**temp4_r1_c8**

编译原理课程设计

设计报告

成员： 09013430 任杰文

09013429 黄路遥

09013413 钱鑫

东南大学计算机科学与工程学院

二0 16 年 5 月

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 设计任务名称 | | SeuYACC | | | |
| 完成时间 | | 2016/5/30 | 验收时间 |  | |
| 本组成员情况 | | | | | |
| 学 号 | 姓 名 | 承 担 的 任 务 | | | 成 绩 |
| 09013430 | 任杰文 | Lex全部任务：分析，设计，实现。  相关文档报告编写。 | | |  |
| 09013429 | 黄路遥 | YACC：完成yySeuYacc分析程序的生成，利用PDA配合Lex完成测试程序的读入和分析，采用语法制导的翻译（SDT）的方式进行中间代码生成 | | |  |
| 09013413 | 钱鑫 | YACC: YACC文件的读取，实现了求FIRST、CLOSUER等函数，完成了LR(1)项目集的构造及相关table的生成，LALR（1）项目集的构造及相关table的生成。 | | |  |

**注：本设计报告中各部分如果页数不够，请自行扩页。原则是一定要把报告写详细，能说明本组设计的成果和特色，能够反映小组中每个人的工作。报告中应该叙述设计中的每个模块。设计报告将是评定各人成绩的重要依据之一。**

|  |
| --- |
| 1 **编译对象与编译功能** |
| 1.1 **编译对象**  （作为编译对象的C语言子集的词法、语法描述）  Minic.y的内容如下  %{  /\* minic.y(1.9) 17:46:21 97/12/10  \*  \* Parser demo of simple symbol table management and type checking.  \*/  #include <stdio.h> /\* for (f)printf() \*/  #include <stdlib.h> /\* for exit() \*/  extern int line = 1; /\* number of current source line \*/  extern int seulex(); /\* lexical analyzer generated from lex.l \*/  char \*yytext; /\* last token, defined in lex.l \*/ /\* current symbol table, initialized in lex.l \*/  char \*base; /\* basename of command line argument \*/  void  yyerror(char \*s)  {  fprintf(stderr,"Syntax error on line #%d: %s\n",line,s);  fprintf(stderr,"Last token was \"%s\"\n",yytext);  exit(1);  }  %}  %union {  char\* name;  int value;  T\_LIST\* tlist;  T\_INFO\* type;  SYM\_INFO\* sym;  SYM\_LIST\* slist;  }  %token INT FLOAT NAME STRUCT IF ELSE RETURN NUMBER LPAR RPAR LBRACE RBRACE  %token LBRACK RBRACK ASSIGN SEMICOLON COMMA DOT PLUS MINUS TIMES DIVIDE EQUAL  %type <name> NAME  %type <value> NUMBER  %type <type> type parameter exp lexp  %type <tlist> parameters more\_parameters exps  %type <sym> field var  %type <slist> fields  %left ELSE  %right EQUAL  %left PLUS MINUS  %left TIMES DIVIDE  %left UMINUS  %left DOT LBRACK  %%  program : declarations  ;  declarations : declaration declarations  |  ;  declaration : fun\_declaration  | var\_declaration  ;  fun\_declaration : type NAME { /\* this is $3 \*/  $<sym>$ = symtab\_insert(scope,$2,0);  scope = symtab\_open(scope); /\* open new scope \*/  scope->function = $<sym>$; /\* attach to this function \*/  }  LPAR parameters RPAR { /\* this is $7 \*/  $<sym>3->type = types\_fun($1,$5);  }  block { scope = scope->parent; }  ;  parameters : more\_parameters { $$ = $1; }  | { $$ = 0; }  ;  more\_parameters : parameter COMMA more\_parameters  { $$ = types\_list\_insert($3,$1); }  | parameter { $$ = types\_list\_insert(0,$1); }  ;  parameter : type NAME {  symtab\_insert(scope,$2,$1); /\* insert in symbol table \*/  $$ = $1; /\* remember type info \*/  }  ;  block : LBRACE { scope = symtab\_open(scope); }  var\_declarations statements RBRACE  { scope = scope->parent; /\* close scope \*/}  ;  var\_declarations: var\_declaration var\_declarations  |  ;  var\_declaration : type NAME SEMICOLON { symtab\_insert(scope,$2,$1); }  ;  type : INT { $$ = types\_simple(int\_t); }  | FLOAT { $$ = types\_simple(float\_t); }  | type TIMES { $$ = types\_array($1); }  | STRUCT LBRACE fields RBRACE  { $$ = types\_record($3); }  ;  fields : field fields { $$ = symtab\_list\_insert($2,$1); }  | { $$ = 0; }  ;  field : type NAME SEMICOLON { $$ = symtab\_info\_new($2,$1); }  ;  statements : statement SEMICOLON statements  |  ;  ifhead : IF LPAR exp RPAR  ;  ifstatement : ifhead statement  ;  statement : ifstatement  | ifstatement ELSE statement  | lexp ASSIGN exp { check\_assignment($1,$3); }  | RETURN exp  { check\_assignment(scope->function->type->info.fun.target,$2); }  | block  ;  lexp : var { $$ = $1->type; }  | lexp LBRACK exp RBRACK{ $$ = check\_array\_access($1,$3); }  | lexp DOT NAME { $$ = check\_record\_access($1,$3); }  ;  exp : exp DOT NAME { $$ = check\_record\_access($1,$3); }  | exp LBRACK exp RBRACK { $$ = check\_array\_access($1,$3); }  | exp PLUS exp { $$ = check\_arith\_op(PLUS,$1,$3); }  | exp MINUS exp { $$ = check\_arith\_op(MINUS,$1,$3); }  | exp TIMES exp { $$ = check\_arith\_op(TIMES,$1,$3); }  | exp DIVIDE exp { $$ = check\_arith\_op(DIVIDE,$1,$3); }  | exp EQUAL exp { $$ = check\_relop(EQUAL,$1,$3); }  | LPAR exp RPAR { $$ = $2; }  | MINUS exp  { $$ = check\_arith\_op(UMINUS,$2,0); }  | var { $$ = $1->type; }  | NUMBER { $$ = types\_simple(int\_t); }  | NAME LPAR RPAR { $$ = check\_fun\_call(scope,$1,0); }  | NAME LPAR exps RPAR { $$ = check\_fun\_call(scope,$1,&$3); }  ;  exps : exp { $$ = types\_list\_insert(0,$1); }  | exp COMMA exps { $$ = types\_list\_insert($3,$1); }  ;  var : NAME { $$ = check\_symbol(scope,$1); }  ;  %%  int  main(int argc,char \*argv[])  {  base = "test.c";  yyparse();  }  1.2 **编译功能**  （所完成的项目功能及对应的程序单元）  FileReader.h中  FileReader类:mimicry文件的读入，并将内容分割为C语言头定义段，yacc定义段，规则段，语义定义段和用户子程序段。  ItemStructure.h中  Symbol类，存放终结符以及非终结符。  Rule类，存放单条语法规则。  RuleSet类，存放所有的语法规则，为其他部分提供查询功能，并通过calFF()成员函数完成First集合的计算。同时存放优先级结合律等信息，供其它部分使用。  Item类，存放语法规则的编号和点所在位置以及预测符号集，同时实现了两种判断相等依据以供上层使用。还提供了项层面上的合并函数，供上层使用。  Itemset类，存放项目集，同时实现了两种判断相等依据以供LR和LALR使用，提供了项目集层面上的合并函数，供上层使用。提供了项目集层面上的hash方法。完成闭包的计算，提供了移进项目集的计算。  CalItemSets类，完成LR项目集和LALR项目集的计算，完成冲突处理以及ActionTable和GotoTable的生成。  SeuYacc文件生成部分：SeuYacc分析minic.y后生成yySeuYacc.h、actionTable、gotoTable、translation.h四个文件。  通过SeuYacc根据minic.y生成yySeuYacc后，还需要一些额外的头文件进行语法分析和语义分析，这些头文件包括：  PDA.h中  PDA类，PDA维护一个下推栈和一个状态栈，实现了规约和移进的算法，并初步实现了错误分析程序，可以根据语法分析错误类型，给出错误位置提示，并打印错误处的栈状态方便进行分析。其中PDA根据随yySeuYacc生成actiontable和gototable进行推导，并在每次规约的时候调用minic.y预定义好的语义程序段进行语义分析，生成四元式  SDT.h和Quadurple.h中  实现了四元式的结构，并定义了语义栈、TC栈和FC栈。语义栈与语法分析同步进行了，将分析到的token存储，并在规约的时候弹栈生成对应的中间代码表示。TC栈和FC栈分别在遇到分支语句的时候用于存储真链和假链，当完成if语句中then和else 部分的分析后，通过对TC和FC'弹栈进行真链和假链的拉链回填操作。 |
| 2. **主要特色** |
| 实现了所有要求，通过标记的方式减少了求CLOSUER过程中的消耗。  可以通过LEFT、RIGHT等定义优先级结合律从而处理冲突。  求LALR项目集的过程中做到了边求边合并，节省了空间消耗。  完成了基本的语义分析，可以根据在minic.y中自定义的语义程序进行语义分析，并生成四元式作为中间代码。  可以进行布尔语句和分支语句的正确翻译，并用回填技术进行拉链操作。 |

|  |
| --- |
| 3 **概要设计与详细设计** |
| （由总到分地介绍SeuLex和SeuYACC的设计，包括模块间的关系，具体的算法等。采用面向对象方法的，同时介绍类（或对象）之间的关系。在文字说明的同时，尽可能多采用规范的图示方法。）  3.1 **概要设计**  （以描述模块间关系为主）  SeuYACC主要包含9个类及模块   1. Symbol类，存放终结符以及非终结符。   class Symbol//终结符与非终结符  {  public:  string \_name;  int \_type;//0非终结符，1终结符  }   1. Rule类，由Symbol组成   class Rule//单条规则  {  public:  Symbol Left;//产生式左部  vector<Symbol> Right;//产生式右部  }   1. RuleSet类，由Rule组成，提供计算First的功能   class RuleSet//规则集 id->rule，是数据主要存放的地方  {  public:  vector<Rule> rules;//规则  map<Symbol, set<Symbol>> Firstset;//First集合  //在读yacc文件时存放的额外信息  set<string> Token;  set<string> Left;  set<string> Right;  set<string> Nonassoc;  map<string, int> priority;  //符号集合  set<Symbol> Symbols;  private:  map<Symbol, vector<int>> queryMap;  int ruleNum;  };   1. Item类，由ruleId和dot两个整数表示产生式和点的位置，以及Symbol的集合表示预测符集   class Item  {  public:  int ruleId;//产生式编号  int dot;//点的位置  set<Symbol> predict;//预测符集合  };   1. ItemSet类，由Item组成，提供计算Closure的功能   class ItemSet  {  public:  set<Item> items;//项的集合  };   1. CalItemSets类，由ItemSet组成，计算过程中会使用RuleSet类中的数据   ，提供计算项目集族，填actiontable和gototable的功能  class CalItemSets  {  public:  vector<ItemSet> vItemSet;//项目集族  map<pair<int, Symbol>, int> edgeSet;//<id,Symbol>->id  map<pair<int, Symbol>, string> actionTable;  map<pair<int, Symbol>, int> gotoTable;  unordered\_map<string, int> itemHash;//hash表  ItemSet start;//起点  int index;  };  7）seuYacc 的main函数  主要进行minic.y文件的读取分析，输出action table和goto table，生成yyseuYacc通用部分（yyparse（）实现）并和用户程序段组合形成完整的分析程序。再根据minic.y中语义定义部分，生成语义分析辅助程序，并保存为translation.h。  生成的yyparse如下，主要完成调用yySeuLex生成token并调用PDA进行语法分析。  void yyparse()  {  FileReader fd;  fd.ParseYFile("minic.y");  vector<Symbol> s;  ifstream fin(base);  string str;  while (1)  {  string t = seuLex(fin);  if (t == "ERROR") break;  if (t != "SPACE"){  Symbol sym(t,1,seuLexLastLex);  s.push\_back(sym);}  }  Symbol sym("$",1);  s.push\_back(sym);  fin.close();  PDA pda(rs);  pda.input(s);  pda.readinTables("actionTable","gotoTable");  pda.parse();  }  8）PDA类，主要辅助生成的yySeuYacc进行语法分析，其中封装了标准的shift和reduce过程。根据LALR生成的action table和goto table即可进行语法分析。  class PDA  {  public:  PDA();  PDA(ruleSet RS);  PDA(vector<rule> RS);  ruleSet rs;  vector<Symbol> sentence; //Sentence waiting to be parsed  int reader; //where the PDA is reading  stack<int> stateStack;  stack<Symbol> PDstack;  vector<Symbol> stackData;  map<pair<int, Symbol>, string> actionTable;  map<pair<int, Symbol>, int> gotoTable;  public:  void readinTables(string actionfile, string gotofile);  void parse(); //true yyparse()  void input(vector<Symbol> s);  void shift();  Symbol reduce(rule r); //reduce with a given rule  bool isSuccess();  bool isError();  void success();  void error(int);  };  9）语法制导翻译和四元式部分  主要提供了一个四元式生成、临时变量生成的函数，并定义了语义栈、真链和假链，可以在语义程序中使用，用于分析分支程序。  Quadurple.h:  class quadurple  {  public:  string sym; //the symbol of the operation  string result;  string i; //third element of a quadurple  string j; //fourth element of a quadurple  quadurple(string SYM,string RESULT,string I,string J);  friend ostream& operator << (ostream& out, quadurple);  };  SDT.h  stack<string> SDTStack;  stack<int> TC;  stack<int> FC;  vector<quadurple> quadSet;  int tempID = 0;  int lineo = 0;  ofstream fout("IntermediaCode.c");  string intToString(int a)  {  char c[100] = "";  itoa(a,c,10);  string C(c);  return c;  }  string newTemp() //produce a new temp variable of Tx  {  char id[20] = "";  tempID++;  itoa(tempID,id,10);  string ID(id);  return "T"+ID;  }  void gen(string sym,string result,string i, string j) //generate a new quadruple  {    quadurple q(sym,result,i,j);  quadSet.push\_back(q);  lineo++;  }  void quadOutput()  {  for (int i = 0;i<quadSet.size();i++)  {  fout<<i<<" : ";  fout<<quadSet[i];  }  }  3.2 **详细设计**  （以描述数据结构及算法实现为主）  求Closure算法  龙书上的算法：    实际处理过程中，应该要注意对于每一个项只会被扩展一次，但是扩展过程中它的预测符集可能会被修改多次，所以采用了一个队列对要修改的项进行维护。  void Closure（项集 I）  {  队列 q;  将初始项加入q尾部。  while (!q.empty())  {  取出q的队首元素t；  if (t中点已经到达结束位置)  continue;  Symbol st = t中点所在位置的符号;  if (st是终结符)  continue;  对于形如的项，计算预测符号集b=First()  for (st为产生式左部的每个产生式)  if (已经加入了I，且那项为T)  将b合并到T的预测符号集合里  if T的预测符号集变化了  将合并后的T加入q尾部  else  将加入到I里，并加入q尾部  }  }  求First算法  龙书上的算法    实际处理中，因为要考虑到成环的情况，所以通过一个数组记录了每个产生式的使用情况，避免了无限递归。  int getFirstSet(int ruleId, set<Symbol> &firstSet, set<Symbol> &isUsed)  {  if 该规则的右部为ϵ  返回 0  else {  对于ruleID对应的产生式R;  i=0;  if R形如A→αB(α为终结符)  把α加入firstSet  else  {  那么R形如  当i小于R右部长度n时  对于每个以Xi为产生式左部的产生式r计算getFirstSet(r, firstSet, isUsed)  if 对于Xi的每个产生式出的返回值都不为0，即其对应First都不包含ϵ  break  i++  if i==R右部长度n  把ϵ加入firstSet并返回 0  }  返回 1;  }  }  生成项集族算法（LR（1））  void CalItemSets::genPDALR()  {  index=1  将文法开始符号及对应产生式加入项集start，并计算其Closure  将start加入项集族vItemSet  将start加入队列q尾部  讲<start,index>存入hash表itemHash中  index++  while (!q.empty())  {  取出q队首的项集temp  根据item中各项点的位置，所对应的非终结符，组成候选扩展符号集cand  for (cand中的每个符号s)  {  计算temp接受s后会移进到的项集p  if p不在hash表itemHash中  {  将p加入项集族vItemSet  讲<p,index>存入hash表itemHash中  index++  将<<temp,s>,p>存入edgeSet（此处为了节省空间使用的是temp和p所对应的index）  将p加入队列q尾部  }  else  {  找到p所对应编号pindex  将<<temp,s>,p>存入edgeSet（此处为了节省空间使用的是temp和p所对应的index，p的index也就是pindex）  }  }  }  }  这部分的hash函数是利用项集的每个项及预测符号集合计算出来的  生成项集族算法（LALR（1））  void CalItemSets::genPDALR()  {  index=1  将文法开始符号及对应产生式加入项集start，并计算其Closure  将start加入项集族vItemSet  将start加入队列q尾部  讲<start,index>存入hash表itemHash中  index++  while (!q.empty())  {  取出q队首的项集temp  根据item中各项点的位置，所对应的非终结符，组成候选扩展符号集cand  for (cand中的每个符号s)  {  计算temp接受s后会移进到的项集p  if p不在hash表itemHash中  {  将p加入项集族vItemSet  讲<p,index>存入hash表itemHash中  index++  将<<temp,s>,p>存入edgeSet（此处为了节省空间使用的是temp和p所对应的index）  将p加入队列q尾部  }  else  {  找到p所对应编号pindex  将p与在itemHash中记录的同心项集i1合并得到i2，并保存到i1中  将<<temp,s>,p>存入edgeSet（此处为了节省空间使用的是temp和p所对应的index，p的index也就是pindex）  if i2!=i1  讲i2存入q尾部  }  }  }  }  这部分的hash函数是利用项集的每个项所用规则以及点的位置计算出来的  LALR(1)和LR(1)的计算项目集族的方法还是比较像的，都采用了类似BFS+Hash的方法，避免重复扩展。通过这个过程同时也得到了Goto函数，也就是edgeSet中的内容。  冲突处理及填表算法  int recur(int state ,int\*flag)  {  if 该state已经处理过  返回 0;  flag[state] = 1;  根据is中每个项中点的位置，把这些项归到可规约集合reduceV和可移进集合cand  if (可归约集合reduceV不为空)  {  for (可规约集合reduceV里的每一项it)  {  根据it的ruleId和预测符号集predict，往actionTable中填入相应规约动作，如果actionTable对应位置已有动作则不覆盖，并加入已处理符号集produced，且ruleID为0时填入acc  }  }  for (可移进集合cand每一项s)  {  根据edgeSet得到在当前state和s下会进入的状态next  if s不在已处理符号集produced中  {  根据当前状态state和s还有next往actionTable和gototable中填入相应动作  }  else  {  根据s和已有的优先级以及结合性等信息，确定是否覆盖原来的规约动作，例如本状态的操作优先级低于下个状态的，那么就移进，如果本状态优先级更高就选择规约。如果优先级都一样，s是左结合的就移进，是右结合的就规约  }  递归处理recur(next, mem, flag);  }  返回 0;  }  总体上来说是一个dfs遍历所有项集的过程，在每一个项集按照规则进行处理。  PDA根据action table和goto table进行语法分析：  （1）shift（移进）  将当前读头下的token压下推栈（语义值同时压栈），读头向后移动一个。若超出分析的程序长度，报错reader越界。  （2）reduce（规约）  根据给定规则R进行规约，记为R规则产生式右部规则包含的token个数，将当前下推栈中弹出个元素。每次弹出的时候检查弹出的token是否与当前规则匹配，若不匹配则报错规则不匹配。否则弹栈结束后，将R左侧token压栈。  （3）parse（分析主程序）  初始状态栈栈顶为0，读头指向第一个token。  While（1）  ｛  s = actionTable(stateStack.top ,Sentence[reader]  if (s == error)  error()//错误处理程序  if（s==acc）  success()//提示分析成功  if （s为s+n（n为数字）的结构（即进行移进））  PDStack.push(n)  Shift()  else  //s为r+n的结构（即进行规约）  Reduce(n);  将stateStack栈顶个元素弹栈，并将gotoTable(stateStack.top, L)压栈  运行语义分析程序  ｝  中间代码生成部分：该部分主要分析四则运算语句和条件分支语句，并产生他们的四元式。  四则运算：（以加法为例）  语义程序：T = newTemp（）//产生一个新的临时变量  Gen（'+'，T，$1，$3)  SDTStack.push(T)//新产生的临时变量进入语义栈  赋值语句  语义程序：Gen（'='，$1，\_，$3)  分支语句：（if-then-else为例）  ifhead ---> IF LPAR exp RPAR  语义程序： gen("j",$1,$3,"0");//目标地址记0，待回填  TC.push(lineo); //当前行数记为真链回填位置，下一行为假链回填位置  FC.push(lineo+1);  gen("j","\_","\_","0"); //下一句为一个无条件跳转语句  quadSet[TC.top()-1].j = lineo;//本句已经可以回填，进行回填  TC.pop();//弹出已经回填的行号  ifstatement ---> ifhead statement  语义程序： gen("j","\_","\_","0");//分析完then部分进行一次无条件跳转，跳转到if-then-else模块后一句，暂时不知道地址，待回填  TC.push(lineo);//将当前行号压入真链，待回填  quadSet[FC.top()-1].j =lineo; //同时该位置为上一个假链的跳转位置（马上执行else语句），因此进行一次回填  FC.pop();  statement :---> ifstatement ELSE statement  语义程序： quadSet[TC.top()-1].j = lineo; //当前行号为真链栈顶语句应跳转的位置，进行回填  TC.pop();  语义分析和语法分析同步进行，每次规约的时候进行一次四元式的产生（对于可以产生四元式的规则而言）。在进行语法分析同时维护一个SDTstack，其中存储与语法分析token对应的源程序的符号。 |

|  |
| --- |
| 4 **使用说明** |
| **4.2 SeuYacc使用说明**  **（1）准备好yacc文件，如minic.y，运行seuYacc。**  **（2）若运行成功会在子文件夹genCode中生成actionTable、gotoTable、translation.h、yySeuYacc.cpp四个文件。**  **（3）要进行语法分析，还需要准备FileReader.h、ItemStructure.h、PDA.h、quadruple.h、SDT.h、yySeuLex.h以及它们对应的cpp文件。（可以将这些文件重新组成一个工程文件）**  **（4）准备待分析的主程序，如test.c（根据minic.y中用户程序决定）**  **（5）运行分析程序，若运行成功会得到分析成功提示，并生成intermediateCode.c文件。其中包含的即是根据语法分析产生的中间代码文件；若分析失败会提示分析失败时读头位置，打印出当前栈状态方便分析，同时错误信息会保存在ParsingLog文件中。** |

|  |
| --- |
| 5 **测试用例与结果分析** |
| （1）采用的minic.y如前文所示，运行SeuYacc程序部分结果如图      同时genCode文件夹下生成如下文件。    actionTable内容（稀疏方式存储的actionTable内容）    gotoTable内容（稀疏存储的gotoTable内容）    Translation.h（四元式生成程序）  #pragma once  #include "ItemStructure.h"  #include "SDT.h"  #include "quadruple.h"  void translation(rule R,int ruleID)  {  string temp;  string sym[20];  switch (ruleID)  {  case 24 :  for (int i = 0;i<R.Right.size();i++)  {  sym[i] = SDTStack.top();  SDTStack.pop();  }  gen("j"+sym[2],sym[1],sym[3],"0");TC.push(lineo);FC.push(lineo+1);gen("j","\_","\_","0");quadSet[TC.top()-1].j=intToString(lineo);TC.pop();  break;  case 25 :  for (int i = 0;i<R.Right.size();i++)  {  sym[i] = SDTStack.top();  SDTStack.pop();  }  gen("j","\_","\_","0");TC.push(lineo);quadSet[FC.top()-1].j=intToString(lineo);FC.pop();  break;  case 27 :  for (int i = 0;i<R.Right.size();i++)  {  sym[i] = SDTStack.top();  SDTStack.pop();  }  quadSet[TC.top()-1].j=intToString(lineo);TC.pop();  break;  case 28 :  for (int i = 0;i<R.Right.size();i++)  {  sym[i] = SDTStack.top();  SDTStack.pop();  }  gen(sym[1],sym[2],"\_",sym[0]);  break;  case 36 :  for (int i = 0;i<R.Right.size();i++)  {  sym[i] = SDTStack.top();  SDTStack.pop();  }  temp=newTemp();gen(sym[1],temp,sym[0],sym[2]);SDTStack.push(temp);  break;  case 37 :  for (int i = 0;i<R.Right.size();i++)  {  sym[i] = SDTStack.top();  SDTStack.pop();  }  temp=newTemp();gen(sym[1],temp,sym[0],sym[2]);SDTStack.push(temp);  break;  case 38 :  for (int i = 0;i<R.Right.size();i++)  {  sym[i] = SDTStack.top();  SDTStack.pop();  }  temp=newTemp();gen(sym[1],temp,sym[0],sym[2]);SDTStack.push(temp);  break;  case 39 :  for (int i = 0;i<R.Right.size();i++)  {  sym[i] = SDTStack.top();  SDTStack.pop();  }  temp=newTemp();gen(sym[1],temp,sym[0],sym[2]);SDTStack.push(temp);  break;  default:  break;  }  }  yySeuYacc.cpp（语法分析主程序）  #include "PDA.h"  #include <vector>  #include "yyseuLex.h"  #include "FileReader.h"  /\* minic.y(1.9) 17:46:21 97/12/10  \*  \* Parser demo of simple symbol table management and type checking.  \*/  #include <stdio.h> /\* for (f)printf() \*/  #include <stdlib.h> /\* for exit() \*/  extern int line = 1; /\* number of current source line \*/  extern int seulex(); /\* lexical analyzer generated from lex.l \*/  char \*yytext; /\* last token, defined in lex.l \*/ /\* current symbol table, initialized in lex.l \*/  char \*base; /\* basename of command line argument \*/  void  yyerror(char \*s)  {  fprintf(stderr,"Syntax error on line #%d: %s\n",line,s);  fprintf(stderr,"Last token was \"%s\"\n",yytext);  exit(1);  }  extern ruleSet rs;  void yyparse()  {  FileReader fd;  fd.ParseYFile("minic.y");  vector<Symbol> s;  ifstream fin(base);  string str;  while (1)  {  string t = seuLex(fin);  if (t == "ERROR") break;  if (t != "SPACE"){  Symbol sym(t,1,seuLexLastLex);  s.push\_back(sym);}  }  Symbol sym("$",1);  s.push\_back(sym);  fin.close();  PDA pda(rs);  pda.input(s);  pda.readinTables("actionTable","gotoTable");  pda.parse();  }  int  main(int argc,char \*argv[])  {  base = "test.c";  yyparse();  }  （2）根据使用说明中的方法，将若干其他辅助分析程序头文件和cpp文件组合为语法分析程序后运行:  采用的test.c如下：  int main()  {  int k;  float m;  if (a == 10)  {  if (c == 8)  {  a = 6;  }  else  {  c = 9;  };  }  else  {  b = 8;  };  v = i;  k = k + c;  m = m + 2;  c = 2 - 3;  if (m == 5)  {  k = 8;  };  return 0;  }  1、test.c无语法错误  部分程序结果截图    生成的中间代码为intermediaCode.c，如下：  0 : (j== , 10 , a , 2)  1 : (j , \_ , \_ , 8)  2 : (j== , 8 , c , 4)  3 : (j , \_ , \_ , 6)  4 : (= , a , \_ , 6)  5 : (j , \_ , \_ , 7)  6 : (= , c , \_ , 9)  7 : (j , \_ , \_ , 9)  8 : (= , b , \_ , 8)  9 : (= , v , \_ , i)  10 : (+ , T1 , c , k)  11 : (= , k , \_ , T1)  12 : (+ , T2 , 2 , m)  13 : (= , m , \_ , T2)  14 : (- , T3 , 3 , 2)  15 : (= , c , \_ , T3)  16 : (j== , 5 , m , 18)  17 : (j , \_ , \_ , 20)  18 : (= , k , \_ , 8)  19 : (j , \_ , \_ , 0)  2、test.c存在语法错误  （第20行缺少一个分号）  int main()  {  int k;  float m;  if (a == 10)  {  if (c == 8)  {  a = 6;  }  else  {  c = 9;  };  }  else  {  b = 8;  };  v = i  k = k + c;  m = m + 2;  c = 2 - 3;  if (m == 5)  {  k = 8;  };  return 0;  }  程序运行结果：    生成的ParsingLog文件：  ERROR!  Reader at : k  Error info:Shift into error state!  state stack:  48  54  39  57  40  32  30  PD stack:  NAME  ASSIGN  lexp  SEMICOLON  statement  var\_declarations  标明在读到变量k之前遇到了错误，即在k之前缺少分号 |

|  |
| --- |
| 6 **课程设计总结**（包括设计的总结和需要改进的内容） |
| 本次实验比较完整地实现了全部要求，通过本次实验对LR文法和下推自动机有了更深刻的认识，加强了编码及调试的能力。在调试LR文法的过程中，因为中间过程和数据结构比较复杂，充分意识到了一个好的思路和设计可以减少很多不必要的问题。完成代码过程中也应当注意一些用于调试的接口函数的实现，这样可以极大地方便调试过程。  我们的SEUYacc仍然有些问题需要解决，   1. 部分代码不够简洁。因为部分代码在编写过程中思路不太清晰，改了又改，最后debug时出了问题又调整了，导致有一些冗余代码。如果有时间重构一遍，性能和代码可读性还能更好。 2. 在冲突处理上，默认采用对于规约规约冲突采用先规约序号小的项，移进规约冲突采用先移进的策略。仅提供了全局的结合性以及优先级的控制，没提供对于单条规则的特例的冲突处理功能。   语法分析程序生成和四元式生成部分：  总结：这部分主要完成了yySeuYacc程序的生成，并构造了辅助分析的下推自动机配合LALR部分生成的两个表格进行表格制导的语法分析。同时，在生成了四元式生成的辅助程序，可以根据用户预定义的语义动作进行语义分析，并提供了语义栈、真链栈、假链栈辅助进行分析。针对分析失败（即输入程序有语法错误）的情况，进行了简单的错误检查，并输出错误信息和错误时的堆栈信息。  需要改进的部分：  （1） 语义动作程序部分不能形式化表示，对于用户编写其他yacc程序的语义动作时有一定的困难（需要考虑语义栈、真链假链的填入弹出、回填等操作  （2）语义动作部分的函数并未完全封装，调用的时候会有困难（例如backpatch函数、merge函数）  （3）分支语句没有进行并链，有可能产生一些问题。  （4）错误分析程序还不完善，目前只能输出一些简单类型的错误，还不能分析错误原因。例如上文测试样例中，缺少分号只能提示在分号后的一个字符处有问题，不能提示是缺少分号导致的。 |
| 7 **教 师 评 语** |
| 签名： |

附：包含源程序、可运行程序、输入数据文件、输出数据文件、答辩所做PPT文件、本设计报告等一切可放入光盘的内容的光盘。