**计算机科学与工程学院 实验报告**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **实验课程名称** | | **编译原理** | | |
| **专业** | **计算机科学与技术** | | **班级** | **2206** |
| **学号** | **20225868** | | **姓名** | **李昕鸿** |
| **实验项目目录**   1. 实验一 词法分析 2. 实验二 算术表达式语法分析 3. 实验（选做）。。。 | | | | |
| **实验报告正文**  **实验一 词法分析**  **实验题目**  熟悉并实现一个简单的扫描器，设计扫描器的自动机；设计翻译、生成Token的算法；编写代码并上机调试运行通过。要求扫描器可识别的单词包括：关键字、界符、标识符和常整形数。  **实验目的**  通过设计、编制并调试一个具体的词法分析程序，加深对词法分析原理的理解。掌握在对程序设计语言源程序进行扫描过程中，将其分解为各类单词的词法分析方法。理解词法分析在编译程序中的作用；加深对有穷自动机模型的理解；掌握词法分析程序的实现方法和技术。  **实验原理**  扫描器设计  **实验内容（基础点、扩展点、流程图、源码、注释）**  **基础点**  流程图    源码  #include<bits/stdc++.h>  using namespace **std**;  int k = 0;  int p = 0;  int i = 0;  int c = 0;  map<string, int>K, P, I, C1,C2,CT,ST;  string s;int position=0;  void **pre**()  {      K["int"] = ++k;      K["void"] = ++k;      K["break"] = ++k;      K["float"] = ++k;      K["while"] = ++k;      K["do"] = ++k;      K["struct"] = ++k;      K["const"] = ++k;      K["case"] = ++k;      K["for"] = ++k;      K["return"] = ++k;      K["if"] = ++k;      K["default"] = ++k;      K["else"] = ++k;      P["-"] = ++p;      P["/"] = ++p;      P["("] = ++p;      P[")"] = ++p;      P["=="] = ++p;      P["<="] = ++p;      P["<"] = ++p;      P["+"] = ++p;      P["\*"] = ++p;      P[">"] = ++p;      P["="] = ++p;      P[","] = ++p;      P[";"] = ++p;      P["++"] = ++p;      P["{"] = ++p;      P["}"] = ++p;  }  char ch;  bool **IsLetter**(char ch) *//英文字母*  {      return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'z');  }  bool **IsNumber**(char ch) *//数字*  {      return ch >= '0' && ch <= '9';  }  int  **ReadNumber**(**string**&token) *//识别算数常数*  {      int status = 1;      while (1) {          if (ch == '#') {              if (status == 2) return 0;              else if (status == 4 || status == 7) return 1;              else return -1;          }          if (status == 1) {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];                  status = 2;              }              else return -1;          }          else if (status == 2) {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];              }              else if (ch == '.') {                  token += ch, ch = s[position++];                  status = 3;              }              else if (ch == 'e') {                  token += ch, ch = s[position++];                  status = 5;              }              else if (ch == 'x') {                  if (token.**size**() == 1) {                      token += ch, ch = s[position++];                      status = 8;                  }                  else return -1;              }              else return 0;          }          else if (status == 3) {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];                  status = 4;              }              else return -1;          }          else if (status == 4) {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];              }              else if (ch == 'e') {                  token += ch, ch = s[position++];                  status = 5;              }              else return 1;          }          else if (status == 5) {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];                  status = 7;              }              else if (ch == '+' || ch == '-') {                  token += ch, ch = s[position++];                  status = 6;              }              else return -1;          }          else if (status == 6) {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];                  status = 7;              }              else return -1;          }          else if (status == 7) {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];              }              else return 1;          }          else if (status == 8) {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];                  status = 9;              }              else return -1;          }          else if (status == 9) {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];              }              else if (ch >= 'A' && ch <= 'F') {                  token += ch, ch = s[position++];              }              else if (ch >= 'a' && ch <= 'f') {                  token += ch, ch = s[position++];              }              else {                  long long a = 0;                  for (int i = 2; i < (int)token.**size**(); i++)                      if (**IsNumber**(token[i])) a = a \* 16 + token[i] - '0';                      else if (token[i] >= 'A' && token[i] <= 'Z') a = a \* 16 + 10 + token[i] - 'A';                      else a = a \* 16 + 10 + token[i] - 'a';                  token = **to\_string**(a);                  return 0;              }          }      }    }  int **ReadToken**(**string**& token) {      token.**clear**();      while (ch != '#' && ch == ' ') ch = s[position++];      if (ch == '#') return 0;      if (**IsLetter**(ch)) { *// 关键字、标识符*          while (**IsLetter**(ch) || **IsNumber**(ch)) {              token += ch, ch = s[position++];              if (ch == '#') return 1;          }          return 1;      }      else if (**IsNumber**(ch)) { *// 数字*          int t = **ReadNumber**(token);          if (t == -1) return -1;          return 2 + t;      }      else if (ch == '\'') { *// 字符常量*          ch = s[position++];          if (ch == '#') return -1;          token += ch, ch = s[position++];          if (ch != '\'') return -1;          ch = s[position++];          return 4;      }      else if (ch == '\"') { *// 字符串常量*          ch = s[position++];          while (ch != '\"') {              if (ch == '#') return -1;              token += ch, ch = s[position++];          }          ch = s[position++];          return 5;      }      else { *// 界符*          token += ch, ch = s[position++];          if (ch == '#') return 6;          if (P[token+ch])              token += ch, ch = s[position++];          return 6;      }  }  string a = "";  vector<string>i1, c1, c2, ct, st;  int c02, ct1, st1;  void **first**()  {      ostringstream out;  **pre**();      int t;  **getline**(cin, s);      s += '#';      string token;      out << "Token :";        while (t=**ReadToken**(token))      {          if (token.**size**() == 0)break;          if (t == -1)          {              cout << "ERROR" << endl;              return;          }            if (t == 1) *//关键字，标识符*          {              if (K[token] != 0)*//出现过的关键字*                  out << "(K " << K[token] << ")";              else if (I[token] != 0)*//出现过的标识符*                  out << "(I " << I[token] << ")";              else *//未出现的标识符*              {                  I[token] = ++i;                  i1.**push\_back**(token);                  out << "(I " << I[token] << ")";              }          }          else if (t == 2)*//整数*          {              if (C1[token] != 0)*//出现过的整数*                  out << "(C " << C1[token] << ")";              else *//未出现的整数*              {                  C1[token] = ++c;                  c1.**push\_back**(token);                  out << "(C " << C1[token] << ")";              }          }          else if (t == 3)*//浮点数*          {              if (C2[token] != 0)*//出现过的浮点数*                  out << "(C2 " << C2[token] << ")";              else              {                  C2[token] = ++c02;                  c2.**push\_back**(token);                  out << "(C2 " << C2[token] << ")";              }          }          else if (t == 4)*//字符*          {              if (CT[token] != 0)*//出现过*                  out << "(CT " << CT[token] << ")";              else              {                  CT[token] = ++ct1;                  ct.**push\_back**(token);                  out << "(CT " << CT[token] << ")";              }          }          else if (t == 5)          {              if (ST[token] != 0)*//出现过*                  out << "(ST " << ST[token] << ")";              else              {                  ST[token] = ++st1;                  st.**push\_back**(token);                  out << "(ST " << ST[token] << ")";              }          }          else if(P[token])*//界符*              out << "(P " << P[token] << ")";          if (ch == '#')break;      }      out << "\nI :";      for(int j=0;j<i1.**size**();j++)      {        out<<i1[j];        if (j!=i1.**size**()-1)        out <<" ";      }        out << "\nC :";      for(int j=0;j<c1.**size**();j++)      {        out<<c1[j];        if(j!=c1.**size**()-1) out<<" ";      }      cout << out.**str**();  }  int **main**()  {  **first**();  }  **实验结果（截图）**    **扩展点**  流程图    源码  #include<bits/stdc++.h>  using namespace **std**;  int k = 0;  int p = 0;  int i = 0;  int c = 0;  map<string, int>K, P, I, C1,C2,CT,ST;  string s;int position=0;  void **pre**()  {      K["int"] = ++k;      K["void"] = ++k;      K["break"] = ++k;      K["float"] = ++k;      K["while"] = ++k;      K["do"] = ++k;      K["struct"] = ++k;      K["const"] = ++k;      K["case"] = ++k;      K["for"] = ++k;      K["return"] = ++k;      K["if"] = ++k;      K["default"] = ++k;      K["else"] = ++k;      P["-"] = ++p;      P["/"] = ++p;      P["("] = ++p;      P[")"] = ++p;      P["=="] = ++p;      P["<="] = ++p;      P["<"] = ++p;      P["+"] = ++p;      P["\*"] = ++p;      P[">"] = ++p;      P["="] = ++p;      P[","] = ++p;      P[";"] = ++p;      P["++"] = ++p;      P["{"] = ++p;      P["}"] = ++p;  }  char ch;  bool **IsLetter**(char ch) *//英文字母*  {      return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'z');  }  bool **IsNumber**(char ch) *//数字*  {      return ch >= '0' && ch <= '9';  }  int  **ReadNumber**(**string**&token) *//识别算数常数*  {      int status = 1;      while (1) {          if (ch == '#')          {              if (status == 2) return 0;              else if (status == 4 || status == 7) return 1;              else return -1;          }          if (status == 1)          {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];                  status = 2;              }              else return -1;          }          else if (status == 2)          {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];              }              else if (ch == '.') {                  token += ch, ch = s[position++];                  status = 3;              }              else if (ch == 'e') {                  token += ch, ch = s[position++];                  status = 5;              }              else if (ch == 'x') {                  if (token.**size**() == 1) {                      token += ch, ch = s[position++];                      status = 8;                  }                  else return -1;              }              else return 0;          }          else if (status == 3)          {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];                  status = 4;              }              else return -1;          }          else if (status == 4)          {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];              }              else if (ch == 'e') {                  token += ch, ch = s[position++];                  status = 5;              }              else return 1;          }          else if (status == 5)          {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];                  status = 7;              }              else if (ch == '+' || ch == '-') {                  token += ch, ch = s[position++];                  status = 6;              }              else return -1;          }          else if (status == 6)           {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];                  status = 7;              }              else return -1;          }          else if (status == 7)          {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];              }              else return 1;          }          else if (status == 8)          {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];                  status = 9;              }              else return -1;          }          else if (status == 9)          {              if (**IsNumber**(ch)) {                  token += ch, ch = s[position++];              }              else if (ch >= 'A' && ch <= 'F') {                  token += ch, ch = s[position++];              }              else if (ch >= 'a' && ch <= 'f') {                  token += ch, ch = s[position++];              }              else {                  long long a = 0;                  for (int i = 2; i < (int)token.**size**(); i++)                      if (**IsNumber**(token[i])) a = a \* 16 + token[i] - '0';                      else if (token[i] >= 'A' && token[i] <= 'Z') a = a \* 16 + 10 + token[i] - 'A';                      else a = a \* 16 + 10 + token[i] - 'a';                  token = **to\_string**(a);                  return 0;              }          }      }  }  int **ReadToken**(**string**& token) {      token.**clear**();      while (ch != '#' && ch == ' ') ch = s[position++];      if (ch == '#') return 0;      if (**IsLetter**(ch)) { *// 关键字、标识符*          while (**IsLetter**(ch) || **IsNumber**(ch)) {              token += ch, ch = s[position++];              if (ch == '#') return 1;          }          return 1;      }      else if (**IsNumber**(ch)) { *// 数字*          int t = **ReadNumber**(token);          if (t == -1) return -1;          return 2 + t;      }      else if (ch == '\'') { *// 字符常量*          ch = s[position++];          if (ch == '#') return -1;          token += ch, ch = s[position++];          if (ch != '\'') return -1;          ch = s[position++];          return 4;      }      else if (ch == '\"') { *// 字符串常量*          ch = s[position++];          while (ch != '\"') {              if (ch == '#') return -1;              token += ch, ch = s[position++];          }          ch = s[position++];          return 5;      }      else { *// 界符*          token += ch, ch = s[position++];          if (ch == '#') return 6;          if (P[token+ch])              token += ch, ch = s[position++];          return 6;      }  }  string a = "";  vector<string>i1, c1, c2, ct, st;  int c02, ct1, st1;  void **first**()  {      ostringstream out;  **pre**();      int t;  **getline**(cin, s);      s += '#';      string token;      out << "Token :";        while (t=**ReadToken**(token))      {          if (token.**size**() == 0)break;          if (t == -1)          {              cout << "ERROR" << endl;              return;          }            if (t == 1) *//关键字，标识符*          {              if (K[token] != 0)*//出现过的关键字*                  out << "(K " << K[token] << ")";              else if (I[token] != 0)*//出现过的标识符*                  out << "(I " << I[token] << ")";              else *//未出现的标识符*              {                  I[token] = ++i;                  i1.**push\_back**(token);                  out << "(I " << I[token] << ")";              }          }          else if (t == 2)*//整数*          {              if (C1[token] != 0)*//出现过的整数*                  out << "(C1 " << C1[token] << ")";              else *//未出现的整数*              {                  C1[token] = ++c;                  c1.**push\_back**(token);                  out << "(C1 " << C1[token] << ")";              }          }          else if (t == 3)*//浮点数*          {              if (C2[token] != 0)*//出现过的浮点数*                  out << "(C2 " << C2[token] << ")";              else              {                  C2[token] = ++c02;                  c2.**push\_back**(token);                  out << "(C2 " << C2[token] << ")";              }          }          else if (t == 4)*//字符*          {              if (CT[token] != 0)*//已存在*                  out << "(CT " << CT[token] << ")";              else              {                  CT[token] = ++ct1;                  ct.**push\_back**(token);                  out << "(CT " << CT[token] << ")";              }          }          else if (t == 5)          {              if (ST[token] != 0)*//已存在*                  out << "(ST " << ST[token] << ")";              else              {                  ST[token] = ++st1;                  st.**push\_back**(token);                  out << "(ST " << ST[token] << ")";              }          }          else if(P[token])*//界符*              out << "(P " << P[token] << ")";          if (ch == '#')break;      }      out << "\nI :";      for(int j=0;j<i1.**size**();j++)      {        out<<i1[j];        if (j!=i1.**size**()-1)        out <<" ";      }        out << "\nC1 :";      for(int j=0;j<c1.**size**();j++)      {        out<<c1[j];        if(j!=c1.**size**()-1) out<<" ";      }      out << "\nC2 :";      for(int j=0;j<c2.**size**();j++)      {        out<<c2[j];        if(j!=c2.**size**()-1) out <<" ";      }      out << "\nCT :";      for(int j=0;j<ct.**size**();j++)      {        out<<ct[j];        if(j!=ct.**size**()-1) out<<" ";      }        out << "\nST :";      for(int j=0;j<st.**size**();j++)      {        out<<st[j];        if(j!=st.**size**()-1) out<<" ";      }      cout << out.**str**();  }  int **main**()  {  **first**();  }  **实验结果（截图）**    **实验二 算术表达式语法分析**  **实验题目**  1．设计简单算数表达式语法分析器算法  2．编写代码并上机调试运行通过。  **实验目的**  根据算符优先分析法，对表达式进行语法分析，使其能够判断一个表达式是否正确。通过算符优先分析方法的实现，加深对自下而上语法分析方法的理解。  **实验原理**  LL(1)分析 递归下降分析法  **实验内容（基础点、扩展点、流程图、源码、注释）**  **LL(1)**  流程图    源码  #include<iostream>  #include<string.h>  #include<stdlib.h>  using namespace **std**;  char str[100];  int indexx = 0;  void **LL1**();  void **AS**();  void **T**();  void **MD**();  void **Judge**();  int **main**()  {      int len;      cin**>>**str;      len = **strlen**(str);      str[len] = '\0';  **LL1**();      if (str[indexx] == '\0')          cout**<<**"True\n";      else {          cout**<<**"False\n";      }  **strcpy**(str, "");      indexx = 0;      return 0;  }  void **LL1**() {  **T**();  **AS**();  }  void **AS**() {      if (str[indexx]=='+' || str[indexx]=='-') {          indexx++;  **T**();  **AS**();      }  }  void **T**() {  **Judge**();  **MD**();  }  void **MD**() {      if (str[indexx]=='\*' || str[indexx]=='/')      {          indexx++;  **Judge**();  **MD**();      }  }  void **Judge**() {      if ((str[indexx]>='a' && str[indexx]<='z')||(str[indexx]>='0' && str[indexx]<='9'))      {          indexx++;      } else if (str[indexx] == '(')      {          indexx++;  **LL1**();          if (str[indexx] == ')')          {              indexx++;          } else          {              cout**<<**"False\n";  **exit**(0);          }      } else {          cout**<<**"False\n";  **exit**(0);      }  }  实验结果（截图）    **递归下降分析**  流程图    源码  #include <iostream>  #include<cstring>  #include<string>  #include <vector>  using namespace **std**;    enum **type** { digit, Hh, AF, letter, oparetors };    char\* q;*//指向输入符号串中当前的字符*  char word[20];*//存储当前识别的单词*  int state = 0;*//表示所处的状态,初始状态为0*  int i;*//单词的下标*  int j = 0;*//输出数组out数组的下标*  int wordErro = 1;*//词法分析是否正确的标志,1:正确；0：错误*  char line[100]; *//保存读入的一行表达式*  char out[100]; *//保存词法分析后带i的表达式*    int **isDigitOrChar**(char ch) {       if (ch >= 48 && ch <= 57) *//数字*            return digit;       else if (ch == 72 || ch == 104)            return Hh; *//Hh*       else if ((ch >= 65 && ch <= 70) || (ch >= 97 && ch <= 102))*//ABCDEF或者abcdef*            return AF;       else if ((ch >= 65 && ch <= 90) || (ch >= 97 && ch <= 122))*//除A-F外的其他字母*            return letter;       else if (ch == 40 || ch == 41 || ch == 42 || ch == 43 || ch == 45 || ch == 47 || ch == 35)*//(、)、\*、+、-、/ 、# 这六个运算符*            return oparetors;      return 0;  }    *//词法分析*  int **toExpress**(char\* words, char\* out) {       wordErro = 1;       state = 0;       i = 0;       j = 0;       q = words;       while (\*q) {            switch (state) {            case 0: *//当前为0状态*                 switch (**isDigitOrChar**(\*q)) {                 case digit:*//数字*                      word[i++] = \*q;                      state = 2;                      break;                 case Hh: *//Hh*                 case AF: *//ABCDEF或者abcdef*                 case letter: *//除A-F外的其他字母*                      word[i++] = \*q;                      state = 1;                      break;                 case oparetors: *//运算符*                      word[i++] = \*q;                      out[j++] = word[0];                      state = 0;  **memset**(word, 0, sizeof(word));                      i = 0;                      break;                 default: *//其他字符（非法）*                      word[i++] = \*q;                      state = 5; *//转到出错状态*                      break;                 }                 break;            case 1: *//当前为1状态*                 switch (**isDigitOrChar**(\*q)) {                 case digit:*//数字*                      word[i++] = \*q;                      state = 1;                      break;                 case Hh: *//Hh*                 case AF: *//ABCDEF或者abcdef*                 case letter: *//除A-F外的其他字母*                      word[i++] = \*q;                      state = 1;                      break;                 case oparetors: *//运算符,1状态可以为标识符的结束状态*                      word[i++] = \*q;                      state = 0;                      out[j++] = 'i';                      for (int m = 0;m < int(sizeof(word));m++) {                           if (**isDigitOrChar**(word[m]) == oparetors) {                                out[j++] = word[m];                           }                      }  **memset**(word, 0, sizeof(word));                      i = 0;                      break;                 default: *//其他字符（非法）*                      word[i++] = \*q;                      state = 5; *//转到出错状态*                      break;                 }                 break;            case 2: *//当前为2状态*                 switch (**isDigitOrChar**(\*q)) {                 case digit:*//数字*                      word[i++] = \*q;                      state = 2;                      break;                 case Hh: *//Hh*                      word[i++] = \*q;                      state = 3;                      break;                 case AF: *//ABCDEF或者abcdef*                      word[i++] = \*q;                      state = 4;                      break;                 case oparetors: *//运算符,2状态可以为十进制整数的结束状态*                      word[i++] = \*q;                      state = 0;                      out[j++] = 'i';                      for (int m = 0;m < int(sizeof(word));m++) {                           if (**isDigitOrChar**(word[m]) == oparetors) {                                out[j++] = word[m];                           }                      }  **memset**(word, 0, sizeof(word));                      i = 0;                      break;                 default: *//其他字符（非法）*                      word[i++] = \*q;                      state = 5; *//转到出错状态*                      break;                 }                 break;            case 3: *//当前为3状态*                 switch (**isDigitOrChar**(\*q)) {                 case oparetors: *//运算符,3状态可以为十六进制整数的结束状态*                      word[i++] = \*q;                      state = 0;                      out[j++] = 'i';                      for (int m = 0;m < int(sizeof(word));m++) {                           if (**isDigitOrChar**(word[m]) == oparetors) {                                out[j++] = word[m];                           }                      }  **memset**(word, 0, sizeof(word));                      i = 0;                      break;                 default: *//其他字符（非法）*                      word[i++] = \*q;                      state = 5;                      break;                 }                 break;            case 4: *//当前为4状态*                 switch (**isDigitOrChar**(\*q)) {                 case digit:*//数字*                      word[i++] = \*q;                      state = 4;                      break;                 case Hh: *//Hh*                      word[i++] = \*q;                      state = 3;                      break;                 case AF: *//ABCDEF或者abcdef*                      word[i++] = \*q;                      state = 4;                      break;                 default: *//其他字符（非法）*                      word[i++] = \*q;                      state = 5; *//转到出错状态*                      break;                 }                 break;            case 5: *//出错状态*                 switch (**isDigitOrChar**(\*q)) {                 case oparetors: *//运算符*                      word[i++] = \*q;                      state = 0;  **memset**(word, 0, sizeof(word));                      i = 0;                      wordErro = 0;                      break;                 default: *//其他字符（非法）*                      word[i++] = \*q;                      state = 5; *//转到出错状态*                      break;                 }                 break;            }            q++; *//指针下移（指向输入符号串的下一个字符）*       }         return 0;  }      */\**  *读入由i和运算符组成的式子，*  *判断语法是否合法*  *\*/*    char sym; *//保存输入的字符*  int cur; *//表达式字符串的当前下标*  char cmpchar;*//出栈的字符*  char current;*//输入串中当前分析的字符*  int error; *//语法分析的错误标志  0:正确  -1:错误*    *//定义分析栈的数据结构,双向链表*  typedef struct **node** {       char data;       struct **node**\* before;       struct **node**\* next;  }\***SNode**;  **SNode** temp, top;    *//定义入栈操作*  void **push**(char c) {       temp = (**SNode**)**malloc**(sizeof(**node**));*//申请内存空间*       temp->data = c;       temp->before = top;       temp->next = **NULL**;       top->next = temp;       top = temp;  }    *//定义出栈操作*  char **pop**() {       cmpchar = top->data;       if (top->before != **NULL**) {            temp = top;          top = top->before;            top->next = **NULL**;  **free**(temp);       }       return cmpchar;  }    *// 分析表结构如下：*  *// 存储算符优先关系表*  *// 大于关系为1，等于关系为0，小于关系为-1，出错为9*  *//+, -, \*,  /,  (,  ),  i, #*  int table[8][8] ={ */\* + \*/*{ 1, 1, -1, -1, -1, 1, -1, 1},  */\* - \*/*{ 1, 1, -1, -1, -1, 1, -1, 1},  */\* \* \*/*{ 1, 1,  1,  1, -1, 1, -1, 1},  */\* / \*/*{ 1, 1,  1,  1, -1, 1, -1, 1},  */\* ( \*/*{-1,-1, -1, -1, -1, 0, -1, 9},  */\* ) \*/*{ 1, 1,  1,  1,  9, 1,  9, 1},  */\* i \*/*{ 1, 1,  1,  1,  9, 1,  9, 1},  */\* # \*/*{-1,-1, -1, -1, -1, 9, -1, 0}                   };    *//将字符数字化*  int **Vt2d**(char current) {       switch (current) {       case '+':j = 0;break;       case '-':j = 1;break;       case '\*':j = 2;break;       case '/':j = 3;break;       case '(':j = 4;break;       case ')':j = 5;break;       case 'i':j = 6;break;       case '#':j = 7;       }       return j;  }    *//判断是否是终结符*  bool **isVt**(char ch) {       if (**isDigitOrChar**(ch) == oparetors || ch == 'i') return true;       else return false;  }      *//读入当前字符*  char **read**(**string** line, int k) {       return line**[**k**]**;  }      *//语法分析*  *//输入：符号串line，以‘#’结束*  *//输出：error=0：语法正确  error=-1：语法错误*  void **gramAnalysis**(char line[]) {       int i, j;  **push**('#');       error = 0;*//初始化为正确*       for (int l = 0; ; l++) {  *// 退出条件*            if (top->data == 'N' && top->before->data == '#' && line[l] == '#') break;    **node**\* sktemp = top;            char sk = top->data;            char a = line[l];  *// 任何两终结符之间最多只有一非终结符，若非终结符往前寻找一位即可*            if (!**isVt**(sk)) {                 sk = top->before->data;                 sktemp = top->before;            }            i = **Vt2d**(sk); *// 获取栈顶终结符*            j = **Vt2d**(a); *// 获取当前输入符号*            switch (table[i][j]) {            case 9: *// 语法错误*                 error = -1;                 break;            case 0:  **push**(a);                 break;            case -1: *// 小于，移进*  **push**(a);                 break;            case 1: *// 大于，归约*  **node**\* Q = sktemp;  **node**\* sj = sktemp;                 do {                      sj = Q;                      if (**isVt**(Q->before->data)) Q = Q->before;                      else Q = Q->before->before;                 } while (table[**Vt2d**(Q->data)][**Vt2d**(sj->data)] == 0);                 int idx = 0;                 while (top != Q) {  **pop**();                      idx++;                 }                 if (idx % 2 == 0) {                      error = -1;                      break;                 }  **push**('N');                 l--;                 break;            }            if (error == -1) break;       }  }    int **main**() {  *//逐行读入表达式，并分析是否符合语法规则*      cin**>>**line;      line[**strlen**(line)] = '#';      cur = 0;      error = 0;  *//进行词法分析*  **toExpress**(line, out);  **memset**(line, 0, sizeof(line));      for (int i = 0;i < int(sizeof(line));i++)      {          line[i] = out[i];      }      if (wordErro == 1) {*//若词法正确，则进行语法分析*          sym = **read**(line, cur);*//读入当前字符*  *//初始化指针*          top = **NULL**;          top = (**node**\*)**malloc**(sizeof(**node**));          top->before = **NULL**;          top->next = **NULL**;          top->data = ' ';  **gramAnalysis**(line);*//开始语法分析*  *//如果表达式语法分析出错*          if (error == -1)  **printf**("False");  *//如果表达式合法*          else if (error == 0)  **printf**("True");  **memset**(line, 0, sizeof(line));  **memset**(out, 0, sizeof(out));          wordErro = 1;      }      else **printf**("False");        return 0;  }  实验结果（截图） | | | | |
|  | | | | |