计算器（Calculator）实验

作者	山外メ雲ジ
E-Mail	minisong@foxmail.com
QQ	860317732
博客	sosong.blog.chinaunix.net
硬件平台	野火 STM32 开发板
库版本	ST3.0.0

目录

版权声明

本程序 COUNT 模块部分（包括 main 函数）由本人独立完成，如有雷同，必属抄袭。 

请尊重他人劳动成品，采用本程序源代码时请注明作者信息。 

实验描述

将 MicroSD 卡(以文件系统 FATFS 访问)里面的 BMP 格式文件显示到液晶屏幕上作为主界面，通过触摸 触摸屏上的按键来控制计算器，程序后台运行计算结果并显示到屏幕上。

注意：本程序运行，还需要把 ".\资源备份\CAL.bmp" 放入 SD 卡根目录。

用户文件

核心模块文件	
USER\ COUNT\COUNT.H	数据类型的定义、函数的声明
USER\ COUNT\ COUNT_CFG.H	模块的配置文件（独立出来是为了增强移植性）
USER\ COUNT\COUNT.C	函数的定义
其他需要用到的文件	
USER/main.c	主函数
USER/stm32f10x_it.c	自带库
USER/lcd.c	LCD 液晶显示
USER/Touch.c	屏幕触摸
USER/ Sd_bmp	读取 SD 卡里 BMP 图片函数，用于显示主界面
USER/usart1.c	串口发送

USER/SysTick.c	延时
……	

功能简介

野火 STM32 开发板计算器模块，支持负数运算、小数运算、括号运算、复合运算、加减乘除连续运算……

由于模块的运算是不涉及硬件的，模块从一开始设计就考虑到移植性的问题，具有较好的移植性。可以通过修改仅仅修改 COUNT_CFG.H 就移植到其他单片机上（单片机需要支持递归，部分单片机需要特殊设置才能支持递归的，如 51。移植时就要特殊修改）。

操作界面

“D”，Delete，表示退格，按一次删除式子最后一个输入符号。

“C”，Clean，表示清空，按一次清空式子。

LCD 的第一行显示运算式，第二行显示运算结果。

设计思路、流程图及代码

设计思路很简单，主要是输入式子，把输入的式子以数组的形式存放，当按下等于号就调用 calculate 计算结果，再用 float2stre，把计算结果转换为字符串再显示出来。

为了方便操作输入式子及计算式子，模块定义了两个结构体：

```
/**********定义式子计算结构体，调用 calculate 计算结果用到**********/
struct num_point{
    float       result;     //计算结果
    song_u8     *point;     //指向式子中计算到的位置
};

/**********定义式子数组，便于对式子添加删除操作**********/
struct str_point{
    song_u8     str[SN];    //输入的计算式子
    song_u8     strn;       //式子长度
};
```

对式子的添加元素、删除元素、清空操作函数：

```
//对计算式子操作
#define LastSP(sp) (sp.strn>0 ? sp.str[sp.strn-1]:0)          //查询式子最后一位
#define LastSPp(sp)(sp->strn>0 ? sp->str[sp->strn-1]:0)
extern  void    DelSP    (struct str_point *sp);               //删除式子最后一位
extern  void    ClearSP  (struct str_point *sp);               //清空式子
extern  song_u8 AddCheckSP(struct str_point *sp,song_u8 key);
            //带简单检测功能的添加字符到式子尾部，添加字符有效返回 1，否则返回 0
```

计算式子及将结果转换为字符串：

```
extern struct   num_point    calculate    (song_u8 *str,song_u8 N);
                                    //计算 式子（检查最后一位是否为空）
 extern          void        float2str    (float num,song_u8 *str); //实型转字符串
 extern          void        float2stre   (float num,song_u8 *str);
                                    //实型转字符串,科学记数法表示
```

main 函数

这个程序有 3 个 main 函数，通过宏定义 MODE 来选择编译哪个 main 函数的。

```
#define MODE    1 //这里有 3 个 main 函数，即 3 个模式
                  //模式 1：调用 CountMain,是计算器模块自带的一个测试程序
                  //模式 2，是 CountMain 的展开，可以方便嵌入自己的程序
                  //模式 3，是测量触摸屏按键用的，测试用而已
```

使用模式 1 需要对 COUNT_CFG.H 进行配置输入输出函数：

```
//------------------以下的是测试程序 CountMain 用到----------------------------//
```

```
2.  /*******************************定义显示、清屏函数*******************************
3.  * 这里定义的是液晶的函数，当然也可以改成是串口之类的。
4.  * 例如串口的话：不需要清屏函数，可以把 CLEAN_* 定义为空。
5.  *                 其他的函数参考 COUT 编写串口输出
6.  ************************************************************************/
7.  #include    "lcd.h"
8.  #define     CLEAN_S         LCD_Str_R(26,304,"
     ",36,0x0000,0xffff)                            //清屏（输入式子的位置）
9.  #define     CLEAN_R         LCD_Str_R(59,304,(u8 *)"              ",14,0x0000,0xf
fff)                                                //清屏（计算结果的位置）
10. #define     ERROR           LCD_Str_R(59,304,(u8 *)"         error",14,0x0000,0xf
fff)                                                //输出有误（前面加空格是为了清屏）
11. #define     PRINT_S(x)      LCD_Str_R(26,304,(u8 *)x,36,0x0000,0xffff)
                                                    //显示输出结果
12. #define     PRINT_R(x)      LCD_Str_R(59,304,(u8 *)x,14,0x0000,0xffff)
                                                    //显示输入式子
13.
14. /*******************************定义按键函数*******************************
15. * 需要用户编写的扫描按键函数，传递进去的是指针变量
16. * 有按键按下就返回非 0
17. * x 是按键变量，取值： 0 1 2 3 4 5 6 7 8 9 . = + - * / ( ) D C
18. * D 表示退格 del，C 表示清空 clean
19. * 本来打算用枚举表示的，但反而显示符号没那么直观，就放弃了。
20. * 下面给出的是例子而已，可以是改成是按键、触摸、串口接收之类
21. ************************************************************************/
22. #include    "includes.h"
23. #define     SCAN_KEY(x)         ScanTouch(&x)
24.
25. /*******************************定义延时函数*******************************/
26. #include    "systick.h"
27. #define    DELAY     delay_ms(100)//延时函数，避免触摸屏按的速度太快。
28. //#define  DELAY(x)  delay_ms(x)  //不用这个，方便用户可以自己在这个 CFG 文件中修改延时
29.
30. //----------------------以上的是测试程序 CountMain 用到--------------------------//
```

移植时如果不用到 CountMain 函数 就不需要修改上面的配置文件。然后 main 函数就可以非常简洁了。CountMain 函数 里面的变量 定义为静态变量，即下次调用时仍然保留原来数据。

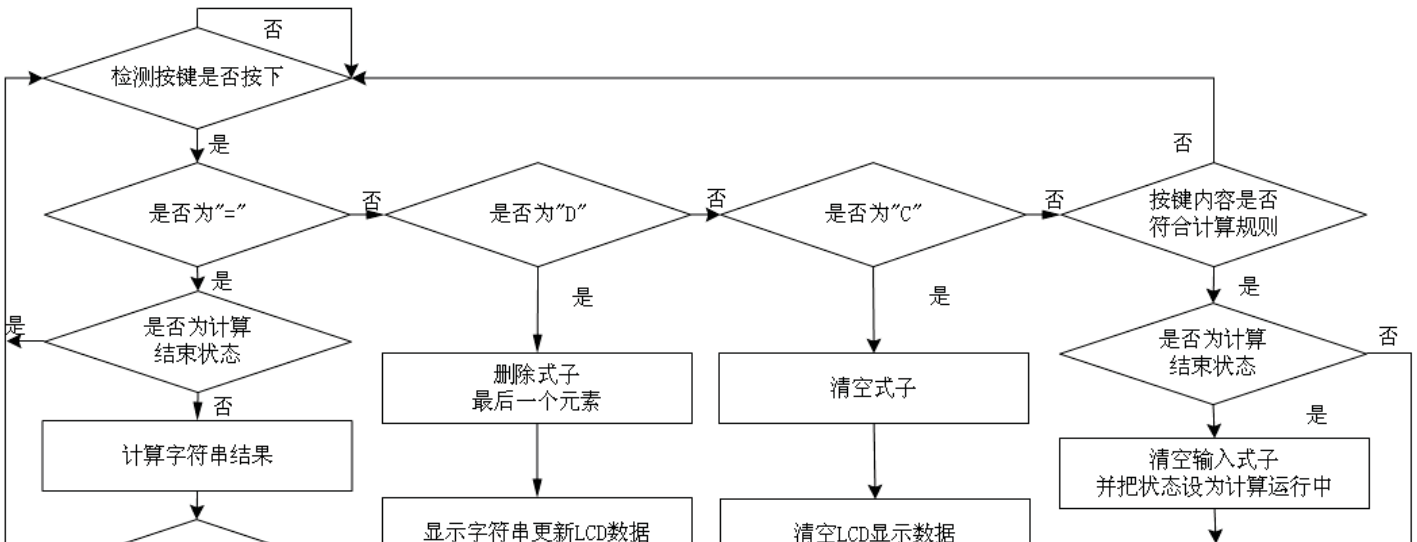
```
1.  int main(void)
2.   {
3.       Init();                                                    //初始化
4.       LCD_Show_8x16_String(2,29,0,"10");                         //显示 左上角 10 进制
5.
6.       while (1)
7.       {
8.           CountMain();          //计算器任务
9.           //其他用户需要的任务……
10.      }//while
11.  }//main
```

提示：加入其他用户需要的任务时，可以修改 COUNT_CFG.H 里的延时宏定义，适当减少延时提高灵敏度。

CountMain 函数 的执行流程跟模式 2 是一样的，独立 CountMain 函数出来是为了用户编写程序时简单一些，代码不用太长。上面的代码是不是很短啊？呵呵，我把计算的执行流程隐藏掉了，方便用户添加自己的其他任务。用户想在计算器的基础上实现其他功能，建议直接在模式 1 中添加，而不像模式 2 那样长长的代码，让人看到都头晕

具体的执行流程如何？这里就直接讲模式 2，模式 2 其实就是 CountMain 函数 的展开：

先来流程图，流程图比较容易了解思路：



```
#define   N  10       //N 是 N 进制，可以改成其他进制，不过输出还是科学计数法，即 10 进制的
int main(void)
{
    char    key=0;                          //按下的键
    u8      Result[RN];                     //计算结果字符串，把计算结果转换为字符串
    struct  num_point tmp;                  //计算结果
    struct  str_point sp;                   //计算式子
    char    CountStatus=calnot;             //计算状态

    Init();                                 //初始化
    LCD_Show_8x16_String(2,29,0,"10");

    ClearSP(&sp);

    while (1)
    {
        if(SCAN_KEY(key))           /*如果触笔有按下*/
        {
            printf( "key=%c ",key );
            switch(key)
            {
            case '=':   //----------------------计算式子----------------------
                        if(CountStatus==caled)continue;
                        CountStatus=caled;              //计算完毕

```

```
26.                             tmp=calculate(sp.str,N);          //计算字符串结果
27.                             if(tmp.point==0)
28.                             {
29.                                 ERROR;                        //输入式子有误
30.                                 break;
31.                             }
32.                             CLEAN_R;                          //清屏
33.                             float2stre(tmp.result,Result);    //把计算结果转为字符串
34.                             PRINT_R(Result);                  //显示计算结果
35.                             break;
36.             case 'C':   //----------------------清空式子----------------------
37.                             CountStatus=caling;
38.                             ClearSP(&sp);
39.                             CLEAN_R;                          //清屏
40.                             CLEAN_S;
41.                             break;
42.             case 'D':   //----------------------退格----------------------
43.                             CountStatus=caling;
44.                             DelSP(&sp);
45.                             CLEAN_R;                          //清屏
46.                             CLEAN_S;
47.                             PRINT_S(sp.str);                  //更新屏幕字符串
48.                             break;
49.             default:    //----------------------输入式子----------------------
50.                                                               //（剩下的就是运算表达
式）
51.                             if(CountStatus==caled)            //如果计算完成后，就清空
52.                             {
53.                                 CountStatus=caling;
54.                                 ClearSP(&sp);
55.                                 CLEAN_R;                      //清屏
56.                                 CLEAN_S;
57.                             }
58.                             if(AddCheckSP(&sp,key))           //输入 key 有效
```

```
                                {
                                    PRINT_S(sp.str);              //更新屏幕字符串
                                }
            }//switch
            delay_ms(100);
        }//if
    }//while
}//main
```

为什么有了模式 1，还给个模式 2 呢？主要是很多人都习惯把程序代码写在 main 函数上，容易看到程序的执行流程。而且很多人连静态变量的声明也不会（用 static 声明），直接讲 模式 1 ，他们不知道程序退出后为什么数据不会被销毁。

看到上面的程序，用法很简单，觉得挺简单吧？

还看不懂？OMG，上帝与你同在那就来个精简版的代码吧，这样肯定能看明白……删掉液晶显示之类，留下对式子控制和计算：

```
while (1)
{
    if(SCAN_KEY(key))            /*如果触笔有按下*/
    {
        switch(key)
        {
        case '=':   //----------------------计算式子----------------------
            tmp=calculate(sp.str,N);        //计算字符串结果
            if(tmp.point==0)  break;        //计算有误时返回 0
            float2stre(tmp.result,Result); //把计算结果转为字符串
            PRINT_R(Result);                //显示计算结果
            break;
        case 'C':   //----------------------清空式子----------------------
            ClearSP(&sp);
            break;
        case 'D':   //-------------------------退格-----------------------
            DelSP(&sp);
            break;
        default:    //----------------------输入式子----------------------
```

```
20.                                                    //（剩下的就是运算表达式）
21.             if(AddCheckSP(&sp,key))          //输入 key 有效
22.                 PRINT_S(sp.str);             //更新屏幕字符串
23.             break;
24.         }//switch
25.     }//if
26. }//while
```

代码简单了好多了吧？其实就是学如何调用我写好的库函数而已。熟悉了如何调用后，就开始讲核心的计算式子，这是这个计算器的关键代码，当然，看懂也是有难度的，做好心理准备哦。

Calculate 函数

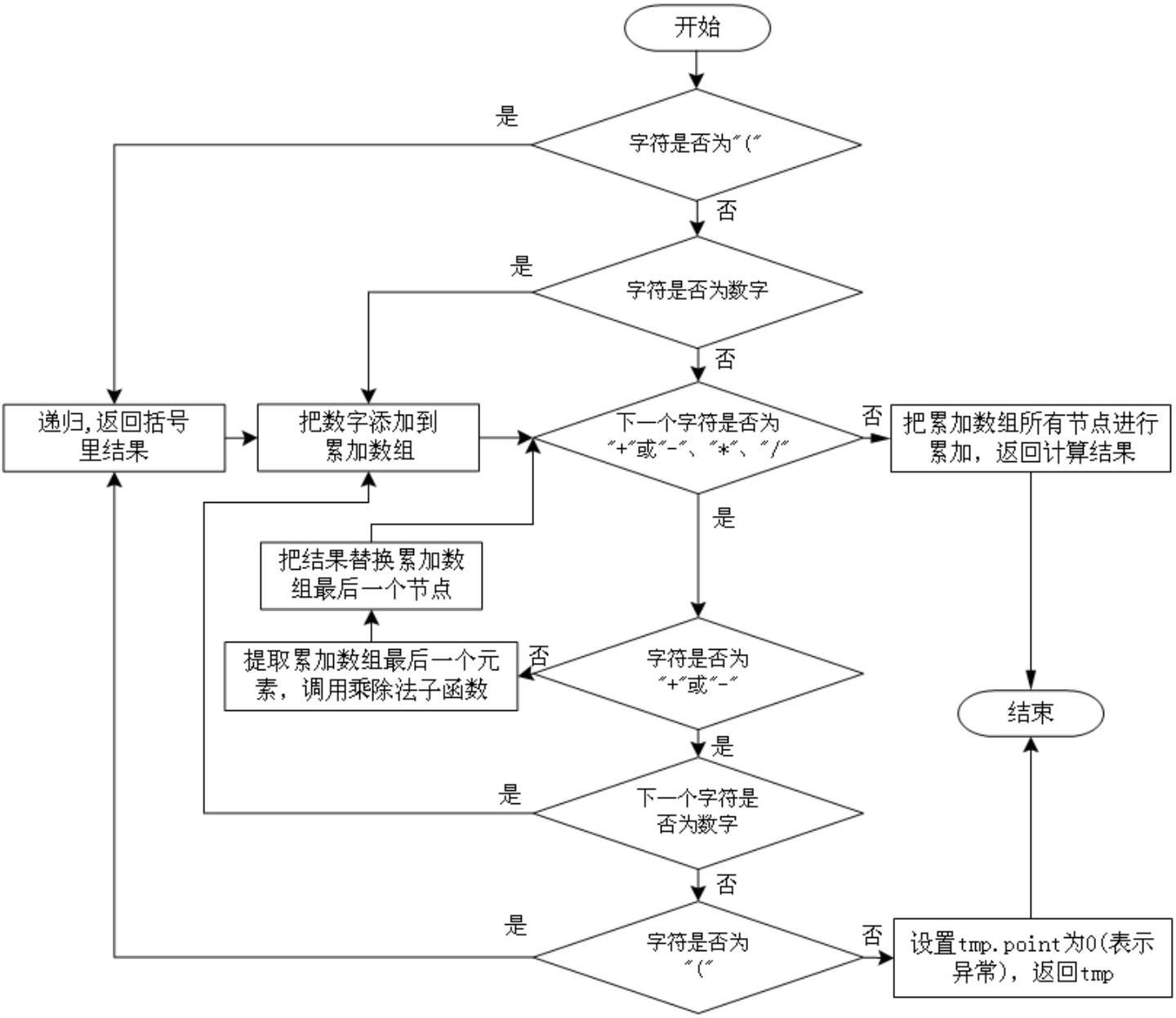
跳进 calculate 函数的函数体看下，就会发现其实 calculate 函数是包了外壳的 con_add 函数，带验证结果是否最终运算结果或者运算错误。

con_add 函数，累加的意思，提取式子的加减数存进累加数组；遇到乘除号就把最后添加到累加数组的元素提取出来，进行乘除运算后放回进去；遇到括号就把括号里面的式子当成一条独立的式子来调用 con_add 函数计算，即递归，把返回值添加累加数组。遇到式子结尾或者其他错误计算表达时返回，结束运算。

里面用到几个内部函数：

```
1.  static struct  num_point  str2num      (song_u8 *str,song_u8 N);       //N 为 N 进制
2.  static struct  num_point  str_division (float cal,song_u8 *str,song_u8 N);//乘除法
3.  static struct  num_point  cal_bk       (song_u8 *str,song_u8 N);
4.                                  // calculate bracket, 递归 con_add 函数来计算括号内式子
```

先看流程图吧（  高手就直接跳到下面的源代码吧）：



举例说一下计算流程：

3	.	1	4	+	5	/	3	+	6	*	(	8	+	1	/	3	)		

调用 str2num 把 3.14 加入累加数组

3.14	0	0	0	0

调用 str2num 把 5 加入累加数组

3.14	5	0	0	0

遇到除号，提取 5 出来，调用 str_division 计算 5/3,替换累加数组里的 5

3.14	1.67	0	0	0

调用 str2num 把 6 加入累加数组

3.14	1.67	6	0	0

遇到乘号，提取 6 出来，乘号后面是前括号，调用 cal_bk 计算括号里面的式子,返回的结果与 6 相乘后，替换累加数组里的 6

3.14	1.67	50	0	0

最后一步就是把累加数组的全部数据进行累加：3.14+1.666666+50

原理不难吧？呵呵，那就看下源代码了。如果你仅仅想移植，而不是研究其内部实现代码，可以直接跳过源代码，直接看移植部分。

con_add 函数的源代码：

```
1.  struct num_point  con_add(song_u8 *str,song_u8 N)
2.  {
3.      struct  num_point tmp;           //返回结果
4.      float   ADDS[ADDN];              //累加数组
5.      float   *adds=ADDS;              //指示式子计算的到的位置
6.      song_u8 i;
7.
8.      for(i=0;i<ADDN;i++)*(ADDS+i)=0; //清空 ADDS
9.
10.     if(*str=='(')  //开头括号,则调用递归函数求出括号内部的值
11.     {
12.         tmp = cal_bk(str,N);if(tmp.point==0) return tmp //递归
13.         *adds++=tmp.result;                                    //插入数据
14.         str=tmp.point;                                         //保存剩余字符串
15.     }
16.
17.     if(*str>='0' && *str<='9')          //数字或者加减号（加减号就是正负数）
18.     {
19.         tmp   = str2num(str,N);   //此处仅识别接着的数字，不会产生空指针，故不检错
20.         *adds++=tmp.result;                                    //插入数据
21.         str   = tmp.point;
22.     } else *adds++=0;       //顶点为括号就前面添加 加 0 ，保证 p 指向一个已经累加的数
23.
24.     while( *str=='+' || *str=='-' || *str=='*' || *str=='/')
25.                                                 //加减乘除，前面已经检查过'('了
26.     {
27.         if( *str=='+' || *str=='-')          //加减号之后只能是数字或者括号
```

```
28.         {
29.             if(*(str+1)=='(')                    //括号则调用递归函数计算括号内的数据
30.             {
31.                 tmp=cal_bk(str+1,N);
32.                 if(tmp.point==0) return tmp;   //递归
33.             }
34.             else if(*(str+1)>='0'&&*(str+1)<='9') tmp    = str2num(str+1,N);
35.                                                     //数字则直接保存
36.             else{CERROR("加减号后有误");tmp.point=0;return tmp;}
37.                                         //+-号后面不是数字或者括号，就报错：* / )号
38.             if(*str=='-') tmp.result = -tmp.result;     //保存 +-数据
39.             *adds++=tmp.result;
40.             str=tmp.point;
41.         }
42.
43.         else if(*str=='*' || *str=='/')    //乘除号之后只能是数字或者括号
44.         {
45.             if(*(str+1)=='(')              //括号则调用递归函数计算括号内的数据
46.             {
47.                 tmp = cal_bk(str+1,N);          //递归
48.                 if(tmp.point==0)
49.                 {
50.                     CERROR("乘除号后调用括号函数有误");
51.                     tmp.point=NULL;return tmp;
52.                 }
53.                 if(*str=='*') *(adds-1)= *(adds-1) * tmp.result;
54.                 else          *(adds-1)= *(adds-1) / tmp.result;
55.                 str=tmp.point;
56.             }else if(*(str+1)>='0'&&*(str+1)<='9')
57.                                 //数字则与原来数据进行乘除再保存，即乘除优先级比加减高
58.             {
59.                 tmp=str_division(*(adds-1),str,N);
60.                 if(tmp.point==0) return tmp;
61.                 str    = tmp.point;
```

```
62.                 *(adds-1)    =    tmp.result; //乘除法不需要移动，直接先计算乘除法
63.             }
64.             else                            // * /号 后面不能是 + - ( 号
65.             {
66.                 tmp.point=0;
67.                 return tmp;
68.             }
69.         }
70.      //对于其他非法字符( 例如：) )，就不用计算，结束计算，交给调用此函数的上一层函数处理
71.     }
72.     tmp.result=0;
73.     for(i=0;i<ADDN;i++)tmp.result+=*(ADDS+i);            //累加 ADDS
74.     tmp.point = str;
75.     return tmp;
76. }
```

有没有头晕。不管你晕不晕，反正我晕了……考虑的条件太多，调试了好久才写出来觉得稳定的。

移植

写在前面的话：**请尊重他人劳动成品，采用本程序源代码时请注明作者信息。**

本程序在设计时已经考虑到移植性的问题，已经测试过仅仅修改 COUNT_CFG.H 就移植飞思卡尔的 MC9S12XS128 单片机上，对于 51 单片机，需要仅仅修改一下声明为递归函数也能快速移植。

本来程序设计时最初是采用链表方式实现的（可以动态分配内存，也可以作为教程来学链表），后来发现占用内存太大，不适合移植到其他小内存的单片机上，最终换成数组实现。配置文件为：COUNT_CFG.H

COUNT_CFG.H 源代码：

```
1.
2.  #ifndef _COUNT_CFG_H_
3.  #define _COUNT_CFG_H_
4.
5.  #include "stm32f10x.h"
6.
7.  define      DUBUG            //Debug 模式
```

```
8.
9.
10. /*****************************设置数据类型*****************************/
11. typedef     unsigned    char        song_u8;      //无符号型
12. typedef     unsigned    short int   song_u16;
13. typedef     unsigned    int         song_u32;
14.
15. typedef                 char        song_s8;      //有符号型
16. typedef                 short int   song_s16;
17. typedef                 int         song_s32;
18. /*****************************设置数据类型*****************************/
19.
20.
21. /*****************************设置计算式子*****************************/
22. #define     ADDN            20              //连加数组个数
23. #define     SN              80              //字符串长度（最多可以接受的计算式子字符）
24. #define     RN              30              //计算结果字符串长度
25.
26.
27. /*********************************定义输出*******************************
28. * 调用时可以不添加双引号：COUT("xx");   COUT(xx);
29. * 这两种表达都行，前者编辑器不识别中文字符，后者打印时多了双引号
30. * 下面给的仅仅是例子，可以改成在液晶上显示、串口发送之类的
31. ************************************************************************/
32. #include "usart1.h"
33. #define     COUT(str)       printf("\nout:"#str"\n")         //定义正常输出
34. #define     CERROR(str)     printf("\nerror:"#str"\n")       //定义异常输出
35.
36. #ifdef      DUBUG
37. #define     CDUBUG(str)     printf("\ndebug:"#str"\n")       //定义调试输出
38. #else
39. #define     CDUBUG(str)                                     //空，即非 DUBUG 模式不产生输出
40. #endif
```

```
41.
42.  //----------------------以下的是测试程序 CountMain 用到--------------------------//
43.
44.  /*******************************定义显示、清屏函数*******************************
45.  * 这里定义的是液晶的函数，当然也可以改成是串口之类的。
46.  * 例如串口的话：不需要清屏函数，可以把 CLEAN_* 定义为空。
47.  *                其他的函数参考 COUT 编写串口输出
48.  ************************************************************************/
49.  #include    "lcd.h"
50.  #define     CLEAN_S         LCD_Str_R(26,304,"
      ",36,0x0000,0xffff)                                  //清屏（输入式子的位置）
51.  #define     CLEAN_R         LCD_Str_R(59,304,(u8 *)"              ",14,0x0000,0xf
fff)                                                       //清屏（计算结果的位置）
52.  #define     ERROR           LCD_Str_R(59,304,(u8 *)"         error",14,0x0000,0xf
fff)                                                       //输出有误（前面加空格是为了清屏）
53.  #define     PRINT_S(x)      LCD_Str_R(26,304,(u8 *)x,36,0x0000,0xffff)
                                                           //显示输出结果
54.  #define     PRINT_R(x)      LCD_Str_R(59,304,(u8 *)x,14,0x0000,0xffff)
55.                                                        //显示输入式子
56.
57.  /*******************************定义按键函数*******************************
58.  * 需要用户编写的扫描按键函数，传递进去的是指针变量
59.  * 有按键按下就返回非 0
60.  * x 是按键变量，取值： 0 1 2 3 4 5 6 7 8 9 . = + - * / ( ) D C
61.  * D 表示退格 del，C 表示清空 clean
62.  * 本来打算用枚举表示的，但反而显示符号没那么直观，就放弃了。
63.  * 下面给出的是例子而已，可以是改成是按键、触摸、串口接收之类
64.  ************************************************************************/
65.  #include    "includes.h"
66.  #define     SCAN_KEY(x)         ScanTouch(&x)
67.
68.  /*******************************定义延时函数*******************************/
69.  #include    "systick.h"
70.  #define     DELAY               delay_ms(100)    //延时函数，避免触摸屏按的速度太快。
71.  //#define   DELAY(x)            delay_ms(x);     //不用这个，方便用户修改延时
```

```
72.
73.  //----------------------以上的是测试程序 CountMain 用到-------------------------//
74.
75.  #endif  //_COUNT_CFG_H_
```

上面的源代码已经注解得很详细了，CountMain 函数用到的宏定义前面也已经说过了，这里就不多说了，主要是下面这 3 个宏定义：

```
1.  #define     ADDN                20   //连加数组个数
2.  #define     SN                  80   //字符串长度（最多可以接受的计算式子字符）
3.  #define     SCAN_KEY(x)         ScanTouch(&x)
```

前面两个是设置数组的，决定你输入式子的长度和最终能输入多少个加减号。

后面一个是 CountMain 函数的配置，注意有个取地址符号：”&”。即调用宏定义函数是传进去看上去不是指针，定义 ScanTouch 时传递进去的是指针哦。即

```
1.  //声明函数
2.  void ScanTouch(char *x); //是指针哦
3.
4.  //函数中调用
5.  char key;                 //不是指针哦
6.  SCAN_KEY(key)
```

教程就讲得这里，是不是有点乱  技术不过关，语言表达能力欠缺   ……

实验讲解完毕，野火嵌入式开发工作室祝大家学习愉快^_^……