**a Compiler for SimplePp**

**Documentation**

Zagkas  Dimosthenis 4359 cs04359

Kalaitsidis Damianos 4370 cs04370

Contents

[**Intro** 2](#_Toc156046072)

[**Introduction to simplepp** 2](#_Toc156046073)

[**Design of the compiler in antlr** 2](#_Toc156046074)

[**Information Extraction from c code** 2](#_Toc156046075)

[**Data structures for storing extracted information** 2](#_Toc156046076)

[**Final code generation** 2](#_Toc156046077)

[**IMPLEMENTATION OF THE COMPILER** 2](#_Toc156046078)

[**TESTING** 2](#_Toc156046079)

[**UNRESOLVED ISSUES AND BUGS** 2](#_Toc156046080)

# **Intro**

Το εξής πρότζεκτ υλοποιεί έναν μεταφραστή από την γλώσσα simplepp (simple-python-python) σε C. Γίνεται χρήση του εργαλείου ANTLR4 (Another Tool for Language Recognition) για την αυτοματοποίηση της γραμματικής ενώ η υλοποίηση του κώδικα γίνεται με java.

# 

# **Introduction to simplepp**

Η simplepp (simple-python-python) είναι μία απλή μορφή της γλώσσας python, σχεδιασμένη συγκεκριμένα για το παρόν πρότζεκτ. Ακολουθούν λεπτομέρειες της γλώσσας καθώς και ιδιαιτερότητές της.

Όντας απλή μορφή της γλώσσας python, κάθε πρόγραμμα γραμμένο σε simplepp θα πρέπει να περνάει από διερμηνέα της Python.

Η γλώσσα ακολουθεί τα αντικειμενοστραφή πρότυπα των κλάσεων και του κυρίως προγράμματος της Python, όπου το κυρίως πρόγραμμα δεν δηλώνεται ούτε λαμβάνεται ως κλάση.

Κάθε κλάση δηλώνεται με την λέξη κλειδί *class* και ακολουθεί το όνομα της κλάσης, ενώ σε ένα πρόγραμμα υπάρχει τουλάχιστον μία κλάση.

Από την simplepp, υποστηρίζεται η λειτουργία της κληρονομικότητας, όπου η κλάση που κληρονομείται, τοποθετείται μέσα σε παρενθέσεις αμέσως μετά την κλάση που κληρονομεί. Όπως και στην Python, οι κλάσεις που κληρονομούνται, τοποθετούνται πριν τις κλάσεις που τις κληρονομούν.

Η ύπαρξη της *\_\_init\_\_* συνάρτησης ισχύει και στην simplepp, με τον ίδιο ακριβώς τρόπο, όπως και στην Python. Είναι υποχρεωτική για κάθε κλάση και σε αυτήν, γίνεται η δήλωση των παραμέτρων της, εάν υπάρχουν.

Οι μέθοδοι δηλώνονται με την λέξη κλειδί *def,* ακολουθεί το όνομα τους και μέσα σε παρένθεση, οι παράμετροί της.

Όπως και στην Python, η λέξη κλειδί *self* χρησιμοποιείται για την δήλωση των αντικειμένων στο οποίο εφαρμόζεται.

Στην simplepp, δεν υπάρχει δήλωση των πεδίων των κλάσεων και των τοπικών μεταβλητών των μεθόδων.

Η simplepp υποστηρίζει μόνο μεταβλητές τύπου integer (*int*).

Οι μέθοδοι μπορούν να δεχθούν ως παράμετρο αντικείμενα.

Η πρόσβαση στα πεδία των αντικειμένου γίνεται με τον συμβατικό τρόπο της Python, δηλαδή αντικείμενο, τελεία, πεδίο (*αντικείμενο.πεδίο).*

Λόγο της απλούστερης μορφής της γλώσσας που υλοποιούμαι, οι εντολές που υποστηρίζονται είναι περιορισμένες, δηλαδή: if-else, while, print και return, καθώς και η εκχώρηση.

Επίσης υποστηρίζονται λογικές συνθήκες και αριθμητικές εκφράσεις, με τη συνήθη προτεραιότητα τελεστών.

Η χρήση μία μεθόδου γίνεται με το όνομα του αντικειμένου που την περιέχει, ακολουθούμενο από μία τελεία, το όνομα της μεθόδου και τις παραμέτρους της σε παρένθεση.

Ακολουθεί εάν απλό παράδειγμα το οποίο η simplepp μεταγλωττίζει.

class Person:

def \_\_init\_\_(self, pid, born) :

self.pid = pid

self.born = born

def getPid(self):

return self.pid

def getBorn(self):

return self.born

def millenium(self):

if self.born < 2000:

return 1

else:

return 2

class Employee(Person):

def \_\_init\_\_(self, pid, born, afm, department):

self.pid = pid

self.born = born

self.afm = afm

self.department = department

def getDepartment(self):

return self.department

def setDepartment(self, department):

self.department = department

def getPid(self):

return self.afm

class StupidPrint:

def \_\_init\_\_(self, employee):

print(employee.pid)

print(employee.born)

print(employee.afm)

print(employee.department)

if \_\_name\_\_ == '\_\_main\_\_':

george = Person(200223, 2002)

john = Person(200055, 2000)

peter = Employee(200122, 2001, 990122, 1)

print(george.getBorn())

print(john.getBorn())

print(peter.getBorn())

print(peter.millenium())

print(peter.getDepartment())

peter.setDepartment(2)

print(peter.getDepartment())

print(george.getPid())

print(peter.getPid())

stupid = StupidPrint(peter)

Το οποίο στην C μεταφράζεται ως εξής:

#include <stdio.h>

typedef struct {

    int pid;

    int born;

} Person;

void Person\_init(Person \*self, int pid, int born) {

    self->pid = pid;

    self->born = born;

}

int Person\_getPid(Person \*self) {

    return self->pid;

}

int Person\_getBorn(Person \*self) {

    return self->born;

}

int Person\_millenium(Person \*self) {

    if (self->born < 2000) {

        return 1;

    }

    return 0;

}

typedef struct {

    Person base;

    int afm;

    int department;

} Employee;

void Employee\_init(Employee \*self, int pid, int born, int afm, int department) {

    Person\_init((Person \*)self, pid, born);

    self->afm = afm;

    self->department = department;

}

int Employee\_getDepartment(Employee \*self) {

    return self->department;

}

void Employee\_setDepartment(Employee \*self, int department) {

    self->department = department;

}

int Employee\_getPid(Employee \*self) {

    return self->afm;

}

typedef struct {

    Employee \*employee;

} StupidPrint;

void StupidPrint\_init(StupidPrint \*self, Employee \*employee) {

    self->employee = employee;

    printf("%d\n", self->employee->base.pid);

    printf("%d\n", self->employee->base.born);

    printf("%d\n", self->employee->afm);

    printf("%d\n", self->employee->department);

}

int main() {

    Person george;

    Person\_init(&george, 200223, 2002);

    Person john;

    Person\_init(&john, 200055, 2000);

    Employee peter;

    Employee\_init(&peter, 200122, 2001, 990122, 1);

    printf("%d\n", Person\_getBorn(&george));

    printf("%d\n", Person\_getBorn(&john));

    printf("%d\n", Person\_getBorn((Person \*)&peter));

    printf("%d\n", Person\_millenium((Person \*)&peter));

    printf("%d\n", Employee\_getDepartment(&peter));

    Employee\_setDepartment(&peter, 2);

    printf("%d\n", Employee\_getDepartment(&peter));

    printf("%d\n", Person\_getPid(&george));

    printf("%d\n", Employee\_getPid(&peter));

    StupidPrint stupid;

    StupidPrint\_init(&stupid, &peter);

    return 0;

}

# **IMPLEMENTATION OF THE COMPILER**

Σε αυτό το κεφάλαιο θα αναλύσουμε την χρήση και την υλοποίηση της γραμματικής της γλώσσας με το εργαλείο ANTLR.

Το ANTLR είναι ένα εργαλείο που προσφέρει αυτοματοποιημένη λεκτική και συντακτική ανάλυση του κώδικα, του οποίου η μόνη απαίτηση είναι η γραμματική της γλώσσας.

Η γραμματική αποτελείται από πολλούς κανόνες συνδεδεμένους μεταξύ τους, ξεκινώντας από τον κανόνα prog, του οποίου η λειτουργία είναι να καλεί τον επόμενο κανόνα.

***prog***

***: classes;***

Ο κανόνας classes χρησιμοποιείται στην αρχή του προγράμματος, καλώντας τον κανόνα class τουλάχιστον μία φορά, όπως απαιτείται και από την γλώσσα, η ύπαρξη δηλαδή τουλάχιστον μία κλάσης στον κώδικα. Το σύμβολο “+” υποχρεώνει τον κανόνα class να λειτουργήσει τουλάχιστον μία φορά. Ο κανόνας classes τελειώνει με κάλεσμα του κανόνα main, στο τέλος του προγράμματος.

***classes***

***:class+ main;***

Ο κανόνας class καλείται για κάθε κλάση του προγράμματος python. Στην πρώτη περίπτωση αναγνωρίζει τις κλάσεις χωρίς κληρονομικότητα. Αποδέχεται την λέξη κλειδί *class* και μετά το όνομα της κλάσης ως *ID* ακολουθούμενο από άνω και κάτω τελεία*.* Μετά βλέπει initFunction για το def \_\_init\_\_ και στην συνέχεια όλες τις υπόλοιπες συναρτήσεις.

Στην δεύτερη περίπτωση, αναγνωρίζει τις κλάσεις με κληρονομικότητα, για αυτό μετά το πρώτο ID, (το όνομα της κλάσης), βλέπει και “(ID)”, για το όνομα της κλάσης που κληρονομείται.

***class***

***: ‘class’ ID ‘:’ initFunction functions***

***| ‘class’ ID ‘(‘ ID ‘)’ ‘:’ initFunction functions***

***;***

Ο κανόνας main καλείται στο τέλος του προγράμματος python, μετά από όλες τις κλάσεις. Η ύπαρξη του είναι απαραίτητη από την simplepp. Αναγνωρίζει την συνθήκη της main κλάσης και συνέχεια βλέπει όλα τα statements μέσα της.

***main:***

***‘if’ ‘\_\_name\_\_’ ‘==’ ‘\’\_\_main\_\_\’’ ‘:’***

***statements;***

***initFunction  
 : ‘def’ ‘\_\_init\_\_’ ‘(‘ formalparlist ‘)’ ‘:’***

***statements***

# **Design of the compiler in antlr**

## **Information Extraction from c code**

// Explanation of information extraction from c code

**Data structures for storing extracted information**// Explanation of the data structures used for storing that information

**Final code generation**// Explanation of the generation of final code

# **TESTING**

// Include the tests used and explanation on which parts each test examines

# **UNRESOLVED ISSUES AND BUGS**