CS536: Homework 7

Keith Funkhouser

November 17th, 2015

1

Assume that the following productions have been added to the grammar for the YES language:

`decl` → `typedef`
`typedef` → `TYPEDEF type ID SEMICOLON`

What other productions need to be changed and/or added to the YES grammar to allow typedefs?

`type` → `STRUCT id`
`type` → `id`

2

a. What information should be stored with each name in the symbol table?

The symbol table should store at minimum the following information:

(a) Variables - name, type

(b) Function declarations - name, return type, parameter types

(c) Struct definitions - name, list of fields, size

(d) **Typedefs** - name, primitive type, and (if struct) struct name

b. What should be done to process a typedef: `typedef T xxx;`?

(a) If T is a primitive type, continue

(b) Else if T is an ID, check that the ID is in the sym table as a typedef (if ID is not in the sym table or not a typedef, generate an error)

(c) Else if T is STRUCT ID, check that the ID is in the sym table as a struct (if ID is not in the sym table or not a struct, generate an error)

(d) Check that xxx has not already been declared in the sym table local scope (if it has, generate an error)

c. What should be done to process a declaration of a variable, function, or parameter named `xxx` and declared to be of type `T`?

(a) If T is a primitive type, continue

(b) Else if T is an ID, check that the ID is in the sym table as a typedef (if ID is not in the sym table or not a typedef, generate an error)

(c) Else if T is STRUCT ID, check that the ID is in the sym table as a struct (if ID is not in the sym table or not a struct, generate an error)

(d) Check that xxx has not already been declared in the sym table local scope (if it has, generate an error)

d. What should be done to process the use of a name `xxx` in a statement?
The name analysis steps for YES given in the Lecture 15 notes include:

(a) If within an arithmetic operator, generate an error if not of type int

(b) Else if within an equality expression or assignment expression, must be the same type as the other side, and cannot be a function, struct name, or struct variable (otherwise generate an error)

(c) Else if within another relational expression, must be an int (otherwise generate an error)

(d) Else if within a logical expression, must be of type boolean (otherwise generate an error)

(e) Else if within cin/cout expression, cannot be a function, struct name, or struct variable (otherwise generate an error)

(f) Else if within if/while loop, condition must evaluate to boolean (otherwise generate an error)

(g) Else if in a function call, name must be of type function, # and type of args must be correct (otherwise generate an error)

(h) Else if within a return statement, must not return a value if function has a return type of void, and must return the correct type of value if another return type (otherwise generate an error)

Illustrate your answer by showing the entries that would be in the symbol table after processing the following declarations:

```
struct MonthDayYear {
    int month;
    int day;
    int year;
};
typedef struct MonthDayYear date;
date today;
typedef int dollars;
dollars salary;
typedef dollars moreDollars;
moreDollars md;
int d;
```

At a high level, the symbol table might look something like the mapping below:

```
MonthDayYear:   {type: structDef, size: 3*int,
                 fields:
                    {   month: int,
                        day: int,
                        year: int
                    }
                }
date:           {type: typedef, primitive: struct (MonthDayYear)}
today:          {type: date}
dollars:        {type: typedef, primitive: int}
salary:         {type: dollars}
moreDollars:    {type: typedef, primitive: int}
md:             {type: moreDollars}
d:              {type: int}
```