Practical 8

Name: Tushar Harsora

Roll No: 19BCE509

# Introduction

The aim of this practical is to parse the expression and control statement in the given source code in addition ***implement type checking***. This practical consists of 2 files *practical.l* and *practical.yy* the first file declares tokens that will be generated during lexing process. It declares the identifiers like types of braces and parentheses including some keywords, datatypes etc. second file is .yy file which is given to bison the bison converts the context free grammar to deterministic LR and generates the parser. It contains the precedence, start symbol, tokens and grammar rules. It also includes the symbol table to check if variable is already declared or not. And also check if the types are compatible to compare or not which is essentially type checking. It creates symbol table and on every declaration it stores the type information and on relational operation and assignment it checks for the type if it mismatches then error is generated.

# Code

Practical.l

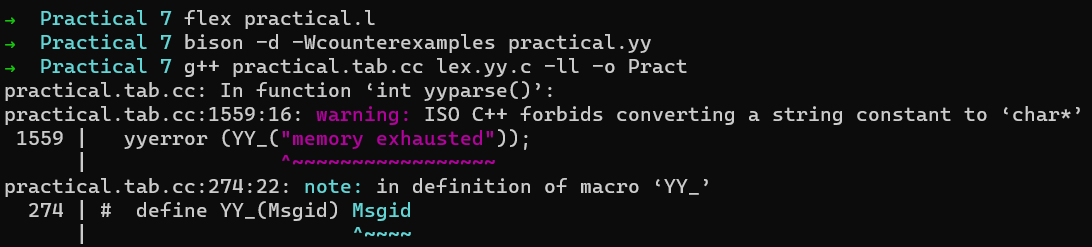
|  |
| --- |
| %{      #include "practical.tab.hh"      #include <stdlib.h>  %}  %option yylineno  %%  "<"|">"|"<="|">="|"||"|"&&"|"=="|"!="     { return REL; }  "int"|"float"         { yylval.name = strdup(yytext); return DATATYPE; }  "if"                    { return IF; }  "else"                  { return ELSE; }  "="                     { return ASSIGN; }  "{"                     { return LBRACES; }  "}"                     { return RBRACES; }  "("                     { return LPAREN; }  ")"                     { return RPAREN; }  [0-9]+.[0-9]+           { yylval.fvalue = atof(yytext); return FCONST; }  [0-9]+                  { yylval.ivalue = atoi(yytext); return ICONST; }  [a-zA-Z][a-zA-Z0-9]\*    { yylval.name = strdup(yytext); return ID; }  "+"|"-"|"/"|"\*"         { return SIGN; }  ";"                     { return SEPERATOR; }  [ \n\t]+                {  }  .                       { return yytext[0]; }  %% |

Practical.yy

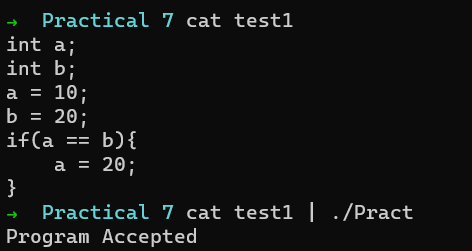
|  |
| --- |
| %{      #include "practical.tab.hh"      // #include <stdio.h>      #include <cstring>      #include <map>      #include <string>      using namespace std;      extern int yylineno;      extern char\* yytext;      int yylex();      void yyerror(char\*);      void yyerror(const string&);      void assert\_is\_compatible(const char\* lhs, const char\* rhs);      map<string, pair<string, string>> symb\_tab;     // maps from variable name to                                                      // variable datatype and value  %}  %union{      char\* name;      int ivalue;      float fvalue;  }  %define parse.lac full  %define parse.error detailed  %define parse.trace  %right '='  %left '+''-'  %left '\*''/'  %left IF\_ELSE\_PREC  %token SEPERATOR REL SIGN IF ELSE ASSIGN LPAREN RPAREN LBRACES RBRACES  %token <ivalue> ICONST  %token <fvalue> FCONST  %token <name> ID DATATYPE  %type Start Decl BooleanThing IfStat ElseStat EXPR MATH  %%  Start : /\* EMPTY \*/          | EXPR Start          | IfStat Start  Decl : DATATYPE ID SEPERATOR      { symb\_tab[$2] = make\_pair($1, ""); }  BooleanThing : ID REL ID                  {                      assert\_is\_compatible($1, $3);                  }  IfStat : IF LPAREN BooleanThing RPAREN LBRACES EXPR RBRACES ElseStat  ElseStat : /\* EMPTY \*/ %prec IF\_ELSE\_PREC              | ELSE LBRACES EXPR RBRACES  EXPR :  Decl          | ID ASSIGN MATH SEPERATOR              {                  /\* Should check for datatype \*/              }  MATH :    FCONST          | ICONST          | ICONST SIGN ICONST          | FCONST SIGN FCONST          | ID SIGN ID              {                  assert\_is\_compatible($1, $3);              }  %%  void assert\_is\_compatible(const char\* lhs, const char\* rhs){      auto f = symb\_tab.find(lhs);      auto s = symb\_tab.find(rhs);      if(f == symb\_tab.end()){          yyerror((string("Identifier ") + lhs + string(" Not Declared")));          exit(-1);      }      else if(s == symb\_tab.end()){          yyerror((string("Identifier ") + rhs + string(" Not Declared")));          exit(-1);      }      if(f->second.first != s->second.first){          yyerror((string("Incompatible Datatypes ") + string(f->second.first) + " " + string(s->second.first)));          exit(-1);      }  }  void yyerror(const string& s){      char\* temp = (char\*) malloc(sizeof(char) \* s.size());      strcpy(temp, s.c\_str());      yyerror(temp);      free(temp);         // i don't think it reaches here      // exit(-1);  }  void yyerror(char\* s) {      printf("ERROR: %s at line %d\n", s, yylineno, yytext);  }  int main(){      int failure = yyparse();      if(!failure){          puts("Program Accepted");      }else{          puts("Program Rejected");      }      return 0;  } |

# Outputs

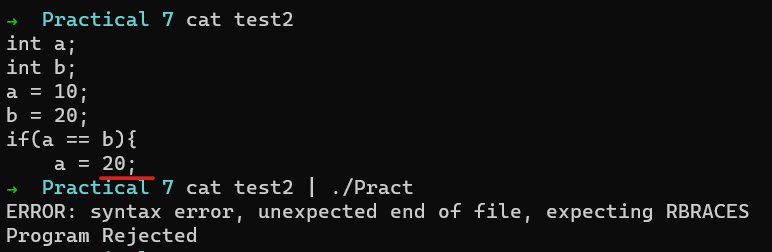
Compilation using flex and bison



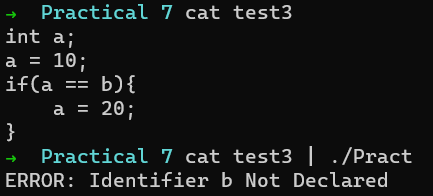
Source code is correct and types are compatible so accepted. No type checking error.



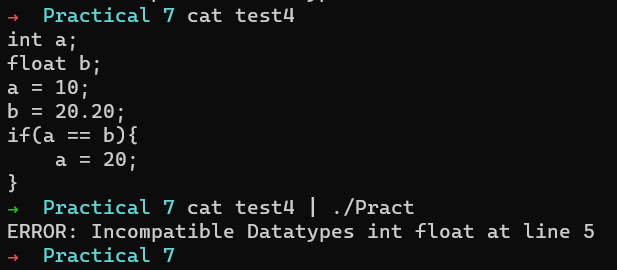
The If statement is missing closing bracket.



The declaration of identifier *b* is missing. Type checker also checks for declerations.



The types of variables *a* and *b* are incompatible. Type checker in action. :-)



# Conclusion

In this practical we implemented how identifiers works and how to create CFG grammar for language incorporating statements, assignments and comparisons. We included basic symbol table that can track the assignments and declaration with corresponding types. The type checker is invoked on comparisons, assignments, etc.