Σημασιολογική Ανάλυση και Αφηρημένο Συνακτικό Δένδρο

Μεταγλωττιστές,

Χειμερινό εξάμηνο 2018-2019

Σημασιολογικός Αναλυτής

- Σημασιολογική Ανάλυση:
  - Η διαδικασία εκτέλεσης σημασιολογικών ελέγχων.
    - έλεγχος τύπου

Code:

```
float a = "example";
```

Σφάλμα σημασιολογικού ελέγχου:

error: incompatible types in initialization

Σημασιολογικός Αναλυτής

- Αποτελεί το πιο απαιτητικό κομμάτι του μεταγλωττιστή.

- Επέκταση του πίνακα συμβόλων
  - Data data; → τα δεδομένα που έχει το ID
  - Type type; → ο τύπος δεδομένων
  - void hashtable_insert(HASHTBL *hashtbl, const char *key, Data data, Type type, int scope, Const isConst, int offset);

- Υλοποίηση συναρτήσεων για σημασιολογικό έλεγχο.

- Συνδέεται άμμεσα με το Αφηρημένο Συντακτικό Δένδρο.

Ανάθεση τύπων και έλεγχος εκφράσεων

```
%union{   int intval;
          Boolean boolval;
          double doubleval;
          char cval;
          char *strval;
          Type typeval;}
%type <typeval> standard_type
```

```
standard_type :T_INTEGER          {$$ = INTEGER;}
          |T_REAL    {$$ = REAL;}
          |T_BOOLEAN {$$ = BOOLEAN;}
          |T_CHAR    {$$ = CHARACTER;}
          |T_STRING {$$ = STRING;}
          ;
```

```
expression : expression T_MULT expression
          {$$ = sem_check_MULT_PLUS_MINUS($1,MULT, $3);}
          |expression T_DIVIDE expression
          {$$ = sem_check_DIVIDE($1, $3);}
          |expression T_DIV expression
          {$$ = sem_check_DIV_MOD($1,DIV, $3);}
          |expression T_MOD expression
          $$ = sem_check_DIV_MOD($1,MOD, $3);}
          |expression T_AND expression
          {$$ = sem_check_AND_OR($1,AND, $3);}
          ;
```

Σημασιολογική έλεγχοι σε while/if

- if_statement :T_IF expression {sem_check_BOOL($2);}
- while_statement :T_WHILE expression {sem_check_BOOL($2);}
- iter_space :expression T_TO expression {sem_check_INT($1); sem_check_INT($3);

- Άρα οι σημασιολογικοί έλεγχοι μπορεί να επιστρέφουν τίμες ή να είναι τύπου void...

- Η επιστρόφή μια τίμης δεν άρκει πολλές φόρες χρειάζεται η δυνατότητα να επιστρέφουν κόμβους μιας δομής (Αφηρημένο συντακτικό δένδρο).

Συναρτήσεις

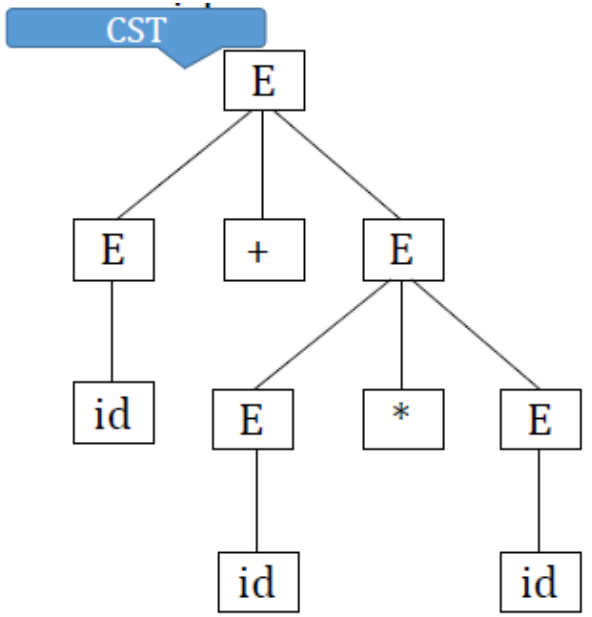
- void sem_check_INT(AST_TS_Union *exp);
- void sem_check_BOOL(AST_TS_Union *exp);
- AST_TS_Union *sem_check_MULT_PLUS_MINUS(AST_TS_Union *exp1, Operation op, AST_TS_Union *exp2);
- AST_TS_Union *sem_check_AND_OR(AST_TS_Union *exp1, Operation op, AST_TS_Union *exp2);
- AST_TS_Union *sem_check_DIV_MOD(AST_TS_Union *exp1, Operation op, AST_TS_Union *exp2);
- AST_TS_Union *sem_check_DIVIDE(AST_TS_Union *exp1, AST_TS_Union *exp2);
- ...

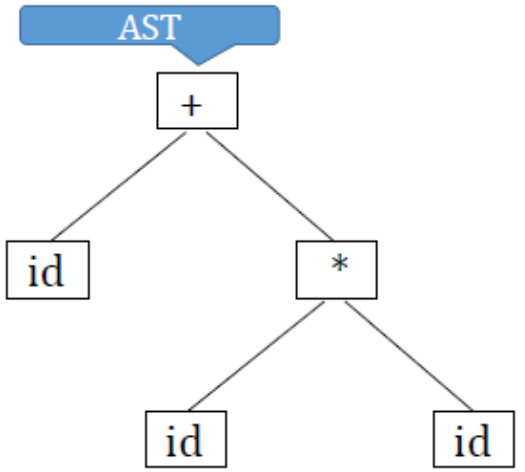
Συντακτικά Δένδρα

- Το συντακτικό δένδρο που μπορεί να κατασκευάσει ο συντακτικός αναλυτής ονομάζεται συμπαγές συντακτικό δένδρο (concrete syntax tree “CST”).
- Αφηρημένο συντακτικό δένδρο (abstract syntax tree “AST”) είναι μία συμπυκνωμένη‘abstract’μορφή του συμπαγούς συντ. Δένδρου όπου δεν υπάρχουν μη-τερματικοί κόμβο

Program: **a + b *c**

Grammar:
E → E * E
|E + E
|id





Κατασκευή αφηρημένου συντακτικού δένδρου

- Τα AST προσφέρουνένα μηχανισμό απαγκίστρωσης των επόμενων σταδίων μεταγλώττισης από την συντακτική ανάλυση.
- Οι εσωτερικοί κόμβοι ενός AST αναπαριστούν τελεστές και οι κόμβοι-απόγονοί τους τους τελεστέους ή τα ορίσματα πάνω στα οποία δρούν.
- Για την κατασκευή ενός AST για αριθμητικές εκφράσεις μπορούν να χρησιμοποιηθούν οι ακόλουθες συναρτήσεις σε συνδυασμό με ένα ορισμό κατευθυνόμενο από την σύνταξη και παραγόμενα χαρακτηριστικά:
  - mknode(op, left, right)
  - mkleaf(id, entry): κόμβος για μεταβλητή τοποθετημένη στη θέση entry του πίνακα συμβόλων.
  - mkleaf(num, val): κόμβος για σταθερά με τιμή val.

Bison: Δημιουργία AST

- Προσθήκη ενεργειών στους συντακτικούς κανόνες.

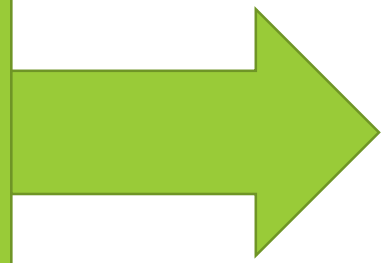
```
decl      : basic_type idents ';'
          ;
Idents    : idents ',' ident
          | ident
          ;
Ident     : ID
          ;
```

```
decl: basic_type idents';' {$$ = function($1, $2);}
```

- Υλοποίηση και εκτέλεση κατά την συντακτική ανάλυση χρησιμοποιώντας και τις σημασιολογικές .

Bison: Δημιουργία AST-Παράδειγμα

```
exp: exp RELATIONAL exp{ $$ = new_ast_relational_node($2, $1, $3); }
| exp EQUALITY exp{ $$ = new_ast_equality_node($2, $1, $3); }
| exp '+' exp{ $$ = new_ast_Exp_node('+', $1, $3); }
| exp '-' exp{ $$ = new_ast_Exp_node('-', $1, $3);}
| exp '*' exp{ $$ = new_ast_Exp_node('*', $1, $3); }
| exp '/' exp{ $$ = new_ast_Exp_node('/', $1, $3); }
| '(' exp')' { $$ = $2; }
| '-' exp %prec UMINUS { $$ = new_ast_Exp_node('M', $2, NULL); }
| NUMBER { $$ = new_ast_number_node($1); }
| NAME { $$ = new_ast_symbol_reference_node($1); }
| NAME '=' exp{ $$ = new_ast_assignment_node($1, $3); }
| NAME '(' ')' { $$ = new_ast_function_node($1, NULL); }
| NAME '(' exp_list')' { $$ = new_ast_function_node($1, $3); }
;
```



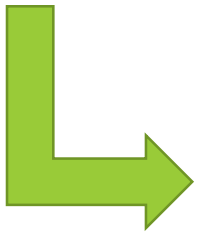
- Ορίζουμε για το AST μια δομή (struct) για τους κόμβος
  - μια κοινή δομή που καλύπτει όλες τις ανάγκες του AST
- Δημιουργούμε συνδέσεις «με pointers»

- Struct:
  - Θα πρέπει να μπειο κωδικός του τελεστή για τους κόμβους έκφρασης
  - Θα πρέπει να μπει ο κωδικός της εντολής για τους κόμβους εντολής
  - Κάποιο πεδίο για τον τύπο του κόμβο

Bison: Δημιουργία AST-Παράδειγμα

- **exp '+' exp{ $$ = new_ast_Exp_node('+', $1, $3); }**

```
struct ast_node*
new_ast_Exp_node(intnode_type, structast_node* left, structast_node* right)
{
struct ast_Exp_node* ast_node= malloc(sizeof(ast_node));
ast_node->node_type= node_type;
ast_node->left= left;
ast_node->right= right;
return ast_node;
}
```



```
struct ast_node// for binary/unary operators and expression lists
{
int node_type;
struct ast_node* left;
struct ast_node* right;
};
```

Bison: Δημιουργία AST-Παράδειγμα

- **$$ = new_ast_symbol_reference_node($1);**

```
struct ast_node*
new_ast_symbol_reference_node(struct symbol_node* symbol)
{
struct ast_symbol_reference_node* ast_node= malloc(sizeof(ast_symbol_reference_node));
ast_node->node_type= 'S';
ast_node->symbol= symbol;
return(structast_node*) ast_node;
}
```

```
struct ast_symbol_reference_node// for symbol references
{
int node_type;
struct symbol_node* symbol;
};
```

Άλλες δομές κόμβων AST-Παράδειγμα

```
struct ast_relational_node// for relational operators
{
int node_type;
enum relational_operator operator;
struct ast_node* left;
struct ast_node* right;
};
struct ast_equality_node// for equality operators
{
int node_type;
enum equality_operator operator;
struct ast_node* left;
struct ast_node* right;
};
struct ast_function_node// for function calls
{
int node_type;
struct ast_node* arguments;
struct symbol_node* symbol;
};
```

Χρήση ενός struct

```
enum relational_operator{
LESS,LESS_OR_EQUAL,GREATER, GREATER_OR_EQUAL};

enum equality_operator{ EQUAL, NOT_EQUAL};
```

```
struct ast_node*
new_ast_relational_node(enum relational_operator
operator,
struct ast_node* left,
struct ast_node* right)
{
struct ast_relational_node*
ast_node=malloc(sizeof(struct ast_relational_node));
ast_node->node_type= 'R';
ast_node->operator= operator;
ast_node->left= left;
ast_node->right= right;
return(structast_node*) ast_node;
}
```

Instruction Nodes (statement: if)

```
selection_statement: IF '(' expression')' statement %prec NO_ELSE
{$$ = new_ast_if_node($3, $5, NULL); }
| IF '(' expression')' statement ELSE statement
{ $$ = new_ast_if_node($3, $5, $7); }
;
```

```
struct ast_if_node// for "if/else" statements
{
int node_type;
struct ast_node* condition;
struct ast_node* if_branch;
struct ast_node* else_branch;
};
```

```
struct ast_node*
new_ast_if_node(struct ast_node* condition,
struct ast_node* if_branch, struct ast_node*
else_branch)
{
Struct ast_if_node* ast_node=
malloc(sizeof(structast_if_node));
ast_node->node_type= 'IF';
ast_node->condition = condition;
ast_node->if_branch= if_branch;
ast_node->else_branch= else_branch;
return (struct ast_node*) ast_node;
}
```

Instruction Nodes (statement: while)

```
iteration_statement
: WHILE '(' expression')' statement{ $$ = new_ast_while_node($3, $5); }
;
```

```
struct ast_while_node// for "while" statements
{
int node_type;
struct ast_node* condition;
struct ast_node* while_branch;
};
```

```
struct ast_node*
new_ast_while_node(struct ast_node* condition,
struct ast_node* while_branch)
{
struct ast_while_node* ast_node=
malloc(sizeof(struct ast_while_node));
ast_node->node_type= 'W';
ast_node->condition= condition;
ast_node->while_branch= while_branch;
return(struct ast_node*) ast_node;
}
```

Συναρτήσεις

- AST_node *new_ast_node(Operation op, Type type, struct ast_node * left, struct ast_node * right);
- AST_node *new_ast_if_node(struct ast_node *condition, struct ast_node *if_branch, struct ast_node *else_branch);
- AST_node *new_ast_while_node(struct ast_node *condition, struct ast_node *while_branch);
- AST_node *new_ast_for_node(Increment inc, struct ast_node *init_condition, struct ast_node *end_condition, struct ast_node *for_branch);
- AST_node *new_ast_leaf_node(Type_Struct *ts_value, char *name);
- ...

Structs

- typedef struct ast_node{ Operation op; Type type; struct ast_node *left; struct ast_node *right;} AST_node;
- struct ast_if_node{ Operation op; struct ast_node *condition; struct ast_node *if_branch; struct ast_node *else_branch;};
- struct ast_while_node{ Operation op; struct ast_node *condition; struct ast_node *while_branch;};
- struct ast_for_node{Operation op; Increment inc; struct ast_node *init_condition; struct ast_node *end_condition; struct ast_node *for_branch;};
- struct ast_leaf_node{ Operation op; char *name; Type_Struct *ts_value;};
- …

Για την εργασία

- Να αποτιμά τα κατηγορήματα των συμβόλων της γραμματικής.
- Να προσθέτει στον ΠΣ πληροφορίες για τα αναγνωριστικά που έχουν εισαχθεί από το ΛΑ ή το ΣΑ:
  - Να κωδικοποιούνται στον ΠΣ οι τύποι και τα συνώνυμα τύπων.
  - Να προσδιορίζονται οι τύποι των μεταβλητών και των συναρτήσεων κατά τη δήλωσή τους.
  - Να αποδίδονται τιμές σε ονόματα σταθερών ή πιθανές αρχικές τιμές σε ονόματα μεταβλητών.
  - Να προσδιορίζονται ο αριθμός, οι τύποι, ο τρόπος περάσματος και τα ονόματα των παραμέτρων των υποπρογραμμάτων που δηλώνονται στο πρόγραμμα

Για την εργασία (ενδιάμεσος κώδικας)

- Να κατασκευάζει το ΑΣΔ
  - Για τις εκφράσεις συνιστάται η χρήση τύπου δέντρου με μεταβλητό αριθμό παιδιών – ανάλογα με τον τελεστή κάθε κόμβου
  - Η δομή των κόμβων του ΑΣΔ πρέπει να είναι τέτοια, ώστε να περικλείει όλη την πληροφορία που παράγεται από τη σημασιολογική ανάλυση των ίδιων και είναι απαραίτητη για τη σημασιολογική ανάλυση των γειτονικών κόμβων.
  - Το δέντρο πρέπει να είναι όσο γίνεται πιο αφηρημένο, ώστε να είναι μικρό και να περιλαμβάνει μόνο τους αναγκαίους κόμβους για τη μετέπειτα ανάλυση του προγράμματος.

Για την εργασία (ενδιάμεσος κώδικας)

- Να δεσμεύει θέσεις στο χώρο δεδομένων (ΧΔ) για τις μεταβλητές του προγράμματος, δημιουργώντας πίνακες δέσμευσης για κάθε εμβέλεια που μεταφράζεται.
  - συμπληρώστε στον ΠΣ τα στοιχεία που θεωρείτε απαραίτητα

ΕΡΩΤΗΣΕΙΣ/ΑΠΟΡΙΕΣ