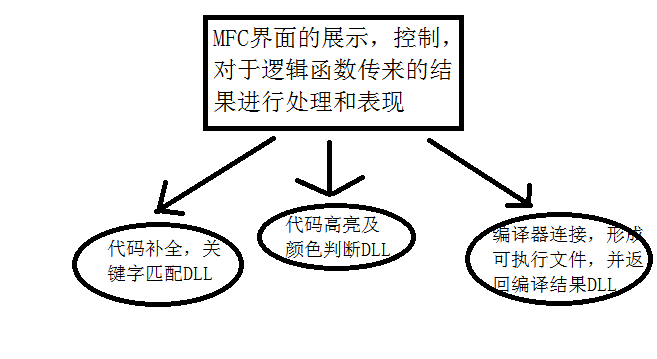
**CocoCode设计说明书**

1. **项目架构：**



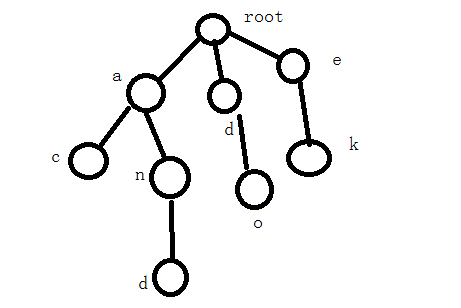
1. **各个模块的实现：**
2. **代码补全**

编辑器采用的是AC自动机算法，通过构造字典树，从而提高关键字前缀匹配的效率。

**a.字典树的构造：**

字典树中每个节点都有k个儿子，其中k为关键字会出现的元素个数（在编辑器中，一般符合变量名命名规则的才能成为关键字，因此包括大小写字母，数字和下划线共63种可能），但是对于一个节点，并不是每个儿子都占用空间，只有那些属于关键字前缀的节点的儿子才会生成新的节点，而其他节点皆为空。

假设我们拥有的关键字是 ac , and , do , ek



**b.时间空间复杂度**

那么构造的字典树如上图所示，a除了有c和n两个儿子有节点存在之外，其他儿子皆指向空节点。这样的存储模式，在空间上最大程度的压缩了关键字占用的内存，而在时间复杂度上，由于树的分叉多，提高了字符串匹配的精准性，从而提高了效率。

**c.节点内部数据结构**

另外对于每一个节点，其内部数据结构由：count和newcount计数器以及next数组指针组成。Count记录从root走到以当前节点为结尾的关键字个数，这样的记录方便识别完整的关键字，同时在count为0时也能即时将节点释放掉，减少以后匹配搜索的代价。

**d.DLL模块实现的功能**

第一个是代码补全：

根据当前用户输入的字符，从root往下搜索到相应的节点，然后以该节点为根，对其子树进行深度优先搜索，找出所有存在于子树中的关键字。

第二个是支持用户自定义关键字：

编辑器每隔一段时间，就会调用DLL中rebuild函数，对当前文本中用户自定义的关键字进行更新，为了提高字典树更新的效率，我并不是把整棵树删掉之后重新插入，而是利用节点中newcount计数器，记录新的关键字在字典树中的分布情况，最后调用maintain函数，对整棵树中的空节点进行及时的释放，以提高补全的效率，当然，搜索整棵树是十分耗时的，因此我在树节点中加入visit变量，因此我在字典树更新的同时，对于字典树经过的路径，将visit赋值为true。这样，当找到visit为false的节点时就不用再往下寻找了,直接将整棵子树删除即可。

**e.安全机制：**

每当一个单词插入字典树时，对每个字符进行合法性分析，对于不符合命名规则的单词将不予以加入字典树，保证了字典树的稳定性。

**2.编译链接模块：**

本来只是想做一个编辑器的，但我觉得一个编辑器没有编译的功能实在是有点美中不足，于是，在代码中调用system()函数，利用windows中cmd窗口的命令，对编辑器生成的文件进行编译。并将结果记录在一个文本文件中，编辑器再从文本文件中获取编译和运行的结果返回给用户。但有些遗憾的是，system函数仅仅支持const char\*作为参数，因此中文的文件路径名在编译过程中可能会产生问题，同时，由于环境变量设置的不同，仅能支持在编辑器所在目录下的文档进行编译，我们小组也上网找了很久，试图解决这一问题，可是一直得不到很好的解决方案。

**3.代码高亮模块：**

1. 代码高亮部分主要借助CRichEditCtrl的特性，加上智能判断当前单词是不是标识符，用CRichEditCtrl::SetSelCharformat来设置单词字体实现高亮功能。
2. 对于单个关键字，在键入一个字符的时候，只有两种情况——①键入这个字符之后，光标左右两侧是一个单词；②键入这个字符之后，光标左右两侧是两个单词，所以只要判断光标左右两个单词需不需要更新高亮即可。
3. 对于C中特有的宏定义部分，实际上宏定义可以看做是一种特殊的行注释，但是由于如果这样写宏定义需要判断的还是关键词太多（如#include #ifdef #ifndef等），所以为了方便起见，宏定义的判断加入到关键字的判断里面。
4. 对于注释部分和字符串显示部分。可以认为注释和字符串显示都是一种特殊的注释，所以这些可以一并而论，而且因为一个注释里面不能包含另外一个注释，所以块注释，行注释还有字符串可以作为同一种注释来一并讨论。
5. 对于编辑器，每隔10s会重新把整个文档全部刷新一次，重新判断高亮部分，方法是先判断是否是注释，然后在没有被注释的区域判断单个关键词的高亮。
6. 正因为5）的原因，本来在复制粘贴还有删除剪切之后全部都要重新判断整个文档的高亮，但是因为10s自动判断一次，所以那些代码全部可以省略，稍微等等就能得到高亮结果。

**函数功能：**

// 得到所指定位置的字符

wchar\_t GetCharAt(long pos, wchar\_t & ch);

// 得到所指定光标相对位置上的单词的颜色

void GetWordColor(long offset, COLORREF & color);

// 封装CRichEditCtrl的Set / Get SelCharformat函数

void SetSelColor(COLORREF color);

void GetSelColor(COLORREF & color);

// 进一步方便编程，开放某一区间字体颜色的查询与设置（左闭右开）

void GetRangeColor(long left, long righ, COLORREF & color);

void SetRangeColor(long left, long righ, COLORREF color);

// 开放某区间文本的替换功能

void ReplaceText(long left, long righ, LPCTSTR text);

// 开放替换一个单词的功能，但注意是相对当前光标offset位置的单词

void ReplaceWord(long offset, LPCTSTR word);

// 得到与设置某一行的缩进量

long GetLineIndent(long index, long & indent);

void SetLineIndent(long index, long indent);

// GetWindowText和SetWindowText的wstring版本

void GetTextWstring(wstring & text);

void SetTextWstring(wstring text);

// 处理输入单个字符(可见)的情况

afx\_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);

// 得到offset位置单词

void GetWordWstring(long offset, wstring & word);

// 得到当前行数

long GetCurrentLine(long & index);

// 判断单词是否高亮

void HilightWord(long offset, bool hilight\_anyway);

// 设置自动缩进的函数

void AutoIndent(long line);

// 全部文档重新判断高亮的函数

void UpdateFromScratch(void);

// 检查哪些是注释

void CheckComment(long start);

afx\_msg void OnTimer(UINT\_PTR nIDEvent);