```
/**************************************
*                                     *
*     Yasser Atiya                    *
*     COP5621                         *
*     Asg 1 (Scanner for C Language)  *
*     g++ -g -o cscan cscan.cpp       *
*                                     *
**************************************/


#include <iostream>
#include <stdio.h>
#include <fstream>
#include <string>
#include <vector>
#include <iomanip>
using namespace std;


struct Token{

      Token()
      {
            count = 1;
            isident = false;
            isstring = false;
            isnumber = false;
            ischarliteral = false;
      }

      int count;
      string contents;
      bool isident, isstring, isnumber, ischarliteral;
};


string GlobalWord;
int line = 1;


vector<Token> Tokens;
vector<Token>::iterator itr;


void Parse();
void Print();
void AddtoVector(Token t);
bool isOneCharToken(char c);
bool isTwoCharToken(char c, char c2);
bool isTwoorThreeCharToken(char c);
bool isNumber(char c);
bool isIdentifier(char c);
bool isCharLiteral(char c);
```

```
bool isString(char c);
void SortVector();
void IgnoreComment(char c);


int main()
{
	Parse();
	Print();
}

void Parse()
{
	string word;
	char c;

	while(!cin.eof())
	{
		GlobalWord = " ";
		Token* t = new Token();


		cin.get(c);



		if(c == '\n')
		{
			line++;

			continue;
		}
		else if(isspace(c))
		{

			continue;
		}
		else if(isTwoCharToken(c,cin.peek()))
		{
			word = c;
			cin.get(c);
			word.push_back(c);

		}
		else if(isTwoorThreeCharToken(c))
		{
			word = GlobalWord;

		}
		else if(isOneCharToken(c))
		{
			word = c;
```

```
            }
            else if(isNumber(c))
            {
                  word = GlobalWord;
                  t->isnumber = true;

            }
            else if(isIdentifier(c))
            {
                  word = GlobalWord;
                  t->isident = true;

            }
            else if(isString(c))
            {
                  word = GlobalWord;
                  t->isstring = true;

            }
            else if(isCharLiteral(c))
            {
                  word = GlobalWord;
                  t->ischarliteral = true;

            }
            else
            {
                  if(c == '/')
                  {

                        IgnoreComment(c);
                        continue;
                  }
                  else if(isspace(GlobalWord[0]))
                        continue;
                  else
                  {
                        cerr << "illegal character: " << c
                        << " on line " << line << endl;
                        continue;
                  }
            }

            t->contents = word;

            AddtoVector(*t);

      }
}

void AddtoVector(Token t)
{
      cout << t.contents << endl;
```

```
	for(itr = Tokens.begin(); itr != Tokens.end(); itr++)
	{


		if(t.contents == itr->contents)
		{
			if(itr->isident || itr->isnumber
				|| itr->ischarliteral || itr->isstring)
				break;

			itr->count++;


			return;
		}
	}


	Tokens.push_back(t);

}

void Print()
{
	Token t1,t2,t3,t4;
	t1.contents = "number";
	t2.contents = "string";
	t3.contents = "ident";
	t4.contents = "char";

	t1.count = t2.count = t3.count = t4.count = 0;

	for(itr = Tokens.begin(); itr != Tokens.end(); itr++)
	{


		if(itr->isnumber)
		{
			t1.count++;
		}
		else if(itr->isident)
		{
			t3.count++;
		}
		else if(itr->ischarliteral)
		{
			t4.count++;
		}
		else if(itr->isstring)
		{
			t2.count++;
		}
```

```
            if(itr->ischarliteral || itr->isstring
                  || itr-> isident || itr->isnumber)
            {

                  Tokens.erase(itr);
                  itr--;
            }
      }

      if(t4.count != 0)
            Tokens.insert(Tokens.begin(),t4);
      if(t3.count != 0)
            Tokens.insert(Tokens.begin(),t3);
      if(t2.count != 0)
            Tokens.insert(Tokens.begin(),t2);
      if(t1.count != 0)
            Tokens.insert(Tokens.begin(),t1);



      cout << "\n" << setw(13) << "token" << setw(16) << "count\n";
      cout  << "--------------------" << setw(8) << "-----\n";


      SortVector();

      for(itr = Tokens.begin(); itr != Tokens.end(); itr++)
      {
            cout << setw(21) << itr->contents  << setw(7) << itr->count << endl;
      }

}

bool isOneCharToken(char c)
{
      switch(c)
      {
            case '(':
            case ')':
            case '*':
            case '+':
            case '-':
            case '=':
            case ',':
            case '.':
            case '!':
            case ':':
            case ';':
            case '?':
            case '[':
            case ']':
            case '{':
            case '}':
```

```
            case '~':
            case '<':
            case '>':
            case '&':
            case '%':
            case '|':
                  return true;
            default:
                  return false;
      }
}

bool isTwoCharToken(char c, char c2)
{
      if(c == '&' && c2 == '&')
      {
            return true;
      }
      else if(c == '|' && c2 == '|')
      {
            return true;
      }
      else if(c == '+' && c2 == '+')
      {
            return true;
      }
      else if(c == '-' && c2 == '-')
      {
            return true;
      }
      else if(c == '-' && c2 == '>')
      {
            return true;
      }
      else
      {
            return false;
      }

}

bool isTwoorThreeCharToken(char c)
{
      switch(c)
      {
            case '|':
            case '^':
            case '&':
            case '+':
            case '-':
            case '%':
            case '*':
            case '/':
```

```
		case '=':
		case '!':
		{
			if(cin.peek() == '=')
			{
				GlobalWord = c;
				cin.get(c);
				GlobalWord.push_back(c);
				return true;
			}

			break;
		}
		case '>':
		case '<':
		{
			GlobalWord = c;

			if(cin.peek() == c)
			{
				cin.get(c);
				GlobalWord.push_back(c);

				if(cin.peek() == '=')
				{
					cin.get(c);
					GlobalWord.push_back(c);
				}

				return true;
			}
			else if(cin.peek() == '=')
			{
				cin.get(c);
				GlobalWord.push_back(c);

				return true;
			}

		}

	}

	return false;
}

bool isNumber(char c)
{
	if(isdigit(c))
	{
		GlobalWord = c;
		c = cin.peek();
		while(isdigit(c) && c != cin.eof())
```

```
		{
			cin.get(c);
			GlobalWord.push_back(c);
			c = cin.peek();
		}
		return true;
	}
	else
		return false;
}

bool isIdentifier(char c)
{

	if(isalpha(c) || c == '_')
	{
		GlobalWord = c;
		c = cin.peek();
		while((isalnum(c) | c == '_') && c != cin.eof())
		{
			cin.get(c);
			GlobalWord.push_back(c);
			c = cin.peek();
		}
		return true;
	}
	else
		return false;

}

bool isCharLiteral(char c)
{
	if(c == '\'')
	{
		GlobalWord = c;

		if(cin.peek() == '\'')
		{
			cin.get(c);
			cerr <<	"character has zero length on line " << line << endl;
			GlobalWord = " ";
			return false;
		}

		do
		{
			cin.get(c);
			GlobalWord.push_back(c);


			if(cin.peek() == '\n')
			{
```

```
                        cin.get(c);
                        cerr << "missing ' for " << GlobalWord
                        << " on line " << line << endl;
                        line++;
                        GlobalWord = "";
                        return false;
                  }
                  if(cin.peek() == '\' && c == '\')
                  {
                        cin.get(c);

                        if(cin.peek() == ''')
                        {
                              GlobalWord.push_back(c);
                              break;
                        }
                        else
                              cin.putback(c);
                  }
                  if(cin.peek() == '''){
                        if(c == '\')
                              continue;

                        break;
                  }
                  if(cin.peek() == cin.eof()){

                        break;
                  }
            }
            while(true);

            cin.get(c);
            GlobalWord.push_back(c);

            if(GlobalWord.length() > 4 || GlobalWord[1] != '\'
                  && GlobalWord.length() > 3)
            {
                  cerr <<     "character constant " << GlobalWord
                  << " too long on line " << line << endl;
                  GlobalWord = "";
                  return false;
            }

            return true;
      }
      else
            return false;

}

bool isString(char c)
{
```

```
if(c == '"')
{



      GlobalWord = c;

      if(cin.peek() == '\n')
      {
            cerr << "missing \" for " << GlobalWord
            << " on line " << line << endl;

            GlobalWord = "";
            return false;
      }
      else if(cin.peek() == '"')
      {
            cin.get(c);
            GlobalWord.push_back(c);
            return true;
      }


      do
      {

            cin.get(c);
            GlobalWord.push_back(c);



            if(cin.peek() == '\n')
            {
cerr << "missing \" for " << GlobalWord
<< " on line " << line << endl;

                  GlobalWord = "";
                  return false;
            }
            if(cin.peek() == '"' && c != '\'){

                  break;
            }
            else if(cin.peek() == cin.eof()){

                  break;
            }
      }
      while(true);

      cin.get(c);
      GlobalWord.push_back(c);
```

```
            return true;
      }
      else
            return false;
}

void SortVector()
{
      bool sorted = false;
      while(!sorted)
      {
            sorted = true;
            for(itr = Tokens.begin(); itr != Tokens.end(); itr++)
            {

if(itr != Tokens.end()-1 && (itr->count < (itr+1)->count
      || (itr->contents.length() < (itr+1)->contents.length()
      && itr->count == (itr+1)->count)
      || (itr->contents > (itr+1)->contents
      && itr->count == (itr+1)->count
      && itr->contents.length() == (itr+1)->contents.length())))
                  {
                        sorted = false;

                        int holder;
                        holder = itr->count;
                        itr->count = (itr+1)->count;
                        (itr+1)->count = holder;
                        itr->contents.swap((itr+1)->contents);
                  }

            }
      }
}

void IgnoreComment(char c)
{
      int nlcount = 0;
      string s;
      s = c;


      if(cin.peek() == '/')
      {
            getline(cin,s);

      }
      else if(cin.peek() == '*')
      {

            do
            {
                  cin.get(c);
```

```
                if(c == '\n'){
                        line++;
                        nlcount++;
                }

                if(c == '*' && cin.peek() == '/')
                        break;
                if(cin.peek() == cin.eof() || nlcount > 10000)
                {
                        cerr << "unclosed comment\n";
                        return;
                }
        }
        while(true);

        cin.get(c);

    }
}
```