ΜΕΤΑΦΡΑΣΤΕΣ

Μέλη: Σπήλιος Σπηλιόπουλος, 4495
Σίμων Γκόνι, 4342

Περιεχόμενα αναφοράς:

- Στην συγκεκριμένη αναφορά αρχικά περιέχονται κάποιες λεπτομέρειες για τον λεκτικό και συντακτικό αναλυτή όπου θα δούμε τον τρόπο λειτουργίας τους.Ύστερα περιγράφεται το σκεπτικό γύρω από τον ενδιάμεσο κώδικα και τον πίνακα συμβόλων και τέλος η παραγωγή του τελικού μας κώδικα.Επίσης θα υπάρχουν στιγμιότυπα από tests που ήταν υπεύθυνα για την κατανόηση της σωστής εκτέλεσης του κώδικα κάθε φάσης, αλλά και ελέγχων που έπρεπε να πραγματοποιηθούν.

Μέθοδος Υλοποίησης:

- Στην παρούσα ενότητα θα παρουσιαστεί μία γενική περιγραφή καθώς και το σκεπτικό υλοποίησης ενός Μεταφραστή (Compiler) για την γλώσσα CutePy. Η γλώσσα στην οποία υλοποιήθηκε ο μεταφραστής είναι η Python.

- Ξεκινώντας, το Project που κληθήκαμε να κάνουμε είναι χωρισμένο σε τρεις φάσεις:

    1. **Λεκτικός / Συντακτικός Αναλυτής**
    2. **Ενδιάμεσος Κώδικας**
    3. **Τελικός Κώδικας**

- Αρχικά, δημιουργήσαμε τον Λεκτικό Αναλυτή [Συνάρτηση: **Lex()**] . Οι αρμοδιότητες του είναι αρκετά συγκεκριμένες. Ο Lex() είναι υπεύθυνος για την ανάγνωση κειμένου κατάληξης **"<name>.cpy"**. Παρέχοντας ένα τέτοιο αρχείο ξεκινάει η συντακτική ανάλυση διαβάζοντας ένα γράμμα κάθε φορά και ξεκινώντας από το πρώτο έως και το τελευταίο (**EOF**). Οι λέξεις δημιουργούνται σύμφωνα με τα τερματικά σύμβολα που έχουν δοθεί από την εκφώνηση της άσκησης. Οι λέξεις αποθηκεύονται σε μία λίστα (**results**) με συγκεκριμένη μορφή (ως λίστες). Η λίστα results περιέχει όλες τις λέξεις (**tokens**) του προγράμματος, όπου αργότερα θα παρέχονται στον Συντακτικό αναλυτή για περαιτέρω ανάλυση.

- Επίσης, για την λεκτική ανάλυση χρησιμοποιείται μία συνάρτηση (**Tag Maker(**)) υπεύθυνη για την απόδοση ετικετών (**Tag / Family**), η οποίες θα μας φανούν χρήσιμες για τον Συντακτικό αναλυτή. Επιπροσθέτως, άλλη μία αρμοδιότητα του Συντακτικού αναλυτή είναι να ελέγχει για τις καταλήξεις του αρχείου, καθώς είναι προσαρμοσμένος να δέχεται μόνο αρχεία με κατάληξη **".cpy"** και επιστρέφει το **Path** του αρχείου.

- Στην συνέχεια,έπρεπε να κατασκευάσουμε τον Συντακτικό Αναλυτή [Συνάρτηση **Syn()**]. Κύρια αρμοδιότητα του συντακτικού αναλυτή είναι η σύνταξη της γραμματικής που υποστηρίζει η γλώσσα μας, καθώς της δίνει την ακριβή περιγραφή. Η περιγραφή αυτή ορίζεται ρητά από την εκφώνηση της άσκησης. Επί της ουσίας, ο Συντακτικός αναλυτής ελέγχει μία ακολουθία από πιθανά ενδεχόμενα. Δηλαδή, ελέγχει αν η σύνταξη του κειμένου που δόθηκε ακολουθεί τους κανόνες και την γραμματική της γλώσσας CutePy ξεκινώντας από τον αρχικό αρχικό κανόνα (**start_rule()**). Με προσπέλαση όλων των λέξεων/token της λίστας results συμπεραίνει αν το υπό εξέταση πρόγραμμα είναι σωστά δομημένο ή όχι. Σε περίπτωση που βρεθεί κάποιο συντακτικό λάθος, ο συντακτικός αναλυτής προσφέρει στον χρήστη μήνυμα σφάλματος με την συγκεκριμένη γραμμή και λέξη του κώδικα όπου εντοπίστηκε.

- Επιπρόσθετα, απαραίτητο κομμάτι για την διεκπεραίωση του μεταφραστή είναι η δημιουργία του ενδιάμεσου κώδικα, ο οποίος λειτουργεί σαν “σκαλοπάτι” ανάμεσα στην αρχική και την τελική γλώσσα. Στόχος είναι η μετατροπή των εντολών του προγράμματος σε μία ενδιάμεση αναπαράσταση από το συντακτικό δέντρο που δημιουργεί ο Συντακτικός αναλυτής. Έτσι, σχηματίζεται μια σειρά από τετράδες (**Quads**) οι οποίες διαφοροποιούνται ανάλογα με την εντολή που διαβάζεται εκείνη την συγκεκριμένη στιγμή. Οι τετράδες αυτές αποτελούνται από έναν τελεστή και τρία τελούμενα καθώς και από την ετικέτα (αριθμός Quad) η οποία είναι ξεχωριστή. Με αυτόν τον τρόπο εμπλουτίζεται ο πίνακας συμβόλων με χρήσιμες πληροφορίες για τα δεδομένα του προγράμματος,όπως ονόματα μεταβλητών και των συναρτήσεων,τον τύπο τους,τη θέσης τους στην μνήμη καθώς και μετάδοση παραμέτρων και κανόνες εμβέλειας. Ο πίνακας συμβόλων (**table**) είναι μια δομή δεδομένων που επεκτείνεται όποτε χρειάζεται δημιουργώντας κύκλους (**scopes**) από συναρτήσεις καθώς και βάθος (**level**) αναλογικά με την εμβέλεια του φωλιάσματος. Μέσα σε κάθε κύκλο δημιουργούνται οντότητες (**entities**) από μεταβλητές, προσωρινές μεταβλητές και παραμέτρους. Ο πίνακας αυτός δημιουργείται με στιγμιότυπα, δηλαδή κατά την ολοκλήρωση μιας συνάρτησης μέσω της συνάρτησης **block()** . Έπειτα, αποθηκεύεται στο αρχείο **“test.symb”** και απελευθερώνει το τελευταίο (**scope**). Οι πληροφορίες αυτές αποθηκεύονται στο αρχείο **“testFile.int**”.

- Συνεχίζοντας, το τελευταίο μέρος του προγράμματος είναι η παραγωγή του τελικού κώδικα. Σε αυτό το σημείο, στόχος είναι να μετατρέψουμε τον ενδιάμεσο κώδικα από τετράδες/quads σε γλώσσα μηχανής (**asm**). Αυτό επιτυγχάνεται με τις συναρτήσεις **gnlvcode(), loadvr(), storevr()** και **produce()**. Η **gnlvcode()** είναι ένα τμήμα κώδικα που αναζητά τη διεύθυνση μιας μεταβλητής ή ενός αντικειμένου στη μνήμη. Αυτή η διεύθυνση χρησιμοποιείται για την προσπέλαση ή την τροποποίηση της τιμής της συγκεκριμένης μεταβλητής ή αντικειμένου. Η συνάρτηση **loadvr()** είναι υπεύθυνη για το φόρτωμα της τιμής μιας μεταβλητής ή ενός προσωρινού αντικειμένου σε έναν εγγραφέα (register) “r”. Αντίστοιχα, η **storevr()** είναι υπεύθυνη για την αποθήκευση της τιμής από έναν εγγραφέα (register) r σε μια μεταβλητή ή προσωρινό αντικείμενο “v”. Τέλος, η συνάρτηση **produce()** είναι υπεύθυνη για την παραγωγή του τελικού κώδικα εξόδου από την λίστα των τετράδων quadList στο αρχείο asmFile. Στην συγκεκριμένη συνάρτηση, γίνεται προσπέλαση σε όλα τα quads που έχουν δημιουργηθεί και προστίθενται ετικέτες όπου είναι απαραίτητο. Επίσης καλούνται οι συναρτήσεις που αναφέρθηκαν προηγουμένως, ανάλογα με το quad που εξετάζεται και μετατρέπονται σε εντολές της γλώσσας **assembly**. Τα αποτελέσματα αποθηκεύονται στο αρχείο **“test.asm”**. Τέλος, αυτό το αρχείο είναι το εκτελέσιμο του αρχικού προγράμματος που δόθηκε στον μεταφραστή, πλέον γραμμένο σε γλώσσα μηχανής.

Λεπτομέρειες

- Σε αυτήν την ενότητα θα θέλαμε να τονίσουμε κάποιες συγκεκριμένες λεπτομέρειες για την υλοποίηση της άσκησης που ίσως φανούν για την επεξήγηση. Το μέρος του κώδικα που αξίζει να σχολιάσουμε περισσότερο είναι ο Λεκτικός Αναλυτής καθώς δεν ακολουθήσαμε την υλοποίηση του αυτόματου που δινόταν στις διαφάνειες.

- Η μέθοδος που ακολουθήσαμε είναι η εξής: Αρχικά ορίσαμε το αλφαβητάρι της γλώσσας CutePy περιέχοντας τα σύμβολα της σε διακεκριμένες λίστες όπως φαίνεται στα στιγμιότυπα παρακάτω. Ο διαχωρισμός των συμβόλων απαιτούσε προσοχή σε συγκεκριμένα σημεία καθώς διαβάζαμε μόνο ένα σύμβολο τη φορά. Σε αντίθεση με τις απαιτήσεις της γλώσσας όπου ένα σύμβολο της CutePy μπορεί να αποτελείται από δύο ξεχωριστά σύμβολα (π.χ. **!= , <=, //, #$** κ.ο.κ.).

- Κάθε χαρακτήρας που διαβάζεται από το αρχείο κειμένου αποθηκεύεται σε ένα αλφαριθμητικό (**word**). Η Python διαχειρίζεται τα strings ως λίστες και αυτό ήταν αρκετά βολικό για την προσπέλαση των γραμμάτων που θέλαμε να εξετάζουμε. Τα γράμματα που μας ενδιαφέρουν περισσότερο είναι κυρίως τα τερματικά σύμβολα, δηλαδή σε ποιό σημείο πρέπει να δημιουργηθεί μία καινούργια λέξη.

```
letters =       [ 'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',
                  'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z' ]

numbers =       [ '0','1','2','3','4','5','6','7','8','9']

addOp   =       ['+','-']

mulOp   =       ['*','/','//']
operators =     {"+":"add","-":"sub","*" :"mul","/":"div"}
relOp     =     {"=" :"beq","!=":"bne",">" :"bgt","<":"blt",">=":"bge","<=":"ble"}

equal =         ['=']

not_eq =        ['!']

smaller =       ['<']

larger =        ['>']

group_symbols = ['(',')','[',']']

hashgroup =     ['{','}']

hashtag =       ['#']

dollar =        ['$']

empty =         ['\n','\t',' ','']

delimiters =    [';',':','.','"',',']

underscore =    ['_']


                                                    ## FOR CHARACTERS  ##

marked =        ['!','=','/','<','>']


                                                    ##    FOR WORDS    ##
reserved_words = ['if','else','not','and','or','main','#declare','while','return','print','def','int','input','__name__','"__main__"']
```

- Με βάση τα παραπάνω ξεχωρίζουμε δύο περιπτώσεις εξαρτώμενες από την κατάσταση της λέξης **word**. Ακολουθεί ψευδοκώδικας για την κατάσταση όπου η λέξη **word είναι κενή:**

    Κατάσταση 1 → Word Empty

    - If word is Empty:

        - If character is not terminal symbol:
            word = character
            { Keep creating the current word }

    { Character is terminal symbol }
        - else:
            { Terminal Symbol }
            { Create Word }
            word = character
            {Append word to results}
            results.append(word)
            { Clear Word }
            word.clear()

- Ακολουθεί ψευδοκώδικας για την κατασταση όπου η λέξη **word ΔΕΝ είναι κενή:**

Κατάσταση 2 → Word Not Empty

- If word is **NOT** Empty:

  - If character is not terminal symbol:

```
{ Cases }
{ Check the last string of the word }
{ If it is terminal symbol }
if word [-1] is terminal symbol:
        { Create the previous word }
        results.append(word)
        { Clear word }
        word.clear()

        { Save new character into word }
        word = characters
else:
        { Character is not terminal symbol }
        { Previous character is not terminal symbol }
        { So keep creating the word }
        word = word + characters
```

{ Character is terminal symbol }

  - else:

```
{ Cases }
{ Check the last string of the word }
{ If it is terminal symbol }
➢ if word [-1] is terminal symbol:

        { Cases }
        { Check if there is a combination of the …}
        { … two symbols for CutePy }
    ★ if there is combination :

        word = word + characters
        results.append(word)
        word.clear()
        { Word [-1]  → terminal symbol }
        { Character → terminal symbol }
```

★ else:

{ Create Previous Word **}
results.append(word)
word.clear()
{ Create current word **}
word = characters
results.append(word)
word.clear()

{** Note: Depending on the cases, word might not be created }
{ For example: if word[-1] == “!” and character == “# “ }
{ previous word will be created (!) but new word (#) will …}
{ ... not be created because “#$” is part of our alphabet }

{ Character is not terminal symbol }

➢ else:

{ Create previous word }
results.append(word)

{ Keep creating the current word, just like Note}
word = characters

- Σε αυτό το σημείο, παρατηρούμε ότι υπάρχει μία εναλλαγή μεταξύ των καταστάσεων της λέξης word. Λέξεις που δημιουργούνται και δεν είναι μέρος της γλώσσας ενημερώνονται από την συνάρτηση Tag_Maker() με token = “not_valid”.

- Σε περίπτωση που συναντήσουμε “not_valid” token το πρόγραμμα σταματά να εκτελείται και επιστρέφει μήνυμα σφάλματος.

- Έτσι, καλύπτουμε όλες τις περιπτώσεις των λέξεων.

- Στο τέλος του κώδικα μας, ενημερώνουμε τα tokens, αφαιρούμε τον κώδικα που περιέχεται σε σχόλια, κάνουμε ελέγχους σχετικούς με τα boundaries και εκτυπώνουμε τα αποτελέσματα γραμμή-γραμμή.
- Η λίστα **results** είναι μια λίστα από λίστες όπου κάθε στοιχείο της είναι αποτέλεσμα του Λεκτικού αναλυτή και είναι της μορφής [word, tag, line]

- Τέλος, για την προσπέλαση των λέξεων στις συναρτήσεις του Συντακτικού Αναλυτή χρησιμοποιείται μια global μεταβλητή (current_index) και ενημερώνεται κάθε φορά από την συνάρτηση check_current_index() αυξάνοντας τον μετρητή για την προσπέλαση όλων των στοιχείων της λίστας results, ελέγχοντας πάντα αν είμαστε εντός των ορίων της λίστας. Τα σημεία που αυτό πραγματοποιείται είναι κάθε φορά που συναντάμε ένα αναμενόμενο στοιχείο της ακολουθίας των κανόνων/γραμματικής που έχει οριστεί.

Παράδειγμα Εκτέλεσης

- Παρακάτω βρίσκεται το στιγμιότυπο από το **test** αρχείο που στείλαμε σε όλες τις φάσεις και επιλέχθηκε κατά κύριο λόγο επειδή περιέχει αρκετές από τις περιπτώσεις που πρέπει να ελεγχθούν.Όπως για παράδειγμα,αν σε κάθε συνάρτησης έχει block που να περικλείεται μέσα σε αγκύλες.

- Επίσης εφαρμόζονται τόσο structured statements όπως η if και η while όσο και simple statements με εντολές ανάθεσης (assignment), εντολές εξόδου (print) και επιστροφή τιμής (return). Ακόμα, ορίζονται νέες συναρτήσεις μέσα στην main οι οποίες καλούνται κατά την εκτέλεση του προγράμματος. Μέσα στο πρόγραμμα υπάρχουν expressions, factors και χρησιμοποιούνται διάφοροι τελεστές.

- Με ένα τέτοιο παράδειγμα, διαπιστώνουμε και τις διάφορες τροποποιήσεις στον πίνακα συμβόλων όπως φαίνεται και στα στιγμιότυπα που δημιουργούνται στο αρχείο **“test.symb”**.

- Τα quads δημιουργούνται στο αρχείο **“testFile.int”** και ύστερα μετατρέπονται σε εντολές assembly στο αρχείο **“test.asm”.**

```
def main_prime():
#{
    #declare i
    def isPrime(x):
    #{
        #declare ibasic
        def divides(x,y):
        #{
            if (y == (y//x) * x):
                return (1);
            else:
                return (0);
        #}
        i = 2;
        while (i<x):
        #{
            if (divides(i,x)==1):
                return (0);
            i = i + 1;
        #}
        return (1);
    #}
    #$ body of main_primes #$
    i = 2;
    while (i<=30):
    #{
    if (isPrime(i)==1):
    print(i);
    i = i + 1;
    #}
#}



if __name__ == "__main__":
main_primes();
```

```
L1:
sw ra,(sp)
L2:
lw t0,-4(sp)
addi t0,t0,-20
lw t1,(t0)
lw t0,-4(sp)
addi t0,t0,-16
lw t2,(t0)
div,t1,t1,t2
sw t1,-12(sp)
L3:
lw t1,-12(sp)
lw t0,-4(sp)
addi t0,t0,-16
lw t2,(t0)
mul,t1,t1,t2
sw t1,-16(sp)
L4:
lw t0,-4(sp)
addi t0,t0,-20
lw t1,(t0)
lw t2,-16(sp)
beq,t1,t2, L6
L5:
b L8
L6:
li t1,1
lw t0,-8(sp)
sw t1,(t0)
L7:
b L9
L8:
li t1,0
lw t0,-8(sp)
```

```
lw t0,-8(sp)
sw t1,(t0)
L9:
li a0, 0
li a7, 93
ecall
L11:
sw ra,(sp)
L12:
li t1,2
sw t1,-12(gp)
L13:
lw t1,-12(gp)
lw t2,-16(sp)
blt,t1,t2, L15
L14:
b L26
L15:
addi fp, sp,0
lw t0,-12(gp)
sw t0,-12(fp)
L16:
lw t0,-16(sp)
sw t0,-16(fp)
L17:
addi t0,sp,-24
sw t0,-8(fp)
L18:
lw t0,-4(sp)
sw t0,-4(fp)
addi sp, sp, 0
jal  L12
addi sp, sp, -32
L19:

lw t1,-24(sp)
li t2,1
beq,t1,t2, L21
L20:
b L23
L21:
li t1,0
lw t0,-8(sp)
sw t1,(t0)
L22:
b L23
L23:
lw t1,-12(gp)
li t2,1
add,t1,t1,t2
sw t1,-28(sp)
L24:
lw t1,-28(sp)
sw t1,-12(gp)
L25:
b L13
L26:
li t1,1
lw t0,-8(sp)
sw t1,(t0)
L27:
li a0, 0
li a7, 93
ecall
L29:
addi sp,sp,28
mv gp,sp
L30:
```

```
L30:
li t1,2
sw t1,-12(gp)
L31:
lw t1,-12(gp)
li t2,30
ble,t1,t2, L33
L32:
b L43
L33:
addi fp, sp,0
lw t0,-12(gp)
sw t0,-12(fp)
L34:
addi t0,sp,-20
sw t0,-8(fp)
L35:
lw t0,-4(sp)
sw t0,-4(fp)
addi sp, sp, 0
jal  L30
addi sp, sp, -28
L36:
lw t1,-20(gp)
li t2,1
beq,t1,t2, L38
L37:
b L40
L38:
lw t1,-12(gp)
li t1, -100
li a7, 1
ecall
L39:
```

```
b L40
L40:
lw t1,-12(gp)
li t2,1
add,t1,t1,t2
sw t1,-24(gp)
L41:
lw t1,-24(gp)
sw t1,-12(gp)
L42:
b L31
L43:
li a0, 0
li a7, 93
ecall
L45:
sw ra,(sp)
L46:
lw t0,-4(sp)
addi t0,t0,-20
lw t1,(t0)
lw t0,-4(sp)
addi t0,t0,-16
lw t2,(t0)
div,t1,t1,t2
sw t1,-12(sp)
L47:
lw t1,-12(sp)
lw t0,-4(sp)
addi t0,t0,-16
lw t2,(t0)
mul,t1,t1,t2
sw t1,-16(sp)
```

```
L48:
lw t0,-4(sp)
addi t0,t0,-20
lw t1,(t0)
lw t2,-16(sp)
beq,t1,t2, L50
L49:
b L52
L50:
li t1,1
lw t0,-8(sp)
sw t1,(t0)
L51:
b L53
L52:
li t1,0
lw t0,-8(sp)
sw t1,(t0)
L53:
li a0, 0
li a7, 93
ecall
L55:
sw ra,(sp)
L56:
li t1,2
sw t1,-12(gp)
L57:
lw t1,-12(gp)
lw t2,-16(sp)
blt,t1,t2, L59
L58:
b L70
```

```
L70:
li t1,1
lw t0,-8(sp)
sw t1,(t0)
L71:
li a0, 0
li a7, 93
ecall
L73:
addi sp,sp,28
mv gp,sp
L74:
li t1,2
sw t1,-12(gp)
L75:
lw t1,-12(gp)
li t2,30
ble,t1,t2, L77
L76:
b L87
L77:
addi fp, sp,0
lw t0,-12(gp)
sw t0,-12(fp)
L78:
addi t0,sp,-20
sw t0,-8(fp)
L79:
lw t0,-4(sp)
sw t0,-4(fp)
addi sp, sp, 0
jal  L74
addi sp, sp, -28
L80:
```

```
addi sp, sp, -32
L80:
lw t1,-20(gp)
li t2,1
beq,t1,t2, L82
L81:
b L84
L82:
lw t1,-12(gp)
li t1,-100
li a7, 1
ecall
L83:
b L84
L84:
lw t1,-12(gp)
li t2,1
add,t1,t1,t2
sw t1,-24(gp)
L85:
lw t1,-24(gp)
sw t1,-12(gp)
L86:
b L75
L87:
li a0, 0
li a7, 93
ecall
```

```
1:  begin_block  divides  null  null
2:  //  y  x  T_1
3:  *  T_1  x  T_2
4:  ==  y  T_2  6
5:  jump  null  null  8
6:  retv  1  null  null
7:  jump  null  null  9
8:  retv  0  null  null
9:  halt  null  null  null
10:  end_block  divides  null  null
11:  begin_block  isPrime  null  null
12:  =  2  null  i
13:  <  i  x  15
14:  jump  null  null  26
15:  par  i  CV  null
16:  par  x  CV  null
17:  par  T_3  RET  null
18:  call  divides  null  null
19:  ==  T_3  1  21
20:  jump  null  null  23
21:  retv  0  null  null
22:  jump  null  null  23
23:  +  i  1  T_4
24:  =  T_4  null  i
25:  jump  null  null  13
26:  retv  1  null  null
27:  halt  null  null  null
28:  end_block  isPrime  null  null
29:  begin_block  main_primes  null  null
30:  =  2  null  i
31:  <=  i  30  33
32:  jump  null  null  43
33:  par  i  CV  null
34:  par  T_5  RET  null
35:  call  isPrime  null  null
36:  ==  T_5  1  38
37:  jump  null  null  40
38:  out  i  null  null
39:  jump  null  null  40
40:  +  i  1  T_6
41:  =  T_6  null  i
42:  jump  null  null  31
43:  halt  null  null  null
44:  end_block  main_primes  null  null
```

```
45:  begin_block  divides  null  null
46:  //  y  x  T_7
47:  *  T_7  x  T_8
48:  ==  y  T_8  50
49:  jump  null  null  52
50:  retv  1  null  null
51:  jump  null  null  53
52:  retv  0  null  null
53:  halt  null  null  null
54:  end_block  divides  null  null
55:  begin_block  isPrime  null  null
56:  =  2  null  i
57:  <  i  x  59
58:  jump  null  null  70
59:  par  i  CV  null
60:  par  x  CV  null
61:  par  T_9  RET  null
62:  call  divides  null  null
63:  ==  T_9  1  65
64:  jump  null  null  67
65:  retv  0  null  null
66:  jump  null  null  67
67:  +  i  1  T_10
68:  =  T_10  null  i
69:  jump  null  null  57
70:  retv  1  null  null
71:  halt  null  null  null
72:  end_block  isPrime  null  null
73:  begin_block  main_prime  null  null
74:  =  2  null  i
75:  <=  i  30  77
76:  jump  null  null  87
77:  par  i  CV  null
78:  par  T_11  RET  null
79:  call  isPrime  null  null
80:  ==  T_11  1  82
81:  jump  null  null  84
82:  out  i  null  null
83:  jump  null  null  84
84:  +  i  1  T_12
85:  =  T_12  null  i
86:  jump  null  null  75
87:  halt  null  null  null
88:  end_block  main_prime  null  null
```

```
SCOPE: 0
  ENTITIES:
  ENTITY{1}:  NAME:i   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:isPrime   TYPE:FUNC   STARTING QUAD:2   FRAME LENGTH:20   RETURN TYPE:Int
    ARGUMENTS:
  ENTITY{3}:  NAME:x   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:16
SCOPE: 1
  ENTITIES:
  ENTITY{1}:  NAME:ibasic  TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:divides   TYPE:FUNC   STARTING QUAD:2   FRAME LENGTH:20   RETURN TYPE:Int
    ARGUMENTS:
  ENTITY{3}:  NAME:x   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:16
  ENTITY{4}:  NAME:y   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:20
SCOPE: 2
  ENTITIES:
  ENTITY{1}:  NAME:T_1   TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:T_2   TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:16
SCOPE: 0
  ENTITIES:
  ENTITY{1}:  NAME:i   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:isPrime   TYPE:FUNC   STARTING QUAD:12  FRAME LENGTH:32   RETURN TYPE:Int
    ARGUMENTS:
  ENTITY{3}:  NAME:x   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:16
SCOPE: 1
  ENTITIES:
  ENTITY{1}:  NAME:ibasic  TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:divides   TYPE:FUNC   STARTING QUAD:12  FRAME LENGTH:32   RETURN TYPE:Int
    ARGUMENTS:
  ENTITY{3}:  NAME:x   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:16
  ENTITY{4}:  NAME:y   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:20
  ENTITY{5}:  NAME:T_3   TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:24
  ENTITY{6}:  NAME:T_4   TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:28
SCOPE: 0
  ENTITIES:
  ENTITY{1}:  NAME:i   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:isPrime   TYPE:FUNC   STARTING QUAD:30  FRAME LENGTH:28   RETURN TYPE:Int
    ARGUMENTS:
  ENTITY{3}:  NAME:x   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:16
  ENTITY{4}:  NAME:T_5   TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:20
  ENTITY{5}:  NAME:T_6   TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:24
```

```
SCOPE: 0
  ENTITIES:
  ENTITY{1}:  NAME:i   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:isPrime   TYPE:FUNC   STARTING QUAD:46  FRAME LENGTH:20   RETURN TYPE:Int
    ARGUMENTS:
  ENTITY{3}:  NAME:x   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:16
SCOPE: 1
  ENTITIES:
  ENTITY{1}:  NAME:ibasic  TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:divides   TYPE:FUNC   STARTING QUAD:46  FRAME LENGTH:20   RETURN TYPE:Int
    ARGUMENTS:
  ENTITY{3}:  NAME:x   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:16
  ENTITY{4}:  NAME:y   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:20
SCOPE: 2
  ENTITIES:
  ENTITY{1}:  NAME:T_7   TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:T_8   TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:16
SCOPE: 0
  ENTITIES:
  ENTITY{1}:  NAME:i   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:isPrime   TYPE:FUNC   STARTING QUAD:56  FRAME LENGTH:32   RETURN TYPE:Int
    ARGUMENTS:
  ENTITY{3}:  NAME:x   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:16
SCOPE: 1
  ENTITIES:
  ENTITY{1}:  NAME:ibasic  TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:divides   TYPE:FUNC   STARTING QUAD:56  FRAME LENGTH:32   RETURN TYPE:Int
    ARGUMENTS:
  ENTITY{3}:  NAME:x   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:16
  ENTITY{4}:  NAME:y   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:20
  ENTITY{5}:  NAME:T_9   TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:24
  ENTITY{6}:  NAME:T_10  TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:28
SCOPE: 0
  ENTITIES:
  ENTITY{1}:  NAME:i   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:12
  ENTITY{2}:  NAME:isPrime   TYPE:FUNC   STARTING QUAD:74  FRAME LENGTH:28   RETURN TYPE:Int
    ARGUMENTS:
  ENTITY{3}:  NAME:x   TYPE:VAR  VARIABLE-TYPE:Int   OFFSET:16
  ENTITY{4}:  NAME:T_11  TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:20
  ENTITY{5}:  NAME:T_12  TYPE:TEMP   Temporary VARIABLE-TYPE:Int   OFFSET:24
```