|  |
| --- |
| 编译原理课程设计  实验报告  **指导教师：张鹏**  **年 级： 2019级**  **班 级： 23班**  **小组编号： 第九组**  **组长学号姓名：21190130 王晓衡**  **组员学号姓名：21190331 熊记**  **组员学号姓名：21190329田堰泽**  **2022年 4 月 27 日**  **计算机科学与技术学院** |

|  |  |  |
| --- | --- | --- |
| **完成实验内容** | | |
| 设计并实现SNL程序设计语言的编译程序：  一.词法分析模块  二.语法分析模块（递归下降方法）  三.语法分析模块（LL(1)方法）  四.语义分析模块 | | |
| **小组成员任务完成情况** | | |
| 姓名 | 具体完成任务 | 工作量  百分比 |
| 王晓衡 | 确定框架，词法分析(递归下降)，协调代码 | 33% |
| 熊记 | 词法分析，语法分析(LL1), 前端界面，协调代码 | 33% |
| 田堰泽 | 语义分析，协调代码 | 33% |
| **小组成员协作情况** | | |
| 首先，确定方案和框架，然后分配给每人任务各自完成，最后整合代码，测试用例，逐步调试得到现在的结果。 | | |
| **实验平台与编程语言** | | |
| **实验平台： visual sutdio 2022**  **编程语言： c++, c#** | | |
| **实验方案设计** | | |
| 本实验我们小组实现了SNL语言编译器的前三个部分：词法分析，语法分析和语义分析，其中语法分析分别用LL1分析方法和递归下降两种方式来实现。  一.词法分析模块  构造token结构体，包含三个成员：token所在行数（Line），词法信息（Lex），语义信息（Sem），创建保留字集合，以便比对是保留字还是标识符。  流程图：  二.语法分析模块（递归下降方法）  ①所用数据结构：语法树，语法树表示为终极符或者非终极符的节点组成。  每个节点的数据包括：  name：表示终极符或非终极符的名字  token:一个指针,存储某个终极符节点的Token信息  child: 存储该节点子节点指针数组。    ②流程图      三.语法分析模块（LL(1)方法）  创建LL1分析栈  流程图：    四.语义分析模块  数据结构：  语义分析模块涉及的数据结构主要通过遍历语法树获得符号表。  符号表的数据结构大体参考了教材，具体如下：  SymbolTable：{  Symbolid：符号表项的id  Atribute：符号表的属性  Next：下一项  }  Atribute：{  Idtype(TypeIR)：表示标识符的类型  Kind：标识符的种类  VarOrProc：包含了函数标识符或者变量标识符的属性  }  TypeIR：{  Int：size;  Ty：表示类型的种类  Array：数组类型的属性  Body：域类型的头指针  }  流程图：  五.前端部分  包含文本编辑框（可直接在框内编写源代码）、文件新建（清空文本框内容）、文件打开（将目标文件源代码导入文本编辑框中）、文件保存（可将文本编辑框中内容保存至txt文件中）、编译按钮（执行编译程序，编译文本框中源代码）、查看按钮（可在第二个文本框中显示各个阶段的执行结果）。 | | |
| **程序界面及运行截图** | | |
|  | | |
| **源程序核心代码** | | |
| 词法分析代码：  if (letter(c)) //以字母打头，不是保留字就是标识符  {  string temp;  while (i < s.size() && (letter(c) || number(c)))  {  temp += c;  i++;  c = s[i];  }  i--;  if (reserved.find(temp) != reserved.end())  {  transform(temp.begin(), temp.end(), temp.begin(), ::toupper);  t->Lex = temp;  }  else  {  t->Lex = "ID";  t->Sem = temp;  }  token.push\_back(\*t);  continue;  }  ……………….  递归下降代码：  TreeNode\* Program()  {  if (now\_token->Lex == "PROGRAM")  {  TreeNode\* root = new TreeNode("Program");  root->add\_node(ProgramHead());  root->add\_node(DeclarePart());  root->add\_node(ProgramBody());  root->add\_node(match\_token("DOT"));  return root;  }  else syntax\_error("PROGRAM");  return NULL;  }  LL1代码：  TreeNode\* LL1() {  ana->push(0, Program);  //cout << ana->top->n << endl;  TreeNode\* root = new TreeNode("Program");  ana->top->treenode = root;  int index = 0;  initLL1Table();  int count=0;  //cout << "进入循环" << endl;  while (ana->top != NULL) {  Token temp\_token = token[index];  if (ana->top->ntflag == 0) {  if (LL1Table[ana->top->n][string2enum(temp\_token.Lex)] == 0) {  cout << "错误的产生式的非终极符:" << ana->top->n << " 终极符：" << temp\_token.Lex << endl;  ll1\_file.push\_back("错误的产生式的非终极符:"+ana->top->n+temp\_token.Lex + "\n");  return NULL;  }  process(LL1Table[ana->top->n][string2enum(temp\_token.Lex)]);  }  else {  if (ana->top->t == string2enum(temp\_token.Lex)) {  cout << "第" << temp\_token.Line << "行的" << temp\_token.Lex << "匹配成功,弹栈" << endl;  ll1\_file.push\_back("第"+ to\_string(temp\_token.Line)+"行的"+ temp\_token.Lex+ "匹配成功,弹栈"+"\n");  ana->top->treenode->token = &token[index];  ana->pop();  }  else {  cout << "第" << temp\_token.Line << "行的" << temp\_token.Lex <<"和"<< terminals\_string[ana->top->t] << "不匹配" << endl;  ll1\_file.push\_back("第"+to\_string(temp\_token.Line)+"行的"+ temp\_token.Lex + terminals\_string[ana->top->t] +"不匹配"+"\n");  return NULL;  }  index++;  }  count++;  }  if (token[index].Lex == "EOF") {  cout <<"栈空，当前token为EOF，LL1分析完成，无语法错误" << endl;  ll1\_file.push\_back("栈空，当前token为EOF，LL1分析完成，无语法错误\n");  }  else {  cout << "token未完，当前token为："<<token[index].Lex << endl;  ll1\_file.push\_back("token未完，当前token为："+ token[index].Lex+"\n");  return NULL;  }  return root;  }  语义分析代码：  void SematicAnalyze(TreeNode\* Program) {  TreeNode\* t = Program;  CreateTable();  // Initialize();  DeclarePart(t->child[1]);  ProgramBody(t->child[2]);  printf("语义分析完成\n");  if (error\_flag == 0)  {  yuyi.push\_back("语义分析完成\n");  stringstream ss;  ss << yuyi[yuyi.size() - 1];  outfile\_yuyi << ss.str();  }    DropTable();  } | | |
|  | | |