Tema: *Interpreter Pythona u C++-u (osnovne aritmetičke operacije, dodela vrednosti promenljivima, uslovi, petlje)*
Autor: *Dimitrijević Katarina 44/2022*
Predmet: *Praktikum iz objektno-orijentisanog programiranja*
Projekat je pisan u programskom jeziku C++ standard 14 , kompajliran pomoću cmake verzija 3.10

Opis projekta:

Funkcionalnost projekta je simuliranje Python interpretera za osnovne aritmetičke operacije (+,-, * , /), dodele vrednosti promenljivima, uslova (if), petlje (for). Podaci(simuliran Python kode) se učitavaju iz eksterne datoteke (.txt) unosom putanje do fajla preko komandne linije. Ispis na komandnu liniju se vrši tek kada se obradi linija sa komandom : *run*.

Projekat se sastoji iz klasa:

- *Variable* - generička klasa koja omogućava čuvanje vrednosti promenljive. Promenljive mogu biti tipa string ili double.
- *Variables* –generička klasa koja omogućava čuvanje više promenljivih. Mapa podataka služiće kao baza podataka u projektu. Kako nema pozivanja mape u istovremeno u različitim metodama, ne može da dođe do neispravnog stanja baze. Postoje dve glave mape: *doubles* i *string*s. Ove mape su externog tipa kako bi bile vidljive svim objketima klasa projekta.
- *PrintStatement* – klasa koja obrađuje komandu tipa print(/...../). Postoje različiti slučajevi:
    1. print(*naziv promenljive*) – pokušava da pronađe promenljivu u *doubles* / *strings* . Ako pod tim imenom postoji promenljiva njena vrednost biće ispisana na komandnu liniju.
    2. print (“ *string* ”) – ispisuje se zadati string na komandnu liniju
    3. print (*broj*) – ispisuje se zadati broj na komandnu liniju
- *AssignmentStatement* – apstraktna klasa koju nasleđuju MathAssignmentStatement i StringAssignmentStatement
- *StringAssignmentStatement* – klasa koja je zadužena da vrši dodelu ako su u pitanju string promenljive. Obezbeđuje i osnovne operacije sa stringovima, sabiranje stringova i množenje stringova sa nekim brojem koji predstavlja broj ponavljanja stringa. Prepoznaje da se radi o dodeli vrednosti kada su u pitanju string promenljive tako sto prepoznaje znak: “ „. Ima mogućnost prepoznavanja imena promenljivih I zamenu sa njihovim vrednostima.
- *MathAssignmentStatement* – klasa koja je zadužena da vrši funkciju dodele ako su u pitanju double / int vrednosti. Kada izračuna izraz (ako je pravilne sintakse) ona tu vrednost ubacuje u bazu koja čuva double promenljive(*doubles*). Radi po principu Shunting Yard Algoritma koji je osmislio Edsger W. Dijkstra. Ima mogućnost prepoznavanja imena promenljivih I zamenu sa njihovim vrednostima
- *BlockStatement* – apstraktna klasa koju nasleđuju IfStatement I ForStatement
- *IfStatement* – klasa koja prepoznaje uslovnu komandu *(if uslov :*), analizira uslov I na osnovu njegove tačnosti odlučuje koji blok koda da odradi. Ako je uslov tačan odradiće *if* blok komandi, u suprotnom proverava da li postoji *else* deo koji treba da odradi. U blokovima mogu da se nađu samo operacije dodele I print operacija. Komanda pripada if petlji ako je uvučena tab-om
- *ForStatement* – klasa koja prepoznaje for komandu (*for i in range(a,b)*:), gde su a I b neki celi brojevi. Omogućava ponovno izvršavanje komandi koje su u tom bloku. Komanda pripada for petlji ako je uvučena tab-om.
- *Parser* – klasa koja određuje koja komanda treba da se izvrši na osnovu njene sintakse.
- Interpreter,*FileReader* –pomoću putanje datoteke omogućavaju čitanje jedne po jedne linije i izvršavanje.

Generička klasa **Variable** omogućava jednostavan pristup čuvanja naziva promenljive (const string *name*) i njene vrednosti(T *value*). Pristup nazivu promenljive je omogućen preko metode *getName*(). Takođe pristup vrednosti promenljive preko metode *T getValue*(). Vrednost promenljive može se kasnije izmeniti pomoću metode *setValue(T t)*.

Izvorni kod : variable.h

```
11 template<typename T>
12 class Variable {
11     private:
12         const string name;
13         T value;
14     public:
15         Variable(T,string);
16         void setValue(T);
17         T getValue() const;
18         string getName() const;
19 };
20
```

Osnovna namena generičke klase **Variables** je da vodi evidenciju o svim promenljivama koje se koriste u programu. Mapa *variables* služi da smesti sve promenljive i onemogućava da postoje promenljive sa istim imenom a različitim vrednostima. Metoda *Variable<T>* findVariable(const string&)* vrši pretragu imena promenljive na osnovu zadatog stringa, a ako ne pronađe vraca nullptr. Metoda *void addVariable(Variable<T>*)* služi za dodavanje promenljive u mapu i njeno evidentiranje, ako promenljiva sa tim imenom već postoji menja se samo njena vrednost. Metoda *void removeVariable(string)* uklanja promenljivu sa mape ako postoji.

Izvorni kod : variables.h

```
21 template<typename T>
22 class Variables {
23     private:
24         map<string, Variable<T>* > variables;
25     public:
26         Variables();
27         ~Variables();
11         Variable<T>* findVariable(const string&) const;
12         void addVariable(Variable<T>*);
13         void removeVariable(string);
14         void iterateVariables() const;
15 };
```

U header datoteci **globals.h** se deklarišu globalne promenljive *doubles* i *strings* tipa Variables<double> i Variables<string>. Deklaracija sa ključnom rečju extern znači da ove promenljive postoje negde u drugom fajlu (globals.cpp), ali ovde se samo najavljuju kako bi bile dostupne i u ostalim fajlovima koji uključe globals.h

Izvorni kod : globals.h

```
6  extern Variables<double> doubles;
7  extern Variables<string> strings;
8  extern std::string output;
```

U **globals.cpp** koji uključuje globals.h, dešava se definicija globalnih promenljivih koje su prethodno bile deklarisane u zaglavlju.Deklaracija u globals.h obaveštava ostatak koda da postoji takva promenljiva, a ovde je ona zapravo kreirana u memoriji.

Izvorni kod : globals.cpp

```
9  Variables<double> doubles;
10 Variables<string> strings;
11 std::string output;
```

Klasa **Interpreter** služi za interakciju sa korisnikom. Kreiranjem instance klase Interpreter, ona se "povezuje" sa kodom koji treba da se interpretira. U konstruktoru se prosleđuje string koji predstavlja putanju do datoteke u kojoj se nalazi Python kod koji treba da se interpretira. Npr. ../examples/pythonCode.txt. Metoda *void run()* pokreće interpreter tako što poziva metodu **FileReader**-a *void processFile(const string&)*. Ova metoda pokušava da otvori datoteku i ako je u tome uspešna poziva metodu klase Parser *void parseLine(string)*.

Izvorni kod : interpreter.h

```
14 class Interpreter
15 {
16     private:
17         class FileReader {
18              friend ostream& operator<<(ostream&,const RunException&);
19               Parser &parser;
20               void processFile(const string);
21              FileReader(Parser &);
22         };
23
24         Parser parser;
25         FileReader fileReader;
26         const string & filepath;
27     public:
28         Interpreter( string &);
29         void run();
30
31 };
```

Klasa **Parser** služi da prepozna koja je vrsta komande. Atributi *bool itIsFor,itIsIf,itIsElse* kao i metode *bool isItFor(const string),bool isItPrint(const string), bool isItIf(const string), bool isItElse(const string),bool isItStringAssign(const string),bool isItMathAssign(const string)* služe kako bi odredile da li se radi o nekoj vrsti komande. Ako Parser naiđe na komandu *run* dešava se izuzetak *RunException* i to okida ispisivanje globalne promenljive *output* na komandnu liniju.Parser ne prepoznaje prazne linije, pa sintaksa glasi da nema praznih linija u kodu, niti komentara.

Izvorni kod : parser.h

```
15 class Parser
16 {
17     private:
18         MathAssignmentStatement mathAssign;
19         StringAssignmentStatement stringAssign;
20         ForStatement myFor;
21         PrintStatement myPrint;
22         IfStatement myIf;
23
24         bool itIsFor = false;
```

```
25          bool itIsIf = false;
26          bool itIsElse = false;
27
28          vector<string> forBody;
29          vector<string> ifBody;
30          vector<string> elseBody;
31
32
33      public:
34          Parser();
35          void parseLine(string);
36          bool isItFor(const string);
37          bool isItPrint(const string);
38          bool isItIf(const string);
39          bool isItElse(const string);
40          bool isItStringAssign(const string);
41          bool isItMathAssign(const string);
42  };
```

Klasa **PrintStatement** je konkretna klasa koja obrađuje linije u kojima se nalazi print naredba. Metoda *void addToOutput(string)* prepoznaje koji deo prosleđenog stringa predstavlja argument funkcije print. Argument može biti broj, koji se prepoznaje pomoću metode *bool isDigit(const string&),* varijabla koja se prepoznaje pomoću metode *bool isVariable(const string&)* ili string. U zavisnosti od vrste argumenta dodaje se određeni string u globalnu promenljivu *output* .

Sintaksa je veoma bitna za pravilno prepoznavanje argumenta print funkcije.Unutar print funkcije može biti ispisan naziv samo jedne promenljive, jedan string ili jedan broj. Posle početne zagrade i pre završne zagrade ne sme da stoji nijedan blank karakter.

Npr. Sintaksa print(a) nije dozvoljena i tada se javlja greška tipa *MyException*.

Izvorni kod : printStat.h

```
13  class PrintStatement
13  {
14      public:
15          PrintStatement();
16          void addToOutput(string);
17          bool isDigit(const string& );
18          bool isVariable(const string&);
19  };
```

Apstraktna klasa **AssignmentStatement** služi da bi objedinila zajedničke osobine koje imaju komande dodele vrednosti promenljivama.Atribut *body* predstavlja celu komandu dodele. Atribut *variableName* predstavlja ime promenljive iz zadatog stringa (pre znaka "="), izdvaja se pomoću metode *void findVariableName()*. Metoda *virtual void addVariableToMap()* je čista virtuelna funkcija čije se telo definiše u klasama MathAssignmentStatement i StringAssignmentStatement koje nasleđuju klasu AssignmentStatement.

Izvorni kod : assignStat.h

```
12  class AssignmentStatement
13  {
14      protected:
15          string body;
16          string variableName;
17      public:
```

```
18          void findVariableName();
19          string getVariableName();
20          void setBody(string);
21          virtual void addVariableToMap() = 0;
22  };
```

Klasa **StringAssignmentStatement** je konkretna klasa koja nasleđuje AssignmentStatement. Obrađuje komandu dodele ako se radi o operacijama sa stringovima. Prepisani operator * omogućava operaciju množenje stringa sa brojem kako bi se dobilo ponavljanje datog stringa. Metoda *string evaluate()* sračunava izraz dok metoda *void addVariableToMap()* smešta promenljivu u globalnu promenljivu *strings*.

Kako bi se prepoznalo da se radi o dodeli i operacijama sa stringovima moraju da se nađu karakteri "". Dozvoljeni operatori su:

- + - kako bi se dva stringa nadovezala
- * - kako bi se string ponovio više puta.Sintaksa je takva da se mora prvo navesti string pa znak *, a zatim željeni broj ponavljanja.

Osim konkretnih stringova ("...") može se navesti i naziv promenljive koja je tipa string i pre toga joj je bila dodeljena vrednost (nalazi se u *strings*).

Ako se sintaksa ne isprati, desiće se greška tipa *MyException*.

Izvorni kod : stringAssign.h

```
12  class StringAssignmentStatement : public AssignmentStatement
13  {
14         public:
15                StringAssignmentStatement();
16                string evaluate();
17                 void addVariableToMap() override;
18  };
19  string operator*(const string&, int);
```

Klasa **MathAssignmentStatement** je konkretna klasa koja nasleđuje AssignmentStatement. Obrađuje komandu dodele ako su svi operandi brojevi ili promenljive iz mape *doubles* pomoću metode *double evaluate()*. Metoda *int precedence(char)* pomaže u tome da se isprate koji operatori imaju prednost. Metod *double applyOperation(double, double, char)* primenjuje operaciju na operande.

Ako je sintaksa pogrešna desiće se izuzetak MyException().

Izvorni kod : mathAssign.h

```
14  class MathAssignmentStatement : public AssignmentStatement
15  {
16      public:
17          MathAssignmentStatement();
18          int precedence(char);
19          double applyOperation(double, double, char );
20          double evaluate();
21          void addVariableToMap() override;
22  };
```

Apstraktna klasa **BlockStatement** sadrži metode i atribute koji su zajednički za IfStatement i ForStatement. Atribut *myPrint* služi kako bi se uspešno izvršila print komanda, *mathAssign* kako bi se izvršile osnove aritmetičke operacije, *stringAssign* kako bi se obradili stringovi. U vectore *ifBody* i *elseBody* se smeštaju linije

koda koja se nalaze u bloku koji pripada određenoj if naredbi. Metode *bool isItPrint(const string line), bool isItStringAssign(const string line), bool isItMathAssign(const string line)* služe da bi odredile tačno koja je vrsta naredbe. Čista virtuelna metoda *void execute(vector<string>)* se definiše u klasama IfStatement i ForStatement koje nasleđuju BlockStatement.

Izvorni kod : blockStat.h

```
23 class BlockStatement
24 {
25     protected:
26         PrintStatement myPrint;
27         MathAssignmentStatement mathAssign;
28         StringAssignmentStatement stringAssign;
29         vector<string> ifBody;
30         vector<string> elseBody;
31     public:
32         bool isItPrint(const string line);
33         bool isItStringAssign(const string line);
34         bool isItMathAssign(const string line);
35         virtual void execute(vector<string>) = 0;
36 };
```

Konkretna klasa **IfStatement** nasleđuje BlockStatement. Obrađuje if komandu. Metodom *void setCondition(string*) se postavlja atribut *condition*, ali se pre toga ekstraktuje iz cele if naredbe pomoću metode *string extractCondition(const string&)*. Metodom *void executeIf()* se pokreće izvršavanje bloka naredbi u zavisnosti od uslova koji se sračunava u metodi *bool evaluateCondition()*. Ako je uslov tačan, izvršava se blok naredbi ispod if-a (nalaze se u *ifBody*), koji se setuje pomoću metode *void setIfBody(vector<string>)*. Ukoliko uslov nije tačan, proverava se da li postoji else deo pomoću atributa *isThereElse*, i ako postoji izvršava se pomoću metode *void execute (vector<string>)*. Atribut *isThereElse* se setuje pomoću metode *void setIsThereElse(bool)*. Blok koda koji pripada else delu (promenljiva *elseBody*) se setuje pomoću metode *void setElseBody(vector<string>)*. Metode *bool applyOperation(bool , bool , const string&), bool applyOperation(double , double , const string&),string applyOperation(string , string , const string&)* pomažu u određivanju tačnosti tvrđenja.

Sintaksa je jako bitna:

- posle "if" mora da se nalazi blank karakter
- if komanda mora da se završava sa :
- else naredba mora da se završava sa :
- kako bi se naznačilo da linija koda pripada if/else bloku uvlači se tab-om (ili 4 razmaka)
- u if upitu ne mogu da se nađu stringovi, mogu true/false , promenljive iz mape doubles i brojevi, ali samo u smislu upoređivanja jer brojevi nemaju logičku vrednost(0-za netačno i obrnuto,ne važi). Operatori koji mogu da se koriste su &&, || , <, > , <=, >=, ==, !=.
- Unutar if/else bloka mogu da se nađu samo operacija dodele(izrazi) i print komanda

Ako se sintaksa ne ispoštuje dešava se izuzetak tipa MyException.

Izvorni kod : ifStat.h

```
15 class IfStatement : public BlockStatement
11 {
12     private:
13         bool conditionMet;
14         string condition;
15         bool isThereElse = false;
```

```
16      public:
17          IfStatement();
18          void setCondition(string);
19          string extractCondition(const string&);
20          int precedence(const string& );
21          bool applyOperation(bool , bool , const string&);
22          bool applyOperation(double , double , const string&);
23          string applyOperation(string , string , const string& );
24          vector<string> tokenize();
25          bool  evaluateCondition();
26          bool isDigit(const string&);
27          void setIfBody(vector<string>);
28          void setElseBody(vector<string>);
29          void setIsThereElse(bool);
30          void executeIf();
31          void execute(vector<string>) override;
32  };
```

Konkretna klasa **ForStatement** je dete klase BlockStatement. Atributi *start* i *end* označavaju koliko će puta blok komandi unutar for-a da se izvrši.Methoda *void execute(vector<string>)* izvršava for petlju tako što poziva metodu *void executeLine(string*).

Sintaksa koja se mora ispoštovati:

- for ima samo jedan oblik: for i in range(a,b): -gde su a i b celi brojevi
- komanda se nalazi u for bloku ako je uvučena tab-om (4 razmaka)
- u for bloku se mogu naći if komanda, komanda dodele(izraz), i print komanda
- if ne sme biti poslednja/jedina komanda u foru

Izvorni kod : forStat.h

```
11  class ForStatement : public BlockStatement
12  {
13      private:
14          string iterator;
15          int start;
16          int end;
17          IfStatement myIf;
18          bool itIsIf = false;
19          bool itIsElse = false;
20      public:
21          ForStatement();
22          void setRange(string);
23          void execute(vector<string>) override;
24          void executeLine(string);
25          bool isItIf (const string &);
26          bool isItElse (const string &);
27          bool isItTabTab(const string &);
28  };
```

Primeri

Primer 1 :

Izvorna datoteka : pythonCode.txt

```
29  if 5 < 4 :
```

```
30      print("prvi if")
31  if 10 == 11 :
32      print("drugi if")
33  else :
34      print("else deo")
35  if  1 == 1 :
36      c="treci if"
37  b = 4-1 * 3
38  print("aritmetika")
39  d = 10
40  e = b + d
41  for i in range(1,5):
42      print(e)
43  print("strings")
44  a1="prvi string"
45  a2=" drugi string"
46  a3=""+a1+a2*2
47  print(a3)
48  run
```

Nakon pojavljivanja linije *enter file path:* od korisnika se očekuje da unese putanju do datoteke.

Command line

```
enter file path:
../examples/pythonCode.txt
>> else deo
>> aritmetika
>> 11.000000
>> 11.000000
>> 11.000000
>> 11.000000
>> strings
>> prvi string drugi string drugi string
```

Primer 2:

Izvorna datoteka : pythonCode1.txt

```
1   for i in range(2,4):
2       if a == 3 :
3           print("a")
4           print(a)
5       else :
6           print("i")
7           print(i)
8       a = i + 1
9   praktikum = "Praktukum iz oop"
10  poop = "POOP"
11  znak = " <3 " * 3
12  predmet = praktikum + " : " + poop + znak * 2
13  print(predmet)
14  a = (10 + 2)/6 + 3
15  print(a)
16  run
```

Na komandoj liniji se pojavljuje rezultat interpretiranja koda.

Command line

```
enter file path:
../examples/pythonCode1.txt
>> i
>> 2.000000
>> a
>> 3.000000
>> Praktukum iz oop : POOP <3  <3  <3 <3  <3  <3
>> 5.000000
```

Primer 3 :

Izvorna datoteka : pythonCode2.txt

```
17 a = 10 * 10
18 b = 30 * 30
19 c = a + b
20 c = "a" + 4
21 run
```

Pojaviće se greška jer se pokušava sabiranje stringa I broja.

Command line

```
command line:
enter file path:
../examples/pythonCode2.txt
>> Unexpected character: 4
```