Oct 18, 2004

Comp Sci 431  Program 4: Typechecking **zinc**  (100 pts)

(suppose you are howeg89)  cp p4.tz /home/DROPBOXES/431/howeg89/

**Due date:** software electronically submitted before **11:45pm Saturday November 6**

**Final due date:** software electronically submitted before **11:45pm Thursday Nov 11**

Listings must be in my office before 3:30pm the next class day

You will continue to work in teams for this assignment.

**Your assignment is to (1) find all environment errors, (2) find some static semantic errors, (3) find problems with main and (4) determine offset values for variables, arrays and parameters, arraySize for arrays and the number of parameters, number of local variable words and frame size of functions.**

**Skeleton Files**

Skeleton files can be found in directories /home/perrie/431/public/Projects/p4/ and .../p4/V1/; you will use the files that you created for programs p1, p2 and p3: symboltable.cc, tokens.lex, grammar.yacc, and unparse.cc. I have placed my solution to this project, “z4”, in the p4/V1/ directory.

Your assignment is to modify or add code to the files

a) main.cc (Written by)
b) namecheck.cc
c) typecheck.cc
d) symlist.cc
e) unparse.cc (from project p3)

You need to have a list of pointers to symbol tables; the code belongs in symlist.cc (item d) above). I have placed the file symlist.obj in the p4/ directory; it can be used until you have implemented symlist.cc and compiled it.

A goal of the typechecker is to prevent cascading error messages. For example, the two consecutive assignment statements

\[ x = y; \]  \[ x = y; \]

should generate only one error message in the case that \( y \) is an undeclared variable. In order to achieve this objective, do the following. When an undeclared variable is found, put it in the symbol table and mark it as an error.

**Listings to hand in**

The listings to hand in can be created by doing a “make listings”. You must hand in listings (hardcopy) from the laser printer in the Linux Lab. Create test files that discover environment and semantic errors; name these files b1.z, b2.z, etc. (bad). Create one test file that has no syntax errors to demonstrate that the unparsing correctly displays offset values and other information; name this file g.z (good). The file “g.un” will contain the unparsing of the program in “g.z”.

**Other**

Email me an individual report and tell me (a) whether the program works perfectly or what the known problems are, (b) what your contribution to the program was and (c) anything else that you want me to know about this project. I must receive this email within 48 hours after p4.tz was submitted.

Do a “make erase” before tarring and be certain you submit all software that will allow me to “make” your executable.

Points will be deducted if the Unix utility function “diff” detects differences between your error messages and the error messages from my executable program “z4”. A bounty will be paid for errors found in my executable.

**Caution**

Although the language **zinc** is superficially similar to C in many ways, **zinc** is not C. (p.s. zinc melts at 419.4°C)
Environment Errors
An environment error is also known as a declaration/use error. This error can be a multiply declared name (variable, array, function, parameter) or the use of an undeclared name (variable, array, function).

Environment Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool x[4]; int x;</td>
<td>x is already declared</td>
</tr>
<tr>
<td>int y; bool y[6];</td>
<td>y is already declared</td>
</tr>
<tr>
<td>int f; bool f(){return 0;}</td>
<td>f is already declared</td>
</tr>
<tr>
<td>void g(){ int g(){return 0;}}</td>
<td>g is already declared</td>
</tr>
<tr>
<td>void h(int x, bool x[]){}</td>
<td>x is already declared</td>
</tr>
<tr>
<td>y = n; y = n[5]; y = f*n(y[0],x); n();</td>
<td>n is undeclared</td>
</tr>
</tbody>
</table>

Note: “void a( int a )” is valid because “(“ starts a new scope.

Static Semantic Errors
There are lots of static semantic errors to look for in a program. In this assignment you will check for only the following errors:
• verify that operands of operators in expressions have the correct data type
• verify that assignment statements are valid
• verify that print items are not void type

main Errors
Check for problems with main:
• verify that the program has a function named “main” which returns an int and does not have any parameters.

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>void main()</td>
<td>main does not return int</td>
</tr>
<tr>
<td>bool main()</td>
<td>main does not return int</td>
</tr>
<tr>
<td>int main(int argc, bool b)</td>
<td>main has a parameter</td>
</tr>
</tbody>
</table>

Your error messages must be exactly as shown above. Error Messages will be created by a call to the Error function in message.cc; the line and column number of the offending syntax is to be printed. When “main” is missing, use line number 0 and column number 0, and display the message “function main is missing”.

Unparser
When a program has no errors, the unparser will use the symbol pointer stored in a node to display an offset count for variables, arrays, and parameters. For functions, it will display the number of parameters, number of memory words for local variables and frame size in bytes. For the print statement, it will display the data type of each argument. For example,

```c
void f1(int y[], bool b, bool c[]) {bool a2[5]; int a3; bool a4[2]; int a1; print(3,true,"\n");
    int main(){}
}
```

would have the unparsing

```
...unparse

functionName<parameters:locals:frameSize>
variableName<offset>
1: void f1 <3:9:56> (  
1:    int y <0> [],  
1:    bool b <4> ,  
1:    bool c <8> [])  
1: {  
1:    bool a2 <36> [5];  
1:    int a3 <40> ;  
1:    bool a4 <48> [2];  
1:    int a1 <52> ;  
1:    print(3 <IntType> , true <BoolType> , "\n" <StringType> );  
1: }  
2: int main <0:0:8> ()  
2: {  
2: }
```
Static Semantic Errors

The following zinc static semantic errors exist but your assignment is not responsible for finding these errors.

ERROR line 11 column 16: left operand of * is a function name
ERROR line 11 column 26: function h: #arguments == 3; #parameters == 0
ERROR line 12 column 8: right operand of * is an array name
ERROR line 16 column 6: left operand of * is a function name
ERROR line 18 column 5: function F: argument does not match parameter

Other Error Messages

right operand of && is not bool
left operand of || is not bool
right operand of < is not int
left operand of != is not int