双栈算术表达式求值算法（C++版）

/\*P80 Dijkstra 的双栈算术表达式求值算法

\*\*逻辑：

将操作数压入操作数栈；

将运算符压入运算符栈；

忽略左括号；

再遇到右括号时，弹出一个运算符，弹出所需数量的操作数，并将运算符和操作数的运算结果压入操作数栈

\*\*效果：

输入表达式并输出表达式结果

\*\*与原来java不同的地方：c++的pop()返回为void，而不是top数据

\*/

|  |
| --- |
| "P80\_1.cpp"  #include <iostream>  #include <stack>  #include <cstdlib>  #include <cmath>  #include "P80\_1.h"  Calculation::Calculation()  {  ;  }  Calculation::~Calculation()  {  ;  }  void Calculation::start()  {  using std::string;  string ss;  std::stack<std::string> ops;  std::stack<double> vals;  while(std::cin >> ss && (ss.compare("quit") != 0))  {  if(ss.compare("(") == 0) ;  else if(ss.compare("+") == 0) ops.push(ss);  else if(ss.compare("-") == 0) ops.push(ss);  else if(ss.compare("\*") == 0) ops.push(ss);  else if(ss.compare("/") == 0) ops.push(ss);  else if(ss.compare("sqrt") == 0) ops.push(ss);  else if(ss.compare(")") == 0)  {  string op = ops.top();  ops.pop();  double v = vals.top();  vals.pop();  if(op.compare("+") == 0) {v = vals.top() + v;vals.pop();}  else if(op.compare("-") == 0) {v = vals.top() - v;vals.pop();}  else if(op.compare("\*") == 0) {v = vals.top() \* v;vals.pop();}  else if(op.compare("/") == 0) {v = vals.top() / v;vals.pop();}  else if(op.compare("sqrt") == 0) v = sqrt(v);  vals.push(v);  }  else  vals.push(atof(ss.c\_str()));  }  std::cout << vals.top() << std::endl;  vals.pop();  }  int main()  {  Calculation c;  while(1)  c.start();  return 0;  } |

|  |
| --- |
| "P80\_1.h"  #include <string>  using std::string;  class Calculation  {  public:  Calculation();  ~Calculation();  void start();  }; |