C++单词拼装器

一. 设计要求

把C++源代码中的各类单词（记号）进行拼装分类，标明.cpp文件中那个单词所属的类别，并对.cpp文件中的一些单词进行替换，设计出自己的C++语言。

二. 设计思路

逐行读进.cpp文件，再对每一行中的每个字母进行，判断它们分别属于哪些类别，对关键字类别的单词，进行简单的小写转大写的替换。

三. 常量和判别方法的定义

```
//关键字类别，具体内容省略
const QString keywords[65] = {....};
//特殊符号类别，单符号的
const char symbols[24] = {.....};
//特殊符号类别，双符号，因为三符号的只有两种，  直接在if种判断
const QString doubleSyms[11] = {.....};
//用来输出类别
const QString output[6] {......};
```

```
int judge(QString word);
void readSentence(QString sentence);
//去除多余的空格
int removeBlank(QString sentence, int begin, int len);
bool isKeywords(QString word);
bool isDigit(QString word);
bool isDoubleSyms(QString s);
bool isSymbols(QCharRef c);
bool _isString(QString s);
```

四. 实现思路

1. 一些变量的声明

```
QString res;      //用来输出类别判断结果的字符串
QString myLag;    //用来输出我的C++语言的字符串
QString sentence; //.cpp文件的某一行
QString word;     //还没输出的字符串
int begin;        //sentence的下标
bool isComment;   //标志现在是否在/*.... */中
bool isString;    //标志现在是否在" "中
bool isNum        //在特殊符号后，则isNum=true，说明'+','-'要与数字连在一起，不能单独做特殊符号
```

2. 对注释类别的处理

```
if(!isString && begin != len - 1 && sentence[begin] == '/' && sentence[begin
+ 1] == '/') {
    for(int t = begin; t<len;t++) {
        res.append(sentence[t]);
    }
    res.append(tab + output[5] + "\n");
    myLag.append(sentence + "\n");
    return;
}
```

对单行注释解(//)的, 因为那一行只能是注解, 所以直接把这一行全都加到结果中, 就可以了。

```
if (!isString && begin != len - 1 && sentence[begin] == '/' &&
sentence[begin + 1] == '*') {
    isComment = true;
    int t = begin;
    for(t = begin; t<len;t++) {
        if (t!=len - 1 && sentence[t] == '*' && sentence[t+1] == '/') {
            res.append("*/" + tab + output[5] + "\n");
            myLag.append("*/");
            isComment = false;
            begin = t + 2;
            break;
        } else {
            res.append(sentence[t]);
            myLag.append(sentence[t]);
        }
    }
    if (isComment) {
        res.append("\n");
        myLag.append("\n");
        return;
    }
    else {
        continue;
    }
}
```

对于多行注释, 如果当前一行读到*/, 就把isComment设置为false, 否则设置为true, 如果isComment为true, 在读取下一行时, 首先执行类似的逻辑, 然后再进行其他的判断。

```
if (isComment) {
    int i = 0;
    for(i = begin; i < len; i++) {
        if (i!=len - 1 && sentence[i] == '*' && sentence[i+1] == '/') {
            res.append("*/" + tab + output[5] + "\n");
            myLag.append("*/");
            isComment = false;
            break;
        } else {
            res.append(sentence[i]);
            myLag.append(sentence[i]);
        }
    }
    if (isComment) {
        res.append("\n");
```

```
            myLag.append("\n");
            return;
        }
        begin = i + 2;
    }
```

3. 对以#开头的sentence(#include, #define) 进行判断

```
//去除#后面的空格
begin = removeBlank(sentence, begin, len);

if (sentence[begin] == 'i') {
    res.append("include" + tab + output[0] + "\n");
    myLag.append("INCLUDE");
    begin += 7;

    //去除include后面的空格
    begin = removeBlank(sentence, begin, len);
    while(begin <= len - 1) {
        if (sentence[begin] == ' ')
            break;
        res.append(sentence[begin]);
        myLag.append(sentence[begin]);
        begin++;
    }
    res.append(tab + output[1] + "\n");
    myLag.append("\n");
    return;
}
```

如果#后面跟着i表明是#include, 先include, 再读取头文件即可。

```
res.append("define" + tab + output[0] + "\n");
myLag.append("DEFINE");
begin += 6;

begin = removeBlank(sentence, begin, len);

while(begin <= len - 1) {
    if (sentence[begin] == ' ')
        break;
    res.append(sentence[begin]);
    myLag.append(sentence[begin]);
    begin++;
}
res.append(tab + output[2] + "\n");

begin = removeBlank(sentence, begin, len);
int res_ = 3;
if (sentence[begin] == '"') res_ = 4;

while (begin <= len - 1) {
    if (sentence[begin] != ' ' || res_ ==4)
        res.append(sentence[begin]);
    myLag.append(sentence[begin]);
    begin++;
```

```
}
myLag.append("\n");
res.append(tab + output[res_] + "\n");
```

否则是#define, 则先读取标识符, 再根据标识符后面是否双引号, 判断标识符后是字符串还是数字。

4. 遇到符号且符号在不在" "中

如果符号是' . ', 且符号前是数字, 则说明是小数点

```
if(sentence[begin] == '.' && !word.isEmpty() && ((word[0]>='0'&&word[0]
<='9')||word[0]=='-' || word[0]=='+')) {
    word+='.';
    begin++;
    continue;
}
```

如果符号是'+', '-', 且isNum

```
if ((sentence[begin] == '-' || sentence[begin] == '+') && isNum) {
    if (begin < len - 1 && sentence[begin + 1] <= '9' && sentence[begin + 1]
>= '0') {
        word += sentence[begin];
        begin++;
        continue;
    }
}
```

除了前面两种情况, 遇到符号时, 如果word不为空, 则把word做输出且输出, res = 0说明是关键字, 让关键字大写, 然后添加到myLag中

```
if (!word.isEmpty()) {
    int res_ = judge(word);
    res.append(word + tab + output[res_] + "\n");
    if (res_ == 0) {
        myLag.append(word.toUpper());
    } else {
        myLag.append(word);
    }
    word = "";
}
```

接来来对符号进行处理, 首先判断符号后面是否还是符号, 如果是, 对三元符号和二元符号进行判断, 不是则直接输出

```
QString temp = "";
temp += c;
if(begin != len - 1 && isSymbols(sentence[begin + 1])) {
    temp += sentence[begin + 1];
    if (begin < len - 2 && ((sentence[begin] == '<' && sentence[begin +1 ]
== '<' && sentence[begin + 2] == '=')||
    (sentence[begin] == '>' && sentence[begin +1] == '>' && sentence[begin +
2] == '='))) {
        res.append("" + sentence[begin] + sentence[begin+1] + sentence[begin
+ 2] + tab + output[1] + "\n");
```

```
        myLag.append("" + sentence[begin] + sentence[begin + 1] +
sentence[begin + 2]);
        begin+=2;
    } else if (isDoubleSyms(temp)) {
        begin++;
        res.append(temp + tab + output[1] + "\n");
        myLag.append(temp);
    }
        else {
            res.append(c + tab + output[1] + "\n");
            myLag.append(c);
    }
} else {
    res.append(c + tab + output[1] + "\n");
    myLag.append(c);
}
```

5. 如果当前字符不是空格
   首先设置isNum = false, 如果当前字符是'e'或者'E', 且word是数字, 则设置isNum = true, 如果当前字符是' " ', 则如果' " '前不是' \ \ '. 则isString = !isString

```
if (sentence[begin] != ' ') {
    isNum = false;
    if ((sentence[begin] == 'e'||sentence[begin] == 'E')&& !word.isEmpty()&&
judge(word) == 3)
        isNum = true;
        word += sentence[begin];
        if (sentence[begin] == '"') {
            if (begin == 0)
                isString = !isString;
             else if (begin > 0 && sentence[begin - 1] != '\\')
                isString = !isString;
         }
}
```

6. 遇到了空格
   首先如果isString = true, 则把空格加入到word中, 否则如果word不为空, 则判断word类型然后输出。

```
else if(sentence[begin] == ' ') {
    if (isString) {
        isNum = false;
        word += sentence[begin];
    }

    else {
        if (word.isEmpty()) {
            begin++;
            continue;
            }

        int res_ = judge(word);
        res.append(word + tab + output[res_] + "\n");
        if (res_ == 0) {
            myLag.append(word.toUpper());
        } else {
            myLag.append(word);
```

```
            }
            word = "";
            begin = removeBlank(sentence, begin, len) - 1;
        }
    }
```

7. 最后如果word不为空, 则把word输出, 且置为空。

五. 代码运行逻辑

首先点击选择文件按钮, 触发点击事件, 读取文件, 且把文件的内容设置的第一个文本框中; 然后点击分词按钮, 触发点击事件, 则调用`d.getDivision(fileName, ui->resText);`, 分词后, 把分词结果储存到d的res字符串中, 把我的语言储存到myLag中, 并把res设置到第二个文本框中; 点击我的语言, 把之前储存的myLag设置到第三个文本框中。

六. 操作逻辑

1. 点击选择文件, 选择要分词的源文件
2. 点击分词, 对源文件进行分词
3. 点击我的语言, 替换关键字, 得到我的cpp程序。