http://c.biancheng.net/cpp/html/1428.html

标准的C和C++都不支持正则表达式，但有一些函数库可以辅助C/C++程序员完成这一功能，其中最著名的当数Philip Hazel的Perl-Compatible Regular Expression库，许多Linux发行版本都带有这个函数库。  
  
C语言处理正则表达式常用的函数有regcomp()、regexec()、regfree()和regerror()，一般分为三个步骤，如下所示：

**C语言中使用正则表达式一般分为三步：**

1. 编译正则表达式 regcomp()
2. 匹配正则表达式 regexec()
3. 释放正则表达式 regfree()

下边是对三个函数的详细解释

1、int regcomp (regex\_t \*compiled, const char \*pattern, int cflags)

这个函数把指定的正则表达式pattern编译成一种特定的数据格式compiled，这样可以使匹配更有效。函数regexec 会使用这个数据在目标文本串中进行模式匹配。执行成功返回０。

参数说明：

①regex\_t 是一个结构体数据类型，用来存放编译后的正则表达式，它的成员re\_nsub 用来存储正则表达式中的子正则表达式的个数，子正则表达式就是用圆括号包起来的部分表达式。

②pattern 是指向我们写好的正则表达式的指针。

③cflags 有如下4个值或者是它们或运算(|)后的值：

REG\_EXTENDED 以功能更加强大的扩展正则表达式的方式进行匹配。

REG\_ICASE 匹配字母时忽略大小写。

REG\_NOSUB 不用存储匹配后的结果。

REG\_NEWLINE 识别换行符，这样'$'就可以从行尾开始匹配，'^'就可以从行的开头开始匹配。

2. int regexec (regex\_t \*compiled, char \*string, size\_t nmatch, regmatch\_t matchptr [], int eflags)

当我们编译好正则表达式后，就可以用regexec 匹配我们的目标文本串了，如果在编译正则表达式的时候没有指定cflags的参数为REG\_NEWLINE，则默认情况下是忽略换行符的，也就是把整个文本串当作一个字符串处理。执行成功返回０。

regmatch\_t 是一个结构体数据类型，在regex.h中定义：

typedef struct

{

regoff\_t rm\_so;

regoff\_t rm\_eo;

} regmatch\_t;

成员rm\_so 存放匹配文本串在目标串中的开始位置，rm\_eo 存放结束位置。通常我们以数组的形式定义一组这样的结构。因为往往我们的正则表达式中还包含子正则表达式。数组0单元存放主正则表达式位置，后边的单元依次存放子正则表达式位置。

参数说明：

①compiled 是已经用regcomp函数编译好的正则表达式。

②string 是目标文本串。

③nmatch 是regmatch\_t结构体数组的长度。

④matchptr regmatch\_t类型的结构体数组，存放匹配文本串的位置信息。

⑤eflags 有两个值

REG\_NOTBOL 按我的理解是如果指定了这个值，那么'^'就不会从我们的目标串开始匹配。总之我到现在还不是很明白这个参数的意义；

REG\_NOTEOL 和上边那个作用差不多，不过这个指定结束end of line。

3. void regfree (regex\_t \*compiled)

当我们使用完编译好的正则表达式后，或者要重新编译其他正则表达式的时候，我们可以用这个函数清空compiled指向的regex\_t结构体的内容，请记住，如果是重新编译的话，一定要先清空regex\_t结构体。

4. size\_t regerror (int errcode, regex\_t \*compiled, char \*buffer, size\_t length)

当执行regcomp 或者regexec 产生错误的时候，就可以调用这个函数而返回一个包含错误信息的字符串。

参数说明：

①errcode 是由regcomp 和 regexec 函数返回的错误代号。

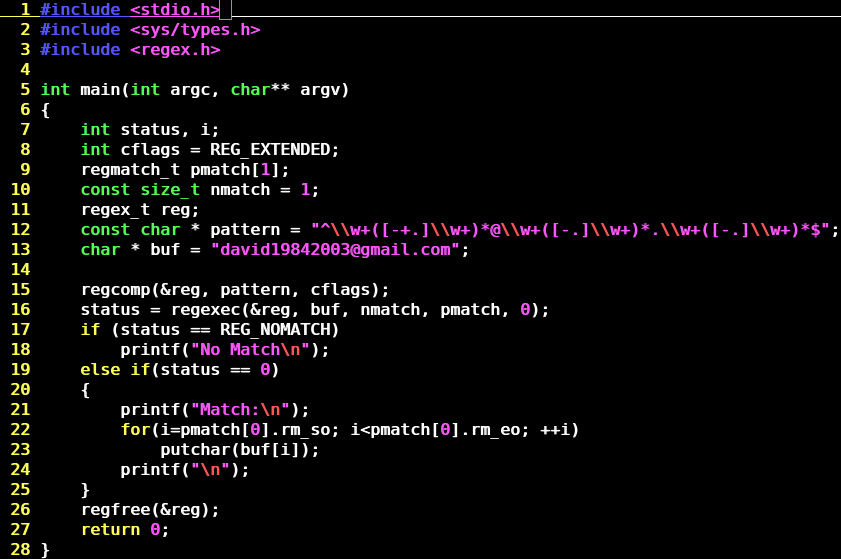
②compiled 是已经用regcomp函数编译好的正则表达式，这个值可以为NULL。

③buffer 指向用来存放错误信息的字符串的内存空间。

④length 指明buffer的长度，如果这个错误信息的长度大于这个值，则regerror 函数会自动截断超出的字符串，但他仍然会返回完整的字符串的长度。所以我们可以用如下的方法先得到错误字符串的长度。

size\_t length = regerror (errcode, compiled, NULL, 0);

下边是一个匹配Email例子，按照上面的三步就可以。



下面的程序负责从命令行获取正则表达式，然后将其运用于从标准输入得到的每行数据，并打印出匹配结果。

#include <stdio.h>

#include <sys/types.h>

#include <regex.h>

/\* 取子串的函数 \*/

static char\* substr(const char\*str,

unsigned start, unsigned end)

{

unsigned n = end - start;

static char stbuf[256];

strncpy(stbuf, str + start, n);

stbuf[n] = 0;

return stbuf;

}

/\* 主程序 \*/

int main(int argc, char\*\* argv)

{

char \* pattern;

int x, z, lno = 0, cflags = 0;

char ebuf[128], lbuf[256];

regex\_t reg;

regmatch\_t pm[10];

const size\_t nmatch = 10;

/\* 编译正则表达式\*/

pattern = argv[1];

z = regcomp(?, pattern, cflags);

if (z != 0){

regerror(z, ?, ebuf, sizeof(ebuf));

fprintf(stderr, "%s: pattern '%s' \n",ebuf, pattern);

return 1;

}

/\* 逐行处理输入的数据 \*/

while(fgets(lbuf, sizeof(lbuf), stdin))

{

++lno;

if ((z = strlen(lbuf)) > 0 && lbuf[z-1] == '\n')

lbuf[z - 1] = 0;

/\* 对每一行应用正则表达式进行匹配 \*/

z = regexec(?, lbuf, nmatch, pm, 0);

if (z == REG\_NOMATCH) continue;

else if (z != 0) {

regerror(z, ?, ebuf, sizeof(ebuf));

fprintf(stderr, "%s: regcom('%s')\n", ebuf, lbuf);

return 2;

}

/\* 输出处理结果 \*/

for (x = 0; x < nmatch && pm[x].rm\_so != -1; ++ x)

{

if (!x) printf("%04d: %s\n", lno, lbuf);

printf(" $%d='%s'\n", x, substr(lbuf, pm[x].rm\_so, pm[x].rm\_eo));

}

}

/\* 释放正则表达式 \*/

regfree(?);

return 0;

}

执行下面的命令可以编译并执行该程序：

# gcc regexp.c -o regexp

# ./regexp 'regex[a-z]\*' < regexp.c

0003: #include <regex.h>

$0='regex'

0027: regex\_t reg;

$0='regex'

0054: z = regexec(?, lbuf, nmatch, pm, 0);

$0='regexec'