Parser.cpp

|  |
| --- |
| #include "Parser.h"  #include <stack>  #include <map>  #include <string>  #define $INT 1 //int  #define $IF 2 //if  #define $ELSE 3 //else  #define $FLOAT 4 //float  #define $PRINT 5 //print  #define $ID 6 //identify  #define $CONSTANT 7 //constant  #define $ASSIGN 8 //=  #define $PLUS 9 //+  #define $STAR 10 //\*  #define $POWER 11 //\*\*  #define $SEMICOLON 12 //;  #define $LPAR 13 //(  #define $RPAR 14 //)  #define $LBRACE 15 //{  #define $RBRACE 16 //}  #define $LZKH 17 //[  #define $RZKH 18 //]  #define $SUBSIGN 19 //-  #define $DIVISION 20 // /  #define $COMMA 21 //,  #define $AND 22 //&&  #define $OR 23 //||  #define $NOT 24 //!  #define $EQUATION 25 //==  #define $ABOVE 26 //>  #define $BELOW 27 //<  #define $ABOVEE 28 //>=  #define $BELOWE 29 //<=  #define $NOTE 30 //!=  #define $TEMP 31  #define $NULL 0  #define $JMP 32  #define $GOTO 33  #define $JL 34  static int lengthOfRule[] = {//记录弹出的数量  1, // <program> ::= <block>  4, // <block> ::= "{" <decls> <stmts> "}"  2, // <decls> ::= <decls> <decl>  0, // <decls> ::= ""  3, // <decl> ::= <type> id ";"  4, // <type> ::= <type> "[" num "]"  1, // <type> ::= basic  3, // <stmts> ::= <stmts> <stmt> <M> 7  0, // <stmts> ::= ""  4, // <stmt> ::= <loc> "=" <bool> ";"  6, // <stmt> ::= "if" "(" <bool> ")" <M> <stmt> 10  10, // <stmt> ::= "if" "(" <bool> ")" <M> <stmt> <N> "else" <M> <stmt> 11  7, // <stmt> ::="while" <M> "(" <bool> ")" <M> <stmt> 12  9, // <stmt> ::= "do" <M> <stmt> "while" <M> "(" <bool> ")" ";"  2, // <stmt> ::= "break" ";"  1, // <stmt> ::= <block>  4, // <loc> ::= <loc> "[" num "]"  1, // <loc> ::= id  4, // <bool> ::= <bool> "||" <M> <join> 18  1, // <bool> ::= <join> 19  4, // <join> ::= <join> "&&" <M> <equality> 20  1, // <join> ::= <equality> 21  3, // <equality> ::= <equality> "==" <rel> 22  3, // <equality> ::= <equality> "!=" <rel> 23  1, // <equality> ::= <rel> 24  3, // <rel> ::= <expr> "<" <expr> 25  3, // <rel> ::= <expr> "<=" <expr> 26  3, // <rel> ::= <expr> ">=" <expr> 27  3, // <rel> ::= <expr> ">" <expr> 28  1, // <rel> ::= <expr> 29  3, // <expr> ::= <expr> "+" <term> 30  3, // <expr> ::= <expr> "-" <term>  1, // <expr> ::= <term>  3, // <term> ::= <term> "\*" <unary>  3, // <term> ::= <term> "/" <unary>  1, // <term> ::= <unary>  2, // <unary> ::= "!" <unary>  2, // <unary> ::= "-" <unary>  1, // <unary> ::= <factor>  3, // <factor> ::= "(" <bool> ")"  1, // <factor> ::= <loc>  1, // <factor> ::= num  1, // <factor> ::= real  1, // <factor> ::= "true"  1, // <factor> ::= "false"  0,  0  };  struct mymap  { int code;  std::string value;  };  struct list  { int value;  struct list \*next;  };  int place=1;  int nextpos=1;  std::stack<list \*> stknext;  std::stack<list \*> stktrue;  std::stack<list \*> stkfalse;  std::stack<int> stkpos;  std::stack<mymap> stktemp;//常量，变量，临时变量  mymap gen[100][4];//生成的四元式  void backpatch(list \*list1,int t)  { int i;  while(list1)  {i=list1->value;  gen[i][3].code=$GOTO;  gen[i][3].value=std::to\_string(t);  list1=list1->next;  }  }  list\* mymerge(list \*list1,list \*list2)  { list \*list3=list1;  if(list1==NULL)  {return list2;  }  while(list3->next)  list3=list3->next;  list3->next=list2;  return list1;  }  void tiaojian\_3() //S->if ( B ) M S1 10  {  list \*mytrue=stktrue.top();  stktrue.pop();  list \*myfalse=stkfalse.top();  stkfalse.pop();  int mypos=stkpos.top();  stkpos.pop();  backpatch(mytrue,mypos);  // std::cout<<"333";  list \*mynext=stknext.top();  stknext.pop();  mynext=mymerge(myfalse,mynext);  stknext.push(mynext);  }  void tiaojian\_while() //S->while  {  list \*mytrue=stktrue.top();  stktrue.pop();  list \*myfalse=stkfalse.top();  stkfalse.pop();    int myposm2=stkpos.top();  stkpos.pop();  int myposm1=stkpos.top();  stkpos.pop();  list \*mynexts1=stknext.top();  stknext.pop();    backpatch(mynexts1,myposm1);  backpatch(mytrue,myposm2);    // list \*mynext=stknext.top();  // mynext=mymerge(myfalse,mynexts1);  stknext.push(myfalse);    gen[nextpos][0].code=$JMP;  gen[nextpos][1].code=0;  gen[nextpos][2].code=0;  gen[nextpos][3].value=std::to\_string(myposm1);  gen[nextpos][3].code=$GOTO;  nextpos++;  }  void tiaojian\_2() //S->if ( B ) M S1 10 else  {  list \*mytrue=stktrue.top();  stktrue.pop();  list \*myfalse=stkfalse.top();  stkfalse.pop();  int myposm2=stkpos.top();  stkpos.pop();  int myposm1=stkpos.top();  stkpos.pop();  list \*mynexts2=stknext.top();  stknext.pop();  list \*mynextn=stknext.top();  stknext.pop();  list \*mynexts1=stknext.top();  stknext.pop();    backpatch(mytrue,myposm1);  backpatch(myfalse,myposm2);    mynexts1=mymerge(mynexts1,mynexts2);  mynexts1=mymerge(mynexts1,mynextn);  stknext.push(mynexts1);  }  void tiaojian\_14() //M->0  {  stkpos.push(nextpos);  }  void tiaojian\_7() //B->!B1  { list \*mytrue=stktrue.top();  stktrue.pop();  list \*myfalse=stkfalse.top();  stkfalse.pop();  stktrue.push(myfalse);  stkfalse.push(mytrue);    }  void tiaojian\_15() //N->0  { list \*mylist=new list;  mylist->next=NULL;  mylist->value=nextpos;  stknext.push(mylist);  gen[nextpos][0].code=$JMP;  gen[nextpos][1].code=0;  gen[nextpos][2].code=0;  gen[nextpos][3].code=$GOTO;  nextpos++;  }  void tiaojian\_stmts() // <stmts> ::= <stmts> <stmt> <M>  {  list \*mynext=stknext.top();  //list \*myfalse=stkfalse.top();  if(mynext!=NULL)  { int mypos=stkpos.top();  stkpos.pop();  backpatch(mynext,mypos);  //stknext.pop();    }  }  void tiaojian\_5() // <join> ::= <join> "&&" <equality> 20  {  list \*mytrue2=stktrue.top();  stktrue.pop();  list \*mytrue1=stktrue.top();  stktrue.pop();  stktrue.push(mytrue2);  int mypos=stkpos.top();  stkpos.pop();  backpatch(mytrue1,mypos);  list \*myfalse2=stkfalse.top();  stkfalse.pop();  list \*myfalse1=stkfalse.top();  stkfalse.pop();  myfalse1=mymerge(myfalse1,myfalse2);  stkfalse.push(myfalse1);  }  std::shared\_ptr<Node> tiaojian\_8(std::stack<std::shared\_ptr<Node>> & nodes) // <rel> ::= <expr> "<" <expr> 25  { list \*mylist=new list;  mylist->next=NULL;  mylist->value=nextpos;  stktrue.push(mylist);  mylist=new list;  mylist->next=NULL;  mylist->value=nextpos+1;  stkfalse.push(mylist);    std::shared\_ptr<Node> right =nodes.top();  gen[nextpos][2].value=nodes.top()->ToString();  gen[nextpos][2].code=$ID;  nodes.pop();    Token t = nodes.top()->GetToken();  gen[nextpos][0].code=t.code;  // gen[nextpos][0].value="j<";  nodes.pop();  gen[nextpos][1].value=nodes.top()->ToString();  gen[nextpos][1].code=$ID;  std::shared\_ptr<Node> left =nodes.top();  nodes.pop();  gen[nextpos][3].code=$GOTO;  nextpos++;  // std::cout<<"四元式：("<<"j<"<<" "<<gen[nextpos][1].value<<" "<<"-"<<" "<<gen[nextpos][2].value<<")\n";    // return std::make\_shared<Node>(t, left, left);  gen[nextpos][0].code=$JMP;  gen[nextpos][1].code=0;  gen[nextpos][2].code=0;  gen[nextpos][3].code=$GOTO;  nextpos++;  // std::cout<<"四元式：("<<"j"<<" "<<gen[nextpos][1].value<<" "<<"-"<<" "<<gen[nextpos][2].value<<")\n";  return std::make\_shared<Node>(t, left, right);  }  void tiaojian\_6() // <bool> ::= <bool> "||" <join> 18  { list \*myfalse2=stkfalse.top();  stkfalse.pop();  list \*myfalse1=stkfalse.top();  stkfalse.pop();  stkfalse.push(myfalse2);  int mypos=stkpos.top();  stkpos.pop();  backpatch(myfalse1,mypos);  list \*mytrue2=stktrue.top();  stktrue.pop();  list \*mytrue1=stktrue.top();  stktrue.pop();  mytrue1=mymerge(mytrue1,mytrue2);  stktrue.push(mytrue1);  }  std::shared\_ptr<Node> MakeNode2(std::stack<std::shared\_ptr<Node>> & nodes)  {    std::shared\_ptr<Node> right = nodes.top();  nodes.pop();  Token t = nodes.top()->GetToken();  //printToken(t);  nodes.pop();  std::shared\_ptr<Node> left = nodes.top();  std::cout<<"四元式：("<<returntoken(t)<<" "<<left->ToString()<<" "<<"-"<<" "<<right->ToString()<<")\n";    nodes.pop();    gen[nextpos][2].value=right->ToString();  gen[nextpos][2].code= $ID;  gen[nextpos][0].value=t.type;  gen[nextpos][0].code= t.code;    gen[nextpos][1].value=left->ToString();  gen[nextpos][1].code= $ID;    gen[nextpos][3].value=std::to\_string(place) ;  gen[nextpos][3].code=$TEMP;  t.code=$TEMP;  t.codevalue=place;  nextpos++;  place++;  return std::make\_shared<Node>(t, left, right);  }  std::shared\_ptr<Node> Parser::Parse()  {  Token token = NextToken();  // std::stack<Token> op = {};  std::stack<std::shared\_ptr<Node>> nodes = {};  std::stack<int> states = {};  std::string signstates[50]={};  int signtop=-1;  states.push(0);  int top = states.top();  list \*mylist=new list;  mylist->next=NULL;  //mylist->value=nextpos;  stkfalse.push(mylist);  while (true)  {  top = states.top();  int indexOfToken = Convert(token);  int action = actionTable[top][indexOfToken];  printf("下一个");  printToken(token);  //std::cout<<indexOfToken<<"\n";  //printf("indexoftoken = %d, action = %d, top = %d\n", indexOfToken, action, top);  if (action > 0 && action < 500) // shift to some state  {  states.push(action);  signtop++;  //printToken(token);  //std::cout<<(returntoken(token));  signstates[signtop]=returntoken(token);    std::cout<<signtop;  //printf("移入：actiontable 列 = %d, action = %d, 行 = %d\n", indexOfToken, action, top);  printf("移入，actiontable位置（列 = %d,行 = %d）,action = %d 。\n", indexOfToken, top,action);  printf("\n");  if (token.type == NUM){  token.code=$ID;  //printf("integer %d \n",token.value.integer);  nodes.push(std::make\_shared<Node>(token));  //stktemp.push()  }    else if (token.type == PLUS) {  //op.push(token);  token.code=$PLUS;  nodes.push(std::make\_shared<Node>(token));  //printf("op %d \n",token.value.integer);  } else if (token.type == MINUS) {  //op.push(token);  token.code=$SUBSIGN;  nodes.push(std::make\_shared<Node>(token));  //printf("op %d \n",token.value.integer);  }else if (token.type == ID) {  token.code=$ID;  nodes.push(std::make\_shared<Node>(token));  //printf("id %s \n",token.value.string);  }else if (token.type == LESS) {  //op.push(token);  token.code=$BELOW;  nodes.push(std::make\_shared<Node>(token));  //printf("op %d \n",token.value.integer);  }else if (token.type == GREATER) {  //op.push(token);  token.code=$ABOVE;  nodes.push(std::make\_shared<Node>(token));  //printf("op %d \n",token.value.integer);  }else if (token.type == LESSEQUAL) {  //op.push(token);  token.code=$BELOWE;  nodes.push(std::make\_shared<Node>(token));  //printf("op %d \n",token.value.integer);  }  else if (token.type == GREATEREQUAL) {  //op.push(token);  token.code=$ABOVEE;  nodes.push(std::make\_shared<Node>(token));  //printf("op %d \n",token.value.integer);  } else if (token.type == EQUAL) {  //op.push(token);  token.code=$EQUATION;  nodes.push(std::make\_shared<Node>(token));  //printf("op %d \n",token.value.integer);  }else if (token.type == NOTEQUAL) {  //op.push(token);  token.code=$NOTE;  nodes.push(std::make\_shared<Node>(token));  //printf("op %d \n",token.value.integer);  }  else if (token.type == ASSIGN) {  //op.push(token);  token.code=$ASSIGN;  nodes.push(std::make\_shared<Node>(token));  //printf("= %s \n",token.value.string);  }else if (token.type == AND) {  //op.push(token);  //token.code=$ASSIGN;  nodes.push(std::make\_shared<Node>(token));  //printf("= %s \n",token.value.string);  }  token = NextToken();    }  else if (action >= 500) // reduce a handle  {      int indexOfRule = action-500;  //std::cout<<indexOfRule<<".";  //int indexOfRule = action & ~(1 << 8); 这行代码将 action 的最高位清零，并将结果赋值给 indexOfRule 变量。  int i=0;  int j=lengthOfRule[indexOfRule];    for (i =0; i < j; i++)  {  //printf("pop = %d \n", i);  states.pop();  if (signtop>-1)signtop--;  }    printf("归约，对应文法序号 = %d，gototable 位置（列(未转换前) = %d, 行 = %d）。\n",indexOfRule, indexOfRule, states.top());  printf("\n");  //top = states.top();  //printf("lengthOfRule[indexOfRule] = %d, action = %d, top = %d\n", lengthOfRule[indexOfRule], action, states.top());    switch (indexOfRule)  {  case 0:  states.push(gotoTable[states.top()][program]);  //nodes.push(MakeNode(op, nodes));  signtop++;  signstates[signtop]="<program>";  break;  case 1:  states.push(gotoTable[states.top()][block]);  signtop++;  signstates[signtop]="<block>";  break;  case 2:case 3:  //printf("top = %d \n", gotoTable[states.top()][decls]);  states.push(gotoTable[states.top()][decls]);  //nodes.push(MakeNode(op, nodes));  signtop++;  signstates[signtop]="<decls>";  break;  case 4:  states.push(gotoTable[states.top()][decl]);  signtop++;  signstates[signtop]="<decl>";  //nodes.push(MakeNode(op, nodes));  break;  case 5: case 6:  states.push(gotoTable[states.top()][type]);  //nodes.push(MakeNode(op, nodes));  signtop++;  signstates[signtop]="<type>";  break;    case 7:  tiaojian\_stmts();  states.push(gotoTable[states.top()][stmts]);  signtop++;  signstates[signtop]="<stmts>";  //nodes.push(MakeNode(op, nodes));  break;  case 8:  states.push(gotoTable[states.top()][stmts]);  signtop++;  signstates[signtop]="<stmts>";  //nodes.push(MakeNode(op, nodes));  break;  case 9:// <stmt> ::= <loc> "=" <bool> ";"  MakeNode1(nodes);  states.push(gotoTable[states.top()][stmt]);  signtop++;  signstates[signtop]="<stmt>";  break;  case 10:  stknext.push(NULL);  states.push(gotoTable[states.top()][stmt]);  signtop++;  signstates[signtop]="<stmt>";  tiaojian\_3();  break;  case 11:  states.push(gotoTable[states.top()][stmt]);  signtop++;  signstates[signtop]="<stmt>";  tiaojian\_2();  //nodes.push(MakeNode(op, nodes));  break;  case 12:    states.push(gotoTable[states.top()][stmt]);  signtop++;  signstates[signtop]="<stmt>";  tiaojian\_while();  case 13:case 14:case 15:  states.push(gotoTable[states.top()][stmt]);  signtop++;  signstates[signtop]="<stmt>";  //nodes.push(MakeNode(op, nodes));  break;  case 16:case 17:  states.push(gotoTable[states.top()][loc]);  signtop++;  signstates[signtop]="<loc>";  //nodes.push(MakeNode(op, nodes));  break;  case 18:  tiaojian\_6();  states.push(gotoTable[states.top()][bool\_term]);  signtop++;  signstates[signtop]="<bool\_term>";  //nodes.push(MakeNode(op, nodes));  break;  case 19:  states.push(gotoTable[states.top()][bool\_term]);  signtop++;  signstates[signtop]="<bool\_term>";  //nodes.push(MakeNode(op, nodes));  break;  case 20:  tiaojian\_5();  states.push(gotoTable[states.top()][join]);  signtop++;  signstates[signtop]="<join>";  //nodes.push(MakeNode(op, nodes));  break;  case 21:  states.push(gotoTable[states.top()][join]);  signtop++;  signstates[signtop]="<join>";  //nodes.push(MakeNode(op, nodes));  break;  case 22:case 23: //eq  nodes.push(tiaojian\_8(nodes));    states.push(gotoTable[states.top()][rel]);  signtop++;  signstates[signtop]="<rel>";  //nodes.push(MakeNode(op, nodes));  break;    case 24:  states.push(gotoTable[states.top()][equality]);  signtop++;  signstates[signtop]="<equality>";  //nodes.push(MakeNode(op, nodes));  break;  case 25:case 26: case 27:case 28:    nodes.push(tiaojian\_8(nodes));    states.push(gotoTable[states.top()][rel]);  signtop++;  signstates[signtop]="<rel>";  //nodes.push(MakeNode(op, nodes));  break;  case 29:  states.push(gotoTable[states.top()][rel]);  signtop++;  signstates[signtop]="<rel>";  //nodes.push(MakeNode(op, nodes));  break;  case 30: //plus  signtop++;  signstates[signtop]="<expr>";  states.push(gotoTable[states.top()][expr]);  nodes.push(MakeNode2(nodes));  break;  case 31:  signtop++;  signstates[signtop]="<expr>";  states.push(gotoTable[states.top()][expr]);  nodes.push(MakeNode2(nodes));  break;  case 32:  signtop++;  signstates[signtop]="<expr>";  states.push(gotoTable[states.top()][expr]);  //nodes.push(MakeNode(op, nodes));  break;  case 33:case 34:case 35:  signtop++;  signstates[signtop]="<term>";  states.push(gotoTable[states.top()][term]);  //nodes.push(MakeNode(op, nodes));  break;  case 36: //非运算  tiaojian\_7();  signtop++;  signstates[signtop]="<unary>";  states.push(gotoTable[states.top()][unary]);  case 37:case 38:  signtop++;  signstates[signtop]="<unary>";  states.push(gotoTable[states.top()][unary]);  //nodes.push(MakeNode(op, nodes));  break;  case 39:case 40:case 41:case 42:case 43:case 44:  signtop++;  signstates[signtop]="<factor>";  states.push(gotoTable[states.top()][factor]);  //nodes.push(MakeNode(op, nodes));  break;  case 45:  signtop++;  signstates[signtop]="<M>";  tiaojian\_14();  states.push(gotoTable[states.top()][m\_zero]);  break;  case 46:  signtop++;  signstates[signtop]="<N>";  tiaojian\_15();  states.push(gotoTable[states.top()][n\_zero]);  //nodes.push(MakeNode(op, nodes));  break;  default:  printf("\n err %d",action);  break;  }  // token = NextToken();  }      else if (-action){  printf("\n 格式错误！！ %d",action);  return nodes.top();    }else{    printf("\n accept！！ %d \n",action);    return nodes.top();  }  printf(" 栈内容：");  for(int k=0;k<=signtop;k++){  std::cout<<(signstates[k]);  }  printf(" \n");    }  }  void tiaojian\_4() // S->i=E  { gen[nextpos][1]=stktemp.top();  stktemp.pop();  gen[nextpos][0]=stktemp.top();  stktemp.pop();  gen[nextpos][3]=stktemp.top();  stktemp.pop();  gen[nextpos][2].code=0;  nextpos++;  stknext.push(NULL);  }  void Parser::SetLine(std::string& l)  {  //lexer.SetLine(l);  }  // how do terminals map to indices: "!" -> 0; "!=" -> 1; "&&" -> 2; "(" -> 3; ")" -> 4; "\*" -> 5; "+" -> 6; "-" -> 7; "/" -> 8; ";" -> 9; "<" -> 10; "<=" -> 11; "=" -> 12; "==" -> 13; ">" -> 14; ">=" -> 15; "[" -> 16; "]" -> 17; "break" -> 18; "do" -> 19; "else" -> 20; "false" -> 21; "if" -> 22; "true" -> 23; "while" -> 24; "{" -> 25; "||" -> 26; "}" -> 27; basic -> 28; eof -> 29; id -> 30; num -> 31; real -> 32;  int Parser::Convert(const Token& t)  {  switch (t.type)  {  case NOT:  return 0;  case NOTEQUAL:  return 1;  case LOGICAL\_AND:  return 2;  case LPAREN:  return 3;  case RPAREN:  return 4;  case MULTIPLY:  return 5;  case PLUS:  return 6;  case MINUS:  return 7;  case DIVIDE:  return 8;  case SEMICOLON:  return 9;  case LESS:  return 10;  case LESSEQUAL:  return 11;  case ASSIGN:  return 12;  case EQUAL:  return 13;  case GREATER:  return 14;  case GREATEREQUAL:  return 15;  case LBRACKET:  return 16;  case RBRACKET:  return 17;  case BREAK:  return 18;  case DO:  return 19;  case ELSE:  return 20;  case FALSE:  return 21;  case IF:  return 22;  case TRUE:  return 23;  case WHILE:  return 24;  case LBRACE:  return 25;  case OR:  return 26;  case RBRACE:  return 27;  case BASIC:  return 28;  case ID:  return 30;  case END://important  return 29;  case NUM:  return 31;  case REAL:  return 32;  default:  return -1;  }  }  std::shared\_ptr<Node> Parser::MakeNode(std::stack<Token>& op, std::stack<std::shared\_ptr<Node>>& nodes)  {  Token t = op.top();  op.pop();  std::shared\_ptr<Node> right = nodes.top();  nodes.pop();  std::shared\_ptr<Node> left = nodes.top();  nodes.pop();  return std::make\_shared<Node>(t, left, right);  }  std::shared\_ptr<Node> Parser::MakeNode1(std::stack<std::shared\_ptr<Node>> & nodes)  {    std::shared\_ptr<Node> right = nodes.top();  nodes.pop();  Token t = nodes.top()->GetToken();  //printToken(t);  nodes.pop();  std::shared\_ptr<Node> left = nodes.top();  std::cout<<"四元式：("<<returntoken(t)<<" "<<left->ToString()<<" "<<"-"<<" "<<right->ToString()<<")\n";    nodes.pop();    gen[nextpos][1].value=right->ToString();  gen[nextpos][1].code= right->GetToken().code;      gen[nextpos][0].value=t.type;    gen[nextpos][3].value=left->ToString();    gen[nextpos][3].code= $ID;  gen[nextpos][0].code=$ASSIGN;  nextpos++;  stknext.push(NULL);  return std::make\_shared<Node>(t, left, right);  } |

Main.py

|  |
| --- |
| #define $INT 1 //int  #define $IF 2 //if  #define $ELSE 3 //else  #define $FLOAT 4 //float  #define $PRINT 5 //print  #define $ID 6 //identify  #define $CONSTANT 7 //constant  #define $ASSIGN 8 //=  #define $PLUS 9 //+  #define $STAR 10 //\*  #define $POWER 11 //\*\*  #define $SEMICOLON 12 //;  #define $LPAR 13 //(  #define $RPAR 14 //)  #define $LBRACE 15 //{  #define $RBRACE 16 //}  #define $LZKH 17 //[  #define $RZKH 18 //]  #define $SUBSIGN 19 //-  #define $DIVISION 20 // /  #define $COMMA 21 //,  #define $AND 22 //&&  #define $OR 23 //||  #define $NOT 24 //!  #define $EQUATION 25 //==  #define $ABOVE 26 //>  #define $BELOW 27 //<  #define $ABOVEE 28 //>=  #define $BELOWE 29 //<=  #define $NOTE 30 //!=  #define $TEMP 31  #define $NULL 0  #define $JMP 32  #include <cstdio>  #include <iostream>  #include "LR1Table.cpp"  #include "Parser.h"  #include "Parser.cpp"  #include <string>  #include <fstream>  #include "utilities/Node.h"  #include "utilities/Node.cpp"  #define $GOTO 33  #define $JL 34  using namespace std;  int main()  {  //initTokenTable();  initializeSymbolTable();  inputFile = fopen("test-2.txt", "r");  if (inputFile == NULL) {  error("Cannot open input file.");  }  Parser p;  std::shared\_ptr<Node> root = p.Parse();  //printf(">> %lf\n", root->Yield());  ofstream coutf;  coutf.open("genquad.txt");  if(!coutf)  { cout<<"Can not open genquad file !"<<endl;  return 0;  }  coutf<<'\t'<<"Op"<<'\t'<<"arg1"<<'\t'<<"arg2"<<'\t'<<"result"<<endl;  int i ;  for(i=1;i<nextpos;i++)  { coutf<<'('<<i<<')'<<'\t';  for(int j=0;j<4;j++){    switch (gen[i][j].code)    { case $JMP:coutf<<'j'<<'\t';break;  case $NULL:coutf<<'\t';break;  case $ID:coutf<<gen[i][j].value<<'\t';break;  // case $CONSTANT:coutf<<myconst[gen[i][j].value]<<'\t';break;  case $TEMP:coutf<<'t'<<gen[i][j].value<<'\t';break;  case $PLUS:coutf<<'+'<<'\t';break;  case $SUBSIGN:coutf<<"-"<<'\t';break;  case $ASSIGN:coutf<<'='<<'\t';break;  case $STAR:coutf<<'\*'<<'\t';break;  case $EQUATION:coutf<<'j'<<'='<<'='<<'\t';break;  case $ABOVE:coutf<<'j'<<'>'<<'\t';break;  case $BELOW:coutf<<'j'<<'<'<<'\t';break;  case $ABOVEE:coutf<<'j'<<'>'<<'='<<'\t';break;  case $BELOWE:coutf<<'j'<<'<'<<'='<<'\t';break;  case $NOTE:coutf<<'j'<<'!'<<'='<<'\t';break;  case $GOTO:coutf<<'('<<gen[i][j].value<<')'<<'\t';break;  }  coutf<<'|'<<'\t';  }coutf<<endl;  }  coutf<<'('<<i<<')'<<'\t';  coutf.close();  return 0;  } |

Lexer.cpp

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <ctype.h>  #include <string.h>  #define MAX\_LEXEME\_LENGTH 1000  #define MAX\_SYMBOLS 1000  void error(const char \*message) {  fprintf(stderr, "Error: %s\n", message);  //exit(1);  }  typedef enum {  KEY, // 关键字  IDENTIFIER, // 标识符  CONSTANT, // 常数  OPERATOR, // 运算符  DELIMITER // 分隔符  } SymbolType;  typedef struct {  SymbolType type;  char lexeme[MAX\_LEXEME\_LENGTH]; // 用于存储标识符或常数的字符串  int value;    } Symbol;  // 符号表项  typedef struct {  char lexeme[MAX\_LEXEME\_LENGTH]; // 标识符的名称  SymbolType type; // 标识符的类型  } SymbolTableEntry;  // 符号表  SymbolTableEntry symbolTable[MAX\_SYMBOLS];  int symbolTableSize = 0;  // 向符号表添加条目  void addToSymbolTable(char\* lexeme, SymbolType type) {  if (symbolTableSize >= MAX\_SYMBOLS) {  error("Symbol table is full.");  }  strcpy(symbolTable[symbolTableSize].lexeme, lexeme);  symbolTable[symbolTableSize].type = type;  symbolTableSize++;  }  // 在符号表中查找条目  //-std=c11 or -std=gnu11 to compile your code  SymbolTableEntry\* lookup(char\* lexeme) {  int i;  for (i = 0; i < MAX\_SYMBOLS; i++) {  if (strcmp(symbolTable[i].lexeme, lexeme) == 0) {  return &symbolTable[i];  }  }  return NULL; // 未找到匹配的条目  }  // 初始化符号表  void initializeSymbolTable() {  // 添加基本保留字  addToSymbolTable((char\*)"if", KEY);  addToSymbolTable((char\*)"else", KEY);  addToSymbolTable((char\*)"then", KEY);  addToSymbolTable((char\*)"while", KEY);  addToSymbolTable((char\*)"for", KEY);  addToSymbolTable((char\*)"do", KEY);  addToSymbolTable((char\*)"break", KEY);  addToSymbolTable((char\*)"true", KEY);  addToSymbolTable((char\*)"false", KEY);  addToSymbolTable((char\*)"num", KEY);  addToSymbolTable((char\*)"int", KEY);  addToSymbolTable((char\*)"real", KEY);  addToSymbolTable((char\*)"bool", KEY);  addToSymbolTable((char\*)"or", KEY);  addToSymbolTable((char\*)"and", KEY);  }  typedef enum {  // 关键字  IF, ELSE, WHILE,FOR, DO, BREAK, TRUE, FALSE,  // 标识符和常量  ID, NUM, REAL, STRING,BASIC,  // 运算符和分隔符  PLUS, MINUS,TIMES, MULTIPLY, DIVIDE, NOT, LOGICAL\_AND, LOGICAL\_OR, EQUAL, NOTEQUAL,  LESS, LESSEQUAL, GREATER, GREATEREQUAL, ASSIGN, SEMICOLON,  LPAREN, RPAREN, LBRACE, RBRACE, LBRACKET, RBRACKET,AND,OR,  // 其他  ERROR, ENDFILE, END  } TokenType;  typedef struct {  TokenType type;  union {  char \*string;  int integer;  double real;  char c;  } value;  int code;  int codevalue;  } Token;  Token currentToken;  FILE \*inputFile;  Token \*tokenTable; // Token 表  int tokenCount = 0; // Token 表中 Token 的数量  void initTokenTable(){  int initialCapacity = 10; // 初始容量  tokenTable = (Token\*)malloc(initialCapacity \* sizeof(Token));  tokenCount = 0;  }  void addTokenToTable(Token token) {  tokenTable = (Token\*)realloc(tokenTable, (tokenCount + 1) \* sizeof(Token));  tokenTable[tokenCount] = token;  tokenCount++;  }  void getToken() {  //getToken()跳过无效符号，获取单个Token，并加入到表中  int c;  //跳过无效符号(continue)，获取符号后return  while ((c = fgetc(inputFile)) != EOF) {  if (isspace(c)) {  continue; // 跳过空格和空白符号  } else if (c == '/') {  // 检查是否为注释  if ((c = fgetc(inputFile)) == '/') {  // 单行注释，忽略整行  while ((c = fgetc(inputFile)) != '\n' && c != EOF) {}  }else if ((c = fgetc(inputFile)) == '\*') {  // 多行注释  int prev\_c = 0,line=0;  while (1) {  c = fgetc(inputFile);  if (c == EOF) {  error("Unexpected end of file in comment.");  break;  } else if (prev\_c == '\*' && c == '/') {  break;  } else if (c == '\n') {  line++;  }  prev\_c = c;  }  }else {  ungetc(c, inputFile); // 不是注释，放回  break;  }  } else if (isalpha(c)) {  // 关键字或识别符号    // 关键字或识别符号  char buffer[256];  int length = 0;  buffer[length++] = (char) c;  while (isalnum(c = fgetc(inputFile))) {  buffer[length++] = (char) c;  }  buffer[length] = '\0';  ungetc(c, inputFile);  // 在符号表中查找关键字或标识符  SymbolTableEntry\* entry = lookup(buffer);  if (strcmp(buffer, "if") == 0) {  currentToken.type = IF;  } else if (strcmp(buffer, "else") == 0) {  currentToken.type = ELSE;  } else if (strcmp(buffer, "for") == 0) {  currentToken.type = FOR;  } else if (strcmp(buffer, "while") == 0) {  currentToken.type = WHILE;  } else if (strcmp(buffer, "do") == 0) {  currentToken.type = DO;  } else if (strcmp(buffer, "break") == 0) {  currentToken.type = BREAK;  } else if (strcmp(buffer, "true") == 0) {  currentToken.type = TRUE;  } else if (strcmp(buffer, "false") == 0) {  currentToken.type = FALSE;  } else if (strcmp(buffer, "int") == 0) {  currentToken.type = BASIC;  } else {  currentToken.type = ID;  currentToken.value.string = strdup(buffer);  }    currentToken.value.string = strdup(buffer);  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (isdigit(c)) {  // 整数或实数常量  char buffer[256];  int length = 0;  buffer[length++] = (char) c;  while (isdigit(c = fgetc(inputFile))) {  buffer[length++] = (char) c;  }  if (c == '.') {  buffer[length++] = '.'; // 记录小数点  while (isdigit(c = fgetc(inputFile))) {  buffer[length++] = (char) c;  }  buffer[length] = '\0';  currentToken.type = REAL;  currentToken.value.real = atof(buffer);  } else {  buffer[length] = '\0';  currentToken.type = NUM;  currentToken.value.integer = atoi(buffer);  }  ungetc(c, inputFile);  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '"') {  // 字符串常量  char buffer[256];  int length = 0;  while ((c = fgetc(inputFile)) != '"') {  if (c == EOF) {  error("Unexpected end of file inside string constant.");  }  buffer[length++] = (char) c;  }  buffer[length] = '\0';  currentToken.type = STRING;  currentToken.value.string = strdup(buffer);  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '=') {  // 等号或等于运算符  if ((c = fgetc(inputFile)) == '=') {  currentToken.type = EQUAL;  } else {  ungetc(c, inputFile);  currentToken.type = ASSIGN;  }  currentToken.value.c=c;    addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '!') {  // 不等于运算符  if ((c = fgetc(inputFile)) == '=') {  currentToken.type = NOTEQUAL;  } else {  ungetc(c, inputFile);  currentToken.type = NOT;  }  currentToken.value.c=c;    addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '<') {  // 小于或小于等于运算符  if ((c = fgetc(inputFile)) == '=') {  currentToken.type = LESSEQUAL;  } else {  ungetc(c, inputFile);  currentToken.type = LESS;  }  currentToken.value.c=c;    addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '>') {  // 大于或大于等于运算符  if ((c = fgetc(inputFile)) == '=') {  currentToken.type = GREATEREQUAL;  } else {  ungetc(c, inputFile);  currentToken.type = GREATER;  }  currentToken.value.c=c;    addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '&') {  // 逻辑与运算符  if ((c = fgetc(inputFile)) == '&') {  currentToken.type =LOGICAL\_AND;  } else {  error("Invalid character '&'.");  }  currentToken.value.c=c;    addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '|') {  // 逻辑或运算符  if ((c = fgetc(inputFile)) == '|') {  currentToken.type = LOGICAL\_OR;  } else {  error("Invalid character '|'.");  }  currentToken.value.c=c;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '+') {  // 加法运算符  currentToken.value.c=c;    currentToken.type = PLUS;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '-') {  // 减法运算符  currentToken.value.c=c;    currentToken.type = MINUS;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '\*') {  // 乘法运算符  currentToken.value.c=c;    currentToken.type = MULTIPLY;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '/') {  // 除法运算符  currentToken.value.c=c;    currentToken.type = DIVIDE;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '(') {  // 左括号  currentToken.value.c=c;    currentToken.type = LPAREN;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == ')') {  // 右括号  currentToken.value.c=c;    currentToken.type = RPAREN;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  } else if (c == '{') {  currentToken.value.c=c;    currentToken.type = LBRACE;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  }else if (c == '}') {  currentToken.value.c=c;    currentToken.type = RBRACE;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  }else if (c == ';') {  currentToken.value.c=c;    currentToken.type = SEMICOLON;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  return;  }else {  error("Invalid character");  }  }  // 到达文件末尾  currentToken.type = END;  addTokenToTable(currentToken); // 将 Token 加入 Token 表  }  std::string returntoken(Token token) {  switch (token.type) {  case BASIC:  return(token.value.string);  break;  case IF:  return("<if>");  break;  case ELSE:  return(token.value.string);  break;  case FOR:  return(token.value.string);  break;  case WHILE:  return(token.value.string);  break;  case DO:  return(token.value.string);  break;  case BREAK:  return(token.value.string);  case TRUE:  return(token.value.string);  case FALSE:  return(token.value.string);  case ID:  return(token.value.string);  break;  case NUM:  return("<num>");  case REAL:  return("<num>");  case PLUS:  return("<+>");    case MINUS:  return("<->");    case MULTIPLY:  return("<\*>");  case DIVIDE:  return("</>");  case NOT:  return("<not>");  case AND:  return("<&&, op>");  break;  case OR:  return("<||, op>");  break;  case EQUAL:  return("<==, relop>");  break;  case NOTEQUAL:  return("<!=, relop>");  break;  case LESS:  return("< <, relop>");  break;  case LESSEQUAL:  return("< <=, relop>");  break;  case GREATER:  return("< >, relop>");  break;  case GREATEREQUAL:  return("< >=, relop>");  break;  case ASSIGN:  return("<=>");  break;  case SEMICOLON:  return("<;, 界符>");  break;  case LPAREN:  return("<(, 界符>");  break;  case RPAREN:  return("<), 界符>");  break;  case LBRACE:  return("<{, 界符>");  break;  case RBRACE:  return("<}, 界符>");  break;  case LBRACKET:  return("<[, 界符>");  break;  case RBRACKET:  return("<], 界符>");  break;  case ERROR:  return("<ERROR>\n");  break;  case ENDFILE:  return("<EOF>\n");  break;  }  return ("<ERROR>");  }  void printToken(Token token) {  switch (token.type) {  case BASIC:  printf("<%s, key>\n", token.value.string);  break;  case IF:  printf("<%s, key>\n", token.value.string);  break;  case ELSE:  printf("<%s, key>\n", token.value.string);  break;  case FOR:  printf("<%s, key>\n", token.value.string);  break;  case WHILE:  printf("<%s, key>\n", token.value.string);  break;  case DO:  printf("<%s, key>\n", token.value.string);  break;  case BREAK:  printf("<%s, key>\n", token.value.string);  break;  case TRUE:  printf("<%s, key>\n", token.value.string);  break;  case FALSE:  printf("<%s, key>\n", token.value.string);  break;  case ID:  printf("<%s, id>\n", token.value.string);  break;  case NUM:  printf("<%d, num>\n", token.value.integer);  break;  case REAL:  printf("<%f, num>\n", token.value.real);  break;  case PLUS:    printf("<%c, op>\n", token.value.c);  break;  case MINUS:  printf("<%c, op>\n", token.value.c);  break;    case MULTIPLY:  printf("<%c, op>\n", token.value.c);  break;  case DIVIDE:  printf("<%c, op>\n", token.value.c);  break;  case NOT:  printf("<!, op>\n");  break;  case AND:  printf("<&&, op>\n");  break;  case OR:  printf("<||, op>\n");  break;  case EQUAL:  printf("<==, relop>\n");  break;  case NOTEQUAL:  printf("<!=, relop>\n");  break;  case LESS:  printf("< <, relop>\n");  break;  case LESSEQUAL:  printf("< <=, relop>\n");  break;  case GREATER:  printf("< >, relop>\n");  break;  case GREATEREQUAL:  printf("< >=, relop>\n");  break;  case ASSIGN:  printf("<=, 赋值>\n");  break;  case SEMICOLON:  printf("<;, 界符>\n");  break;  case LPAREN:  printf("<(, 界符>\n");  break;  case RPAREN:  printf("<), 界符>\n");  break;  case LBRACE:  printf("<{, 界符>\n");  break;  case RBRACE:  printf("<}, 界符>\n");  break;  case LBRACKET:  printf("<[, 界符>\n");  break;  case RBRACKET:  printf("<], 界符>\n");  break;  case ERROR:  printf("<ERROR>\n");  break;  case ENDFILE:  printf("<EOF>\n");  break;  }  }  void token2int(Token token) {  switch (token.type) {  case BASIC:  printf("<%s, key>\n", token.value.string);  break;  case IF:  printf("<%s, key>\n", token.value.string);  break;  case ELSE:  printf("<%s, key>\n", token.value.string);  break;  case FOR:  printf("<%s, key>\n", token.value.string);  break;  case WHILE:  printf("<%s, key>\n", token.value.string);  break;  case DO:  printf("<%s, key>\n", token.value.string);  break;  case BREAK:  printf("<%s, key>\n", token.value.string);  break;  case TRUE:  printf("<%s, key>\n", token.value.string);  break;  case FALSE:  printf("<%s, key>\n", token.value.string);  break;  case ID:  printf("<%s, id>\n", token.value.string);  break;  case NUM:  printf("<%d, num>\n", token.value.integer);  break;  case REAL:  printf("<%f, num>\n", token.value.real);  break;  case PLUS:    printf("<%c, op>\n", token.value.c);  break;  case MINUS:  printf("<%c, op>\n", token.value.c);  break;    case MULTIPLY:  printf("<%c, op>\n", token.value.c);  break;  case DIVIDE:  printf("<%c, op>\n", token.value.c);  break;  case NOT:  printf("<!, op>\n");  break;  case AND:  printf("<&&, op>\n");  break;  case OR:  printf("<||, op>\n");  break;  case EQUAL:  printf("<==, relop>\n");  break;  case NOTEQUAL:  printf("<!=, relop>\n");  break;  case LESS:  printf("< <, relop>\n");  break;  case LESSEQUAL:  printf("< <=, relop>\n");  break;  case GREATER:  printf("< >, relop>\n");  break;  case GREATEREQUAL:  printf("< >=, relop>\n");  break;  case ASSIGN:  printf("<=, 赋值>\n");  break;  case SEMICOLON:  printf("<;, 界符>\n");  break;  case LPAREN:  printf("<(, 界符>\n");  break;  case RPAREN:  printf("<), 界符>\n");  break;  case LBRACE:  printf("<{, 界符>\n");  break;  case RBRACE:  printf("<}, 界符>\n");  break;  case LBRACKET:  printf("<[, 界符>\n");  break;  case RBRACKET:  printf("<], 界符>\n");  break;  case ERROR:  printf("<ERROR>\n");  break;  case ENDFILE:  printf("<EOF>\n");  break;  }  }  Token NextToken()  {  getToken();    return currentToken;  } |

Node.cpp

|  |
| --- |
| #include "Node.h"  #include <cmath>  Node::Node()  {  t = {};  left = {};  right = {};  }  Node::Node(const Token& token)  {  t = token;  left = {};  right = {};  }  Node::Node(const Token& token, std::shared\_ptr<Node> l, std::shared\_ptr<Node> r)  {  t = token;  left = l;  right = r;  }  const Token& Node::GetToken()  {  return t;  }  std::shared\_ptr<Node> Node::GetLeft()  {  return left;  }  std::shared\_ptr<Node> Node::GetRight()  {  return right;  }  void Node::SetLeft(std::shared\_ptr<Node> l)  {  left = l;  }  void Node::SetRight(std::shared\_ptr<Node> r)  {  right = r;  }  std::string Node::ToString()  {  if (!left || !right)  {  if(t.code==$TEMP){  return std::to\_string(t.codevalue);  }  if (t.type==NUM){return std::to\_string(t.value.integer);}  return t.value.string;  }  if(t.code==$TEMP){  return std::to\_string(t.codevalue);  }  return '(' + left->ToString() + ' ' + t.value.string + ' ' + right->ToString() + ')';  } |

LR1\_table.py

|  |
| --- |
| from LR1\_items import \*  from SLR\_table import CStyleTable  from leftrec import \*  from sets import \*  if \_\_name\_\_ == '\_\_main\_\_':  nts = ParseBNF('1.txt').Build()  # nts = EliminateLeftRecursion(nts)  augmentedSymbol = '<program\'>'  init = LRState()  init += Item1('<program\'>', ['<program>',], 0, 'eof')  C, transition = GetStates(init, nts, '<program>', closureLR1)  # nts = ParseBNF('grammar.txt').Build()  # augmentedSymbol = '<expr\'>'  # init = LRState()  # init += Item1('<expr\'>', ['<expr>',], 0, 'eof')  # C, transition = GetStates(init, nts, '<expr>', closureLR1)  f=open("output.txt","w",encoding="utf-8-sig")    f1=open("ltems.txt","w",encoding="utf-8-sig")  actionTable = dict()  gotoTable = dict()    #print(closureLR1,file=f1)  for state in C:  name = state.name  #print(name)    for item in state:    if item.name == augmentedSymbol and (not item.Where()):  actionTable[(name, 'eof')] = 'Accept!'  else:  next2Dot = item.Where()    if not next2Dot:  actionTable[(name, item.lookahead)] = item # an item to reduce，点号后面没东西，归约  elif next2Dot and next2Dot[0] != '<': #点号后面有东西且为终结符  to = transition[(name, next2Dot)]  actionTable[(name, next2Dot)] = to # shift to some state    for state in C:  print(state,file=f1)  for symbolName in nts.GetName():  key = (state.name, symbolName)  if key in transition.keys():  gotoTable[key] = transition[key]    print('ACTION TABLE:',file=f)  for key, value in actionTable.items():  #(3, '""') key  print(key, ': ', sep=' ', end = '',file=f)  if isinstance(value, Item1):  print('reduce ',value.Index, ",",str(value),file=f)  elif value == 'Accept!':  print(value,file=f)  else:  print("shift ", value, sep = ' ',file=f)    print('----------------------------------',file=f)  print('GOTO TABLE',file=f)  for key, value in gotoTable.items():  print(key, ': ', value, sep=' ',file=f)  print()  print(CStyleTable('actionTable', actionTable, True),file=f)  print(CStyleTable('gotoTable', gotoTable, False),file=f)  f.close()  f1.close() |