ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΓΛΩΣΣΕΣ ΚΑΙ ΜΕΤΑΦΡΑΣΤΕΣ – ΗΥ340

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2022
ΔΙΔΑΣΚΩΝ: ΑΝΤΩΝΙΟΣ ΣΑΒΒΙΔΗΣ

ΒΑΣΙΚΗ ΕΡΓΑΣΙΑ
ΦΑΣΗ 2η από 5
Ανάθεση: Πέμπτη 24 Μαρτίου 2022, 16:00 (μεσημέρι)
Παράδοση: Πέμπτη 7 Απριλίου 2022, 24:00 (μεσάνυχτα)

Γενικά

Στη 2η φάση θα κατασκευάσετε έναν απλό Συντακτικό Αναλυτή για τη γλώσσα *alpha* ο οποίος θα κάνει τα εξής:

- Εκτυπώνει στην έξοδο τους γραμματικούς κανόνες όπως τους «ανάγει» κάθε φορά κατά τη διάρκεια της συντακτικής ανάλυσης.
- Διατηρεί κατάλληλο πίνακα συμβόλων – symbol table (π.χ. για μεταβλητές και συναρτήσεις).
- Εκτυπώνει στο τέλος της συντακτικής ανάλυσης τον πίνακα σύμβολων (άρα δεν θα κάνετε "free" κανένα σύμβολο – θα δημιουργήσουμε two-phase compiler), αναγράφοντας:
  - το όνομα του κάθε συμβόλου,
  - τον τύπο του (π.χ. μεταβλητή, τυπικό όρισμα συνάρτησης, συνάρτηση προγραμματιστή, συνάρτηση βιβλιοθήκης),
  - τη γραμμή ορισμού (1ης εμφάνισης) στον κώδικα (0 για συναρτήσεις βιβλιοθήκης)
  - την εμβέλεια του (π.χ. τοπική ή καθολική).

Αντικείμενο εργασίας

Η κατασκευή του συντακτικού αναλυτή θα γίνει χρησιμοποιώντας το εργαλείο YACC (ή Bison) στη γλώσσα C ή C++, με χρήση του Λεξικογραφικού Αναλυτή της 1ης φάσης. Επιπλέον, θα θέσετε και τις βάσεις για την κατασκευή του πίνακα συμβόλων (πληροφορία βοηθητική υπάρχει σε σχετικό φροντιστηριακό υλικό αναρτημένο στην ιστοσελίδα του μαθήματος).

Γραμματική

Η γραμματική της γλώσσας alpha δίνεται παρακάτω (τα σύμβολα ∗ και | έχουν την ερμηνεία που γνωρίζετε ήδη από τις κανονικές εκφράσεις), ενώ ότι δίδεται ανάμεσα σε *[* και *]* σημαίνει ότι είναι προαιρετικό – προσοχή, μην τα συγχέετε με τα tokens **[** και **]**:

program → *stmt*∗

stmt → *expr* **;**
| *ifstmt*
| *whilestmt*
| *forstmt*
| *returnstmt*
| **break ;**
| **continue;**
| *block*
| *funcdef*
| **;**

expr → *assignexpr*
| *expr op expr*
| *term*

op → + | - | * | / | % | > | >= | < | <= | == | != | **and** | **or**

term → (*expr*)
| - *expr*
| **not** *expr*

| *++lvalue*
| *lvalue++*
| *--lvalue*
| *lvalue--*
| *primary*

assginexpr → *lvalue* = *expr*

primary → *lvalue*
| *call*
| *objectdef*
| **(** *funcdef* **)**
| *const*

lvalue → **id**
| **local id**
| **:: id**
| *member*

member → *lvalue* **. id**
| *lvalue* **[** *expr* **]**
| *call* **. id**
| *call* **[** *expr* **]**

call → *call* **(** *elist* **)**
| *lvalue callsuffix*
| **(** *funcdef***) (** *elist* **)**

callsuffix → *normcall*
| *methodcall*

normcall → **(** *elist* **)**
methodcall →**.. id (** *elist* **)** *// equivalent to lvalue***.id(***lvalue***,** *elist***)**

elist → *[expr [***,** *expr] ∗]*

objectdef → **[** *[elist | indexed]* **]**
indexed → *[indexedelem [***,** *indexedelem] ∗]*
indexedelem → **{** *expr* **:** *expr* **}**

block → **{** *[stmt∗]* **}**

funcdef → **function** *[***id***]* **(***idlist***)** *block*

const → **number** | **string** | **nil** | **true** | **false**
idlist →*[***id** *[***,** **id***] ∗]*

ifstmt → **if** (*expr*) *stmt [* **else** *stmt]*
whilestmt → **while** (*expr*) *stmt*

forstmt → **for** (*elist; expr; elist*) *stmt*
returnstmt → **return** *[expr]***;**

Κανόνες προτεραιότητας και προσεταιριστικότητας

Προτεραιότητα (υψηλή προς χαμηλή)

```
()
[]
. ..
not ++  --  -
* / %
+ -
> >= < <=
== !=
and
or
=
```

Προσεταιριστικότητα

Δεν υφίσταται
```
> >= < <= == !=
```

Δεξιά
```
not  ++  --  - =
```

Αριστερή
Όλοι οι υπόλοιποι τελεστές

Συναρτήσεις βιβλιοθήκης

Οι παρακάτω συναρτήσεις θεωρούνται συναρτήσεις βιβλιοθήκης, που σημαίνει ότι αντιμετωπίζονται ως ήδη δηλωμένα καθολικά αναγνωριστικά τύπου «*library function*». Έτσι ξεχωρίζουν από αυτές που είναι «*user-defined function*», δηλ. τις συναρτήσεις που ορίζει ο προγραμματιστής.

- `print`
- `input`
- `objectmemberkeys`
- `objecttotalmembers`
- `objectcopy`
- `totalarguments`
- `argument`
- `typeof`
- `strtonum`
- `sqrt`
- `cos`
- `sin`

Χώροι εμβέλειας

- Οι συναρτήσεις βιβλιοθήκης, συναρτήσεις προγράμματος, οι μεταβλητές και τα τυπικά ορίσματα συναρτήσεων, συνιστούν όλα έναν ενιαίο χώρο ονομάτων. Άρα, στον ίδιο χώρο εμβέλειας συγκρούονται.
- Επιπλέον, δεν επιτρέπεται να οριστεί μεταβλητή η συνάρτηση με όνομα ίδιο κάποιας συνάρτησης βιβλιοθήκης, δηλ. τα library functions δεν μπορούν ποτέ να γίνουν ποτέ «shadowed» από μεταβλητές ή συναρτήσεις χρήστη.
- Δεν επιτρέπεται να οριστεί μεταβλητή με όνομα συνάρτησης της οποίας η εμβέλεια είναι ενεργή
- Επίσης, το όνομα μίας συνάρτησης χρήστη είναι πάντοτε r-value, δηλαδή δεν επιτρέπεται εκχώρηση σε αυτό (άρα είναι ουσιαστικά constant, όχι variable).
- Ως καθολική εμβέλεια (*global*) ορίζεται αυτή εκτός block και συνάρτησης (ως αντιστοιχία αυτό θα ισοδυναμούσε με το χώρο καθολικών δηλώσεων της C).
- Όλες οι μεταβλητές και οι συναρτήσεις προγραμματιστή που ορίζονται σε καθολική εμβέλεια, καθώς και οι συναρτήσεις βιβλιοθήκης, έχουν ***scope 0***, το οποίο και σημαίνει ουσιαστικά καθολική εμβέλεια (ή global scope).
- Η είσοδος σε block ***αυξάνει το scope κατά 1***, ενώ η έξοδος από το block ***το μειώνει αντίστοιχα κατά 1***.
- Η είσοδος σε ορισμό συνάρτησης σηματοδοτείται από την παρένθεση πριν τα τυπικά ορίσματα και ***αυξάνει το scope κατά 1***, ενώ η έξοδος σηματοδοτείται με την έξοδο από το block της συνάρτησης ***μειώνοντας το scope κατά 1***.
    - *<u>προσοχή (ειδική περίπτωση)</u>: το block της συνάρτησης δεν αυξάνει επιπλέον το scope κατά +1 άρα το κεντρικό block της συνάρτησης είναι +1 σε σύγκριση με το scope που περιέχει τη συνάρτηση*
- Όταν μειώνεται το scope, τότε όλες οι μεταβλητές και συναρτήσεις που έχουν οριστεί σε αυτό το scope απενεργοποιούνται, δηλ. δε σβήνονται, απλώς «μαρκάρονται» ως «μη χρησιμοποιήσιμες». Έτσι, όταν βγαίνουμε από ένα block ή ορισμό συνάρτησης, οι τοπικές μεταβλητές δεν είναι συντακτικά ορατές (syntactically visible).
- Σε μία συνάρτηση επιτρέπεται πρόσβαση μόνο σε:
    - τοπικές μεταβλητές της ίδιας της συνάρτησης που είναι σε ορατή εμβέλεια (enclosing block)
    - τυπικά ορίσματα της ίδιας της συνάρτησης
    - καθολικές μεταβλητές (δηλ. αυτές μόνο με scope 0, ούτε καν αυτές που είναι σε καθολικά ορισμένο κώδικα, όμως μέσα σε κάποιο block)
    - οποιεσδήποτε άλλες συναρτήσεις με ενεργή εμβέλεια

  Δηλ., μία συνάρτηση που ορίζεται μέσα σε μία άλλη ποτέ δεν έχει πρόσβαση σε καμία από τις μεταβλητές ή τα τυπικά ορίσματα οποιασδήποτε ιεραρχικά περιέχουσας συνάρτησης. Επίσης, μία συνάρτηση που ορίζεται μέσα σε ένα block, ποτέ δεν θα έχει πρόσβαση στις μεταβλητές του block.

Κανόνες εμβέλειας αναγνωριστικών ονομάτων

- Εάν χρησιμοποιείται ένα **id** μεταβλητής η συνάρτησης χρήστη / βιβλιοθήκης σε κάποιο εκάστοτε χώρο εμβέλειας τότε:
  - εάν έχει ίδιο όνομα με κάποιο ενεργό αναγνωριστικό *A* στην ίδια εμβέλεια τότε το **id** αναφέρεται σε αυτό το *A*
    - αλλά, δεν επιτρέπεται να αλλάζει το είδος χρήσης ενός ονόματος στην ίδια εμβέλεια: μία μεταβλητής δεν μπορεί ορίζεται εκ νέου ως συνάρτηση, ενώ μία συνάρτηση δεν μπορεί να χρησιμοποιείται ως μεταβλητή (είναι constant)
      `x = 10; function x(){}` *// error: var redefined as a functi*on
      `function f(){} f = 10;` *// error: function used as an l-value*
  - αλλιώς εάν έχει ίδιο όνομα με κάποιο αναγνωριστικό *A* σε περιέχουσα εμβέλεια στην οποία υπάρχει νόμιμη πρόσβαση τότε το **id** αναφέρεται σε αυτό το *A*

    - *Υπενθύμιση:* δεν επιτρέπεται η πρόσβαση στις τοπικές μεταβλητές ή τυπικά ορίσματα οποιασδήποτε περιέχουσας συνάρτησης –σε συναρτήσεις η πρόσβαση είναι απολύτως νόμιμη.

```
function f(y) {
      function g(x) { return x*y; }  // error: var f::y not accessible in g
      return g(x);
}
function g() {
      function h() { return g(); }   // ::g is visible in g::h
}
```

  - αλλιώς δημιουργείται νέο αναγνωριστικό *A* βάσει του **id** στην εμβέλεια αυτή και το **id** αναφέρεται σε αυτό το *A*, με τύπο ανάλογο με το είδος της δήλωσης (μεταβλητή ή συνάρτηση)

```
function f () {
      local f = 10;    // f inside f is a new variable now
      function h () {
            return f;// error, f::f (local) not accessible in h
            return ::f(); // ok, we access global ::f
      }
}
```

Ειδικά στην περίπτωση τυπικών ορισμάτων συνάρτησης επειδή έχουμε ένα είδος δήλωσης πάντοτε δημιουργούνται νέα αναγνωριστικά.

- Εάν χρησιμοποιείται το **:: id** σε κάποιο εκάστοτε χώρο εμβέλειας τότε:
  - εάν έχει ίδιο όνομα με κάποιο αναγνωριστικό *A* σε καθολική εμβέλεια (scope 0, εκτός block) τότε το **::id** αναφέρεται στο καθολικό *A*
  - αλλιώς μήνυμα λάθους «*δεν βρέθηκε το καθολικό **id***»

- Εάν χρησιμοποιείται το **local id** σε κάποιο χώρο εμβέλειας (*οποιονδήποτε*) τότε:
  - εάν έχει ίδιο όνομα με κάποια μεταβλητή *A* ή συνάρτηση στην ίδια ακριβώς εμβέλεια τότε το **local id** αναφέρεται στη μεταβλητή ή συνάρτηση *A*,
  - αλλιώς εάν δεν υπάρχει σύγκρουση με όνομα συνάρτησης βιβλιοθήκης (*error*) δημιουργείται νέα μεταβλητή *A* βάσει του **id** στην εμβέλεια αυτή και το **local id** αναφέρεται σε αυτό το *A*.

Για τα είδη των μεταβλητών, έχουμε ουσιαστικά τρεις γενικές κατηγορίες, όπως φαίνεται παρακάτω. Να αποθηκεύετε και αυτή την κατηγορία στον πίνακα συμβόλων, θα δούμε ότι θα μας χρησιμεύσει ιδιαίτερα στην παραγωγή κώδικα.

1. global (scope 0),
2. τυπικά ορίσματα
3. και τοπικές μεταβλητές (scope >= 0 και όχι τυπικά ορίσματα)

Παραδείγματα

`input(x);`	Αυτόματη δήλωση global **x**, χρήση global **input** library function
`print(typeof(x));`	χρήση global **print, typeof** και **x**
`::print(::typeof(::x));`	χρήση global **print, typeof** και **x**
`function g(x,y) {`	**g** global**, x,y** local με scope 1, τυπικά ορίσματα
`  print(x+y);`	**x**, **y** είναι τα arguments (local) της συνάρτησης **g**
`  local print = y;`	**error** δεν επιτρέπεται shadowing of library functions
`  ::print(print);`	και τα δύο αναφέρονται στο print library function
`  function h() {`	νέα συνάρτηση **h** με scope 1
`  return x+y;`	**error** τα **x** και **y** είναι formal της **g**.
`  }`	
`  return h;`	το **h** είναι η αναγνωριστικό τοπικής συνάρτησης με scope 1
`}`	
`add = (function(x,y){return x+y; });`	το **add** είναι global, τα **x,y** είναι local arguments με scope 1.
`{`	
`  local x = ::x;`	το **x** είναι νέο local με scope 1, το **::x** είναι το global x.
`  local f = (function(){return x; });`	το **f** είναι νέο local με scope 1, η πρόσβαση στο **x** είναι το προηγούμενο local με scope 1 (εκτός της **f**, αλλά όχι top global δηλ. **error** reporting).
`}`	
`function f(a,b)`	τα **a,b** είναι arguments με scope 1
`{ local a = local b; }`	τα **local a** και **local b** είναι αναφορές στα τυπικά ορίσματα (αφού είναι στο ίδιο scope)

Ποσοστά συμμετοχής

- 1η φάση 5%
- 2η φάση 10%
- ***3η φάση 40%***
- 4η φάση 5%
- ***5η φάση 40%***