Μεταφραστές ΙΙ Ακαδημαϊκό Έτος 2021-2022

Διδάσκων: Γεώργιος Μανής

Φοιτητής: Αθανάσιος Παπαναστασίου

ΑΜ: 3057

Υλοποίηση μεταφραστή αντικειμενοστραφούς γλώσσας PPP σε διαδικαστική γλώσσα C με τη χρήση του ANTLR4.

Γραμματική (ppp.g4)

Στο αρχείο **ppp.g4** βρίσκεται η γραμματική της γλώσσας **ppp**. Το ANTLR δημιουργεί το συντακτικό αναλυτή και με τη χρήση Actions καλούνται οι βοηθητικές συναρτήσεις.

Αλλαγές στη Γραμματική

Οι αλλαγές που έγιναν από την αρχική γραμματική αφορούν σφάλμα του συντακτικού αναλυτή αλλά και προσθήκη δυνατοτήτων.

Πιο συγκεκριμένα στο κανόνα **constructor_call::388** προστέθηκε το σύμβολο **'$'** ώστε να ξεχωρίζει από το κανόνα **func_call.**

Επίσης στο κανόνα **method_def::176** αντικαταστάθηκε ο κανόνας **(int | -)** από το **return_type** ώστε οι συναρτήσεις να επιστρέφουν και αντικείμενα κλάσεων.

Actions

Τα Actions καλούν τις συναρτήσεις ώστε να λάβουν τις απαραίτητες πληροφορίες από τις δομές δεδομένων. Στη συνέχεια με τη δυνατότητα του ANTLR να επιστρέφει δεδομένα για κάθε κανόνα (**returns [<type> abc]**) οι πληροφορίες κάθε κανόνα μεταφέρονται στο **Listener**.

Δομές Δεδομένων

Για την αποθήκευση των πληροφοριών χρησιμοποιούνται δύο λεξικά και ένας πίνακας της Python. Πιο συγκεκριμένα η λίστα **stack** χρησιμοποιείται ως στοίβα για την αποθήκευση της Κλάσης που αποθηκεύεται κάθε στιγμή. Το λεξικό table αποτελεί τον πίνακα συμβόλων και αποθηκεύει τη πληροφορία στη μορφή **table[class][symbol] = symbolType**. Το λεξικό **inherit** αποθηκεύει τις σχέσεις κληρονομικότητας ανάμεσα στις κλάσεις στη μορφή **inherit[class] = [parent_1, parent_2, ..].**

Η διαχείριση των παραπάνω δομών γίνεται μόνο από τα **Actions** με τη χρήση των παρακάτω συναρτήσεων.

addScope(scope)

Η **addScope()** αποθηκεύει στη στοίβα τη κλάση **scope** και δημιουργεί καινούρια καταχώριση στο πίνακα συμβόλων εάν δεν υπάρχει.

```
def addScope(self, scope):
 self.stack.append(scope)
 if scope not in self.table.keys():
  self.table[scope] = {}
```

findScope(entityName, scopePath)

Η findScope() αναζητεί το σύμβολο **entityName** στο πίνακα και επιστρέφει τη σειρά κλάσεων που απαιτούνται για την προσπέλαση του. Αν δεν υπάρχει στο πίνακα επιστρέφει **None**.

```
def findScope(self, entityName, scopePath:list):
 currentScope = scopePath[-1]
 if entityName in self.table[currentScope].keys():
  return scopePath, self.table[currentScope][entityName]
 for upperScope in self.inherit[currentScope]:
  entity = self.findScope(entityName, scopePath+[upperScope])
  if entity is not None: return entity
```

searchInherit(parameterClass, argumentClassList)

Η **searchInherit()** αναζητεί το τύπο **parameterClass** στο λεξικό **inherit**. Αν υπάρχει επιστρέφει τη σειρά κλάσεων που απαιτούνται για την προσπέλαση του.

```
def searchInherit(self, parameterClass , argumentClassList):
```

```
 if parameterClass in argumentClassList:
   return [parameterClass]
 for argumentClass in argumentClassList:
   upperClassList = self.inherit[argumentClass]
   for upperClass in upperClassList:
     return [upperClass] + self.searchInherit(parameterClass,
self.inherit[upperClass])
 return []
```

checkArguments(parameterType, argumentType, argument)

Η **checkArguments()** δέχεται 3 λίστες. Μία λίστα με τις ορισμένες παραμέτρους της συνάρτησης, τα ορίσματα που δόθηκαν κατά τη κλήση της και οι τύποι τους. Αν εντοπίσει κάποια διαφορά αναζητεί αν κάποια γονική κλάση ικανοποιεί τις συνθήκες με τη χρήση της **searchInherit().** Αλλιώς επιστρέφει κατάλληλα μηνύματα.

```
def checkArguments(self, parameterType, argumentType , argument):
  if len(parameterType) != len(argumentType):
    self.error('', '',self.stack[-1]+' wrong arguments')
  for i in range(0, len(parameterType)):
    if parameterType[i] != argumentType[i]:
      if (self.inherit[argumentType[i]]):
        argumentClassPathList = self.searchInherit(parameterType[i],
[argumentType[i]])
        if argumentClassPathList[-1] == parameterType[i]:
          argument[i] = [argument[i]] + argumentClassPathList
        else:
          self.error('', argumentType[i], parameterType[i]+' was
expected')
      else:
        self.error('', argumentType[i], parameterType[i]+' was
expected')
  return argument
```

getVariable(variableName)

Η **getVariable()** δέχεται ως είσοδο το όνομα μίας μεταβλητής. Αναζητεί την εγγραφή της στο πίνακα συμβόλων. Αν υπάρχει επιστρέφει το μονοπάτι κλάσεων που απαιτείται για την προσπέλαση της και την εγγραφή της.

```
def getVariable(self, variableName):
```

```
  variableEntity=self.findScope(variableName, [self.stack[-1]])
  if not variableEntity:
        return (False, 'not in scope')
  variableClassPathList, variableType = variableEntity
  variableClassPath = ['$'+upperClass for upperClass in
variableClassPathList[2:]]
  variableList=variableClassPath+[variableName]
  if len(variableClassPathList) > 1 and variableClassPathList[1] !=
'main':
        variableList = ['self']+variableList
  return (variableList, variableEntity)
```

getFunction(functionName, parameterType, argumentList, currentScope)

Η **getFunction()** δέχεται ως είσοδο το όνομα μίας συνάρτησης, τους τύπους των ορισμάτων της, τα ορίσματα της και την αρχική κλάσης αναζήτησης. Αναζητά την εγγραφή της κλάσης στο πίνακα συμβόλων και αν υπάρχει, ελέγχει τα ορίσματα της με τη **checkArguments()**. Επιστρέφει το νέο όνομα της συνάρτησης, το μονοπάτι κλάσεων προς το αντικείμενο-όρισμα της μεθόδου και την εγγραφή της μεθόδου στο πίνακα συμβόλων.

```
def getFunction(self, functionName, parameterType:list,
argumentList:list, currentScope:list, d=2):
  functionEntity=self.findScope(functionName, currentScope)
  if not functionEntity:
        return (False,'not in currentScope', None)
  functionClassPathList,functionType = functionEntity
  self.checkArguments(functionType[1:-1], parameterType, argumentList)
  pppFunctionName=functionName+'$'+functionClassPathList[-1]
  functionClassPath = ['$'+upperClass for upperClass in
functionClassPathList[d:]]
  if len(functionClassPathList) > 1 and functionClassPathList[1] !=
'main':
    functionClassPath = ['self'] +functionClassPath
  return (pppFunctionName, [functionClassPath], functionEntity)
```

getDFunction(variableName, functionName, functionType, argument)

Η **getDFunction()** λειτουργεί παρόμοια με τις **getVariable()** και **getFunction()** αφού τις καλεί. Διαχειρίζεται τις περιπτώσεις όπου ένα αντικείμενο καλεί μια μέθοδο του. Αναζητεί το αντικείμενο και τη μέθοδο του στο πίνακα συμβόλων. Αν υπάρχει επιστρέφει το νέο όνομα της συνάρτησης, το μονοπάτι κλάσεων προς το αντικείμενο όρισμα στη μέθοδο του και την εγγραφή της μεθόδου στο πίνακα συμβόλων.

```
def getDFunction(self, variableName, functionName, functionType:list,
argument:list):
  variableList, variableEntity = self.getVariable(variableName)
  variableClassPathList, variableType = variableEntity

  pppFunctionName, functionClassPath, functionEntity =
self.getFunction(functionName, functionType, argument, [variableType],
1)
  functionClassPathList, functionType = functionEntity

  functionClassPath2 = ['$'+upperClass for upperClass in
functionClassPathList[1:]]
  variableClassPath = ['$'+upperClass for upperClass in
variableClassPathList[2:]]

  dFunctionClassPath=[variableClassPath+[variableName]+functionClassPat
h2]+argument

  if len(variableClassPathList) > 1 and variableClassPathList[1] !=
'main':
    dFunctionClassPath[0] = ['self'] +dFunctionClassPath[0]

  return (pppFunctionName, dFunctionClassPath, functionEntity)
```

error(line, token, message) printTable() printInherit()

Οι συναρτήσεις **prinTable()** και **printInherit()** τυπώνουν τα περιεχόμενα των δομών στην οθόνη ενώ η **error()** εμφανίζει κατάλληλα μηνύματα λάθους και τερματίζει όταν βρεθεί λάθος.

```
def error(self, line, token, message):
 print(str(line)+': '+token+' '+message)
 quit()

def printTable(self):
 for pppClass in self.table:
  print('\nScope: '+pppClass)
  for pppSymbol in self.table[pppClass]:
   print(variable,self.table[pppClass][pppSymbol])

def printInherit(self):
 for pppClass in self.inherit:
  print('\nClass: '+pppClass)
  for pppParentClass in self.inherit[pppClass]:
   print(pppParentClass)
```

pppListener (ppp.py)

Ο **Listener** αναλαμβάνει τη παραγωγή του κώδικα **C**. Επίσης κατά την έξοδο του παράγει το τελικό αρχείο και καλεί το μεταφραστή του αν έχει οριστεί.

__init__(cFile)

Στην αρχικοποίηση του ορίζει 7 μεταβλητές και έχει μία παράμετρο. Η παράμετρος cFile είναι το όνομα του τελικού αρχείου.

cCompiler:str

Το μονοπάτι για το εκτελέσιμο ενός μεταφραστή **C.**

mallocBuffer:str

Η κατάλληλη εντολή δέσμευσης μνήμης για το constructor που πρόκειται να ορισθεί.

argumentFlag:bool

Αποθηκεύει αν η συνάρτηση που καλείται δέχεται ορίσματα και ορίζει το πλήθος ‘,’.

exprBuffer:list

Αποθηκεύει τα μέλη ενός expression. Αφού ολοκληρωθεί τυπώνεται στη **cLines**.

condBuffer:list

Αποθηκεύει τα μέλη ενός condition. Αφού ολοκληρωθεί τυπώνεται στη **cLines**.

argumentBuffer:list

Αποθηκεύει τα πλήρη ορίσματα των συναρτήσεων όταν οι παράμετροι είναι γονικές κλάσεις των ορισμάτων.

cLines:list

Στη λίστα cLines αποθηκεύονται οι γραμμές που θα τυπωθούν στο τελικό αρχείο. Αρχικοποιείται με τις βιβλιοθήκες της **C** για λειτουργίες εισόδου/εξόδου.

__del__()

Στην έξοδο του, ο **Listener** δημιουργεί ένα νέο αρχείο με το όνομα **<cFile>.c** και γράφει το περιεχόμενο τις λίστας **cLines.** Αν έχει οριστεί το μονοπάτι κάποιου εκτελέσιμου μεταφραστή **C(gcc)** το καλεί.