начин смо прошли кроз већину критичних бројева и уноса и тестирали наш алгоритам. Следе неки примери рада рачунара у тест окружењу.

```
//test 1
racunar.set_brojevi({ 2,4,6,3 });
racunar.set_broj_x(10);
racunar.set_broj_y(25);
racunar.set_trazeni_rezultat(634);

racunar.ispis_na_konzolu();

//test 2
racunar.set_brojevi({ 1,6,5,3 });
racunar.set_broj_x(10);
racunar.set_broj_y(25);
racunar.set_trazeni_rezultat(977);

racunar.ispis_na_konzolu();

//test 3
racunar.set_brojevi({ 2,4,6,3 });
racunar.set_broj_x(10);
racunar.set_broj_y(75);
racunar.set_trazeni_rezultat(743);

racunar.ispis_na_konzolu();
```

Слика 7 - Тестови који су коришћени

Први тест : Рачунар проналази тачно решење и исписује израз и вредност на екран.
Други тест: Рачунар не проналази тачно решење, корисника обавештава о томе и исписује најприближније решење, што је у овом примеру 977.
Трећи тест: Рачунар проналази тачно решење и исписује израз ни вредност.

```
NASE RESENJE(racunar):
Izraz: (4+(6*(3*(10+25))))=634
```

Слика 8 - Пример исписа у тест датотеци

5. ЗАКЉУЧАК

Имали смо три главна задатка, а то су били:

- Рачунање решења израза
- Алгоритам за тражење резултата
- Упис свих података у текстуалну датотеку

Рачунање решења израза смо решили тако што смо израз превели у постфиксну нотацију, када је израз записан на овај начин, нема потребе да водимо рачуна о предности операција, као ни о заградама. Све што је остало на нама јесте да исчитамо стринг са лева на десно и реном примењујемо операције.

Приликом имплементације алгоритама за тражење резултата избегли смо коришћење коришћење рекурзије. Добра страна јесте што смо уштедели доста времена и избегли рачунање резултата истих израза по неколико пута, и то изнова и изнова. Мана нашег алгорита јесте меморија коју смо заузели, али у поређењу са уштедом времена, меморија је у овом пројекту била незнатно мала, па смо занемарили ову чињеницу. Наравно, уколико се пројекат ради и наглашено је да је неопходна уштеда меморије, постојала је алтернатива коју смо могли да користимо и модификујемо алгоритам ради побољшања ове чињенице.

Упис свих података у текстуалну датотеку смо решили тако што смо унапред одредили у ком формату се датотека попуњава, све остало је било прослеђивање нових података, чување предходно уписаних, и спајање са новим подацима. На крају програма додавале су се две додатне линије које су нам пружале информације о крајњим победницима.