Dokumentacija projekta
„Klase u mini-C jeziku“
Sintaksna i Semantička analiza
Generisanje koda

Dragan Mirković, SW41-2019
Softversko inženjerstvo i informacione tehnologije
Programski Prevodioci
Novi Sad, jun 2022. godina

1. Korišćeni alati

Alat	Verzija
Flex	2.5.4
Bison	2.4.1
Hipsim	1.2

Tabela 1 Korišćeni alati

2. Evidencija implementiranog dijela

U projektu su odrađeni zadaci podržavanja klasa u mini-C jeziku. Klase se sastoje od atributa, konstruktora i liste funkcija koje sadrže. Odrađene su sintaksna i semantička analiza, takođe i generisanje koda. Projekat je implementiran upotrebom alata navedenih u tabeli, Korišćeni alati.

3. Detalji implementacije

Implementacija rješenja se sastoji od deklarisanja klasa i od njihovog instanciranja i poziva metoda. Deklarisanje klasa se radi na početku C datoteke, gdje se može definisani nula ili više klasa. Definicija započinje ključnom riječju „class“ koju prati naziv klase i zagrade „{ }” za deklarisanje tijela klase. Naredni korak je navođenje atributa klase, ako ih ona ima. Atributi se navode kao i promjenljive u mini-C jeziku. Atribute prati obavezan konstruktor, koji mora imati onoliko argumenata u pozivu koliko ima i atributa. Tijelo konstruktora se sastoji od dodjele vrijednosti argumenata atributima. Posle konstruktora proizvoljno se navodi spisak funkcija klase, one su po formatu iste kao i funkcije mini-C jezika. Što se tiče instanciranja klase ono se radi u sledećem formatu:

„class“ - naziv klase - naziv instance - „=“ – „new“ – naziv klase – „(“ – argumenti, ako ih ima – „)“

Primjer deklarisanja klase i njena upotreba prikazana je Slika 1 .

```
class Dragan {
    int p;
    Dragan(int parametar){
        p = parametar;
    }
    int funkcija1Test(){
        return p;
    }
}

int main() {
    int randomVariable;
    int result;
    randomVariable = 5;
    class Dragan draganInstance = new Dragan(randomVariable);
    result = draganInstance.funkcija1Test();
    return draganInstance.funkcija1Test();
}
```

Slika 1 Deklarisanje klase „Dragan“ i njena upotreba

Argumenti koji se proslijeđuju klasama moraju biti unikatnog naziva na nivou konstruktora i funkcija. Atributi moraju biti unikatnog naziva na nivou klase. Za sve se dodjele i slanje argumenata se vrši semantička provjera po tipu. Zarad dolaska do tačnog rješenja semantičke provjere potrebno je bilo proširiti početnu tabelu simbola sa novim atributom, „atr3“. Koji je takođe po tipu „unsigned“ kao i ostali atributi. Taj atribut se koristi za indentifikaciju kojoj klasi atributi, argumenti ili funkcije pripadaju. Primjer tabele sa popunjenim podacima može se vidjeti na Slika 2.

SYMBOL TABLE

	name	kind	type	atr1	atr2	atr3
0	%0	REG	0	0	0	0
1	%1	REG	0	0	0	0
2	%2	REG	0	0	0	0
3	%3	REG	0	0	0	0
4	%4	REG	0	0	0	0
5	%5	REG	0	0	0	0
6	%6	REG	0	0	0	0
7	%7	REG	0	0	0	0
8	%8	REG	0	0	0	0
9	%9	REG	0	0	0	0
10	%10	REG	0	0	0	0
11	%11	REG	0	0	0	0
12	%12	REG	0	0	0	0
13	%13	REG	0	0	0	0
14	Rectangle	CLAS	0	2	0	0
15	a	ATR	1	0	0	14
16	b	ATR	1	1	0	14
17	parA	PAR	1	1	0	14
18	parB	PAR	1	2	0	14
19	getA	FUN	1	1	1	14
20	getB	FUN	1	0	0	14
21	Circle	CLAS	0	1	0	0
22	r	ATR	1	0	0	21
23	parR	PAR	1	3	0	21
24	getR	FUN	1	0	0	21
25	getRFull	FUN	1	0	0	21

Slika 2 Popunjena tabela simbola

U ovom primjeru su korištene dvije klase „Rectangle“ i „Circle“, detaljna struktura klasa se vidi na Slika 3.

```
class Rectangle{
    int a;
    int b;
    Rectangle(int parA,int parB){
        a = parA;
        b = parB;
    }
    int getA(int par1){
        return a+par1;
    }
    int getB(){
        return b;
    }
}

class Circle{
    int r;
    Circle(int parR){
        r = parR;
    }
    int getR(){
        return r;
    }
    int getRFull(){
        return r+r;
    }
}
```

Slika 3 Struktura klasa „Rectangle“ i „Circle“

Generisanje koda je odrađeno upotrebom globalnih atributa, gdje se atributi klase generišu na globalnom nivou i prilikom poziva konstruktora im se dodjeljuje vrijednost. Konstruktor je implementiran kao funkcija bez povratne vrijednosti i bez lokalnih promjenljivih, a funkcije klase su implementirane kao i obične klase u mini-C jeziku.

Testovi koji su relevantni nalaze se u fasciklama „tests_syntax“, „tests_semantics“ i „tests_codegen“. Oni su namjenjeni, u skladu sa svojim nazivom, za provjeru sintaksne i semantičke analize i provjeru generacije koda.

4. Ideje za nastavak

Problem sa trenutnom implementacijom jeste kod generisanja koda za više klasa u jednom fajlu. Problem predstavlja generisanje atributa druge klase jer se ne nalaze na početku datoteke. Ovaj problem bi se potencijalno mogao riješiti tako da se generisanje koda vrši posle sintaksne i semantičke provjere svih klasa.

5. Literatura

Radovi i stranice korišćenje za izradu rješenja:

- https://www.csee.umbc.edu/courses/undergraduate/313/spring05/burt_katz/lectures/Lect10/structuresInAsm.html
- https://www.cs.uaf.edu/courses/cs301/2014-fall/notes/struct/