设计并实现一个简单的行编辑系统。该系统提供 5 条编辑子命令：

open 文件名：将指定文件读入内存；

quit：将正在编辑的文件写回文件，退出编辑系统；

list n1 – n2：显示第 n1 到第 n2 行的内容；

del 行号，列号，字符数：从指定位置开始删除指定的字符数；

ins 行号，列号，字符串：在指定位置插入字符串。

【解】要实现一个编辑系统，需要有一个能够编辑某一个文件的工具。于是我们先定义一个类 Editor 完成这项任务。当我们需要编辑一个文件的时候，就生成一个 Editor 类的对象，传给它的参数是需要编辑的文件的名字，指示该对象完成对文件的编辑。有了 Editor 类，编辑系统的实现就相当简单。编辑系统只是显示编辑命令菜单，供用户选择。然后调用 Editor 类的成员函数完成相应的功能。编辑系统的实现见代码清单 2-13。

代码清单 2-13 文本编辑系统的实现

```
1. int main(){
2.     Editor *p = NULL;                    //指向正在编辑的对象
3.     char command[20], comName[10];
4.     int i;
5.
6.   //显示使用说明
7.     cout << "欢迎使用行编辑系统~使用说明如下：" << endl;
8.     cout << "open 文件名：将指定文件读入内存；" << endl;
9.     cout << "quit：将正在编辑的文件写回文件；" << endl;
10.    cout << "list n1 – n2 或 list：显示第 n1 到第 n2 行的内容或所有内容；"
11.     << endl;
12.    cout << "del 行号，列号，字符数：从指定位置开始删除指定的字符数；" <<
endl;
13.    cout << "ins 行号，列号，字符串：在指定位置插入字符串。" << endl;
14.    cout << "exit：退出编辑" << endl;
15.
16.    while (true){        //不断接受用户指令，完成相应的功能，直到输入 exit
17.     //输入命令
18.        cout << "请输入命令：";
19.        cin.getline(command, 20);
20.
21.        //取出命令中的名字部分，存入 comName
22.     for (i=0; !(command[i] == ' ' || command[i] == '\0' ); ++i);
23.        strncpy(comName , command ,i);
24.        comName[i] = '\0';
```

```
25.
26.          if (strcmp(comName, "exit") == 0) {                //exit 处理
27.                if (p != NULL) delete p;       //如果有文件在编辑，则关闭
28.                cout << "欢迎再次使用本系统！ " << endl;
29.                break;
30.          }
31.
32.          //打开文件前是否输入了其他编辑命令
33.    if (strcmp(comName, "open") && p == NULL){
34.                cout << "请先打开文件!" << endl;
35.                continue;
36.          }
37.          for (; command[i] == ' ' ; ++i);           // 找到命令的参数部分
38.
39.          if (strcmp(comName, "open" ) == 0)  //open 命令生成一个 Editor 对象
40.        p = new Editor(command + i);
41.          else if (strcmp(comName, "list") == 0)          //list 命令处理
42.        p->List(command + i);
43.          else if (strcmp(comName, "del") == 0)          //delete 命令处理
44.          p->Del(command+i);
45.          else if (strcmp(comName, "ins") == 0)          //insert 命令处理
46.          p->Ins(command+i);
47.          else if (strcmp(comName, "quit") == 0)          //quit 命令处理
48.        { delete p;                      //编辑结束，删除当前对象
49.          p = NULL;                     //当前正在编辑的对象为空
50.        }
51.          else cout << "错误命令！ " << endl;
52.    }
53.    return 0;
54.}
```

对于每个编辑命令，Editor 类都有相应的成员函数。Open 命令由构造函数完成。当需要编辑一个文件时，则生成一个 Editor 类的对象，传给它需要编辑的文件名。构造函数首先将文件读入内存，在内存中完成所有的编辑工作。Quit 命令由析构函数完成。当遇到 quit 命令时，编辑结束，该对象完成了它的历史使命，可以消亡了。消亡前，必须将内存中修改过的文件重新存入文件。其他的三个编辑命令在 Editor 类中都有对应的成员函数。

设计 Editor 类的关键在于读入的文件在内存中如何存放。为了方便插入和删除，我们可以用一个单链表来存储，每个结点存放一行，每一行的值是一个字符串。使用 string 类可以简化处理。因为行有长有短，用 string 类型可以不用考虑存储空间的问题。按照这种存储方式，这几条编辑命令都很容易实现。List 是显示某几个结点的值。Ins 和 del 是对某个结点的数据进行插入或删除。如果在这个编辑系统中还要增加插入一行或删除一行的功能，那只不过是在单链表中插入一个结点或删除一个结点。

Editor 类的定义见代码清单 2-14。对应五条编辑命令，它有五个公有的成员函数。有两个数据成员：需要编辑的文件名以及对应的内存中的单链表的头指针。除此之外，Editor 类还有一组私有的成员函数。它们是公有的成员函数实现时所需要用到的工具函数。我们将在用到时再介绍。

代码清单 2-14 Editor 类的定义

```
1.  class Editor{
2.  private:
3.      struct Node{                    //单链表的结点类，用于存储一行
4.          string data;
5.          Node* next;
6.
7.      public:
8.          Node( string x = "", Node* n = NULL){ data = x; next = n; }
9.          ~Node(){}
10.     };
11.
12.     Node* head;                         //单链表的头指针
13.     char *filename;                     //当前对象对应的文件名
14.
15.     void Error(){                       //出错处理函数
16.         cout << "出错啦~ 您输入的命令有误或超出范围，请重新输入哦~" <<
endl; }
17.
18.     int getNum(char *&s);               // 从字符串 s 中提取一个整数
19.     Node* Move( int x );                // 返回存储第 x 行的结点的地址
20.     bool skip(char *&s, char ch);       //在字符串 s 中寻找字符 ch
21.
22. public:
23.     Editor(const char *);
24.     ~Editor();
25.
26.     void List(char *);
27.     void Del(char *);
28.     void Ins(char *);
29. };
```

构造函数将文件读入单链表。每从文件中读入一行，就在单链表的表尾添加一个存储这一行的结点。它的定义如代码清单 2-15 所示。

代码清单 2-15 Editor 类的构造函数

```
1.  Editor::Editor(const char *s)
2.  {
3.      filename = new char[strlen(s)+1];
```

```
    strcpy(filename, s);

ifstream file(s);  // 打开文件
    string str;

if ( file.is_open()) {              // 文件被正确打开
        Node *p = head = new Node;      // 申请头结点
        while (true) {                  // 读文件，直到结束
            getline(file, str);         // 从文件读入一行
            if (file.eof()) break;      //读完跳出循环
            p->next = new Node( str );   //添加一个结点存储这一行
            p = p->next;
        }
        cout << "打开成功！" << endl;
        file.close();
    } else {
        cout << "打开失败，请确认文件名是否正确！" << endl;
    }
}
```

析构函数将链表中的内容重新写回文件，并释放链表。其实现见代码清单 2-16。

代码清单 2-16 Editor 类的析构函数的实现

```
Editor::~Editor(){
    ofstream file(filename);                    // 打开输出文件

    if ( !file.is_open()) { Error(); return;}
    Node *p = head->next, *q;  // p 为当前正在处理的结点，q 是 p 的后继
    while ( p != NULL ){        // 将所有的结点写回文件，并删除结点
        q = p->next;
        file << p->data << endl;
        delete p;
        p = q;
    }
    delete head;                                //删除头结点
    file.close();                               //关闭文件
    cout << "编辑完成，文件已保存！" << endl;
}
```

List 函数实现 List 命令的功能。arg 是 List 命令的参数，其格式为：起始行 – 终止行。参数也可以为空，参数为空表示显示文件的所有内容。当参数为空时，遍历整个单链表，显示每个结点的值。当参数非空时，从参数中提取起始行和终止行，显示链表中相应的结点值 。List 函数的实现见代码清单 2-17。

代码清单 2-17 List 函数的实现

```
1. void Editor::List(char * arg){ //arg 为显示范围，arg 为空串表示显示整个文件
2.     int beg , end;                          // 显示的起始和终止行号
3.     Node *p ;
4.
5.     if (*arg == '\0') {                     // 显示整个文件
6.         p = head->next;
7.         while (p != NULL) {
8.             cout << p->data << endl;
9.             p = p->next;
10.        }
11.        return;
12.    }
13.
14.  // 显示 beg 到 end 之间的所有行
15.  beg = getNum(arg);                        // 获取起始位置
16.    if (!skip(arg, '-')) { Error(); return; }
17.    end = getNum(arg);                       // 获取终止位置
18.
19.    if ( end < beg || beg < 1) { Error(); return; }      //检查参数合法性
20.
21.    p = Move(beg);                      //获得 beg 行所在的结点地址
22.    if (p ==NULL) {Error(); return;}
23.    end -= beg - 1;
24.    while (end--) {                  // 从 beg 开始显示 end-beg+1 行
25.        if ( p == NULL ) { Error(); return; }
26.        cout << p->data << endl;
27.        p = p->next;
28.    }
29.}
```

Del 函数完成 Del 命令的功能，删除某行中从某一列开始的若干个字符。arg 是 Del 命令的参数，它的格式为：行号，列号，删除的字符数。具体实现见代码清单 2-18。Del 函数首先从参数中取出行号、列号和删除的字符数，然后根据行号找到单链表中相应的结点，在该结点中删除指定的字符数。

代码清单 2-18 Del 函数的实现

```
1. void Editor::Del(char *arg){          // arg 为删除命令的参数
2.     int row, col, num;
3.
4.    row = getNum(arg);                      // 获取行号
5.     if (!skip(arg, ',')) { Error(); return; }
6.    col = getNum(arg);                       //获取列号
```

```
7.      if (!skip(arg, ',')) { Error(); return; }
8.     num = getNum(arg);                    // 获取删除的字符个数
9.
10.     Node *p = Move(row);                  //找到存储所处理行的结点
11.     if (p == NULL) {Error(); return;}
12.     if ( col > p->data.size() ) { Error(); return; } //检查参数合法性
13.     p->data.erase( col-1, num );        //从下表 col-1 开始删除 num 个字符
14.     cout << "删除成功！" << endl;
15.     cout << p->data << endl;                  //显示删除后的结果
16.     }
```

Ins 函数完成 Ins 命令的功能，在某行的某一列开始插入若干个字符。arg 是 Ins 命令的参数，它的格式为：行号，列号，插入的字符。具体实现见代码清单 2-19。Ins 很少拿书的实现与 Del 函数非常相似。首先从参数中提取出行号、列号和所需插入的字符串，然后根据行号找到单链表中相应的结点，在该结点存储的字符串中插入所需插入的字符串。

代码清单 2-19 Ins 函数的实现

```
1.  void Editor::Ins(char *arg){
2.      int row, col;
3.
4.     row = getNum(arg);                    //获取插入的行号
5.      if (!skip(arg, ',')) { Error(); return; }
6.     col = getNum(arg);                    //获取插入的列号
7.      if (!skip(arg, ',')) { Error(); return; }
8.
9.      Node *p = Move(row);                  //找到存储插入行的结点
10.     if (p == NULL) {Error(); return;}
11.     if ( col > p->data.size() ) p->data = p->data + arg;
12.     else p->data.insert( col-1, arg );  //在第 col-1 的位置插入字符串 arg
13.     cout << "插入成功!" << endl;
14.     cout << p->data << endl;                  //显示插入后的结果
15.}
```

在这些公有函数实现时用到了一些工具函数，这些函数被定义为 Editor 类的私有成员函数。getNum 函数是从参数 s 中获取一个十进制的正整数，其定义见代码清单 2-20。

代码清单 2-20 getNum 函数的实现

```
1.  int Editor::getNum(char *&s) {          // s 是某个命令行的参数
2.      int num = 0;                        //保存取出的正整数
3.      while (*s == ' ') ++s;              //跳过参数中前置的空格
4.      while (*s <= '9' && *s >= '0')      // 将读入的参数转换成正整数
5.          num = num * 10 + *(s++) - '0';
6.      return num;
7.  }
```

Move 函数在单链表中找出存放第 x 行的结点的地址，其定义见代码清单 2-21。

代码清单 2-21 Move 函数的实现

```
1. Editor::Node * Editor::Move( int x ){
2.   Node *p = head;
3.     while (x--) {                    // 从 head 开始指针往后移 x 次
4.           if ( p->next == NULL ) return NULL;
5.           p = p->next;
6.     }
7.     return p;
8. }
```

Skip 函数检查字符串 arg 中的第一个有效字符是否为字符 ch，其定义见代码清单 2-22。

代码清单 2-22 skip 函数的实现

```
9. bool Editor::skip(char *&arg, char ch)
10.{
11.    while (*arg == ' ') ++arg;                    // 跳过前置空格
12.    if (*(arg++) == ch) return true; else return false;
13.}
```