C/C++文件处理

# 过滤器

只能从标准输入中读入并且只能写到标准输出上的程序叫做过滤器。

BUFSIZ在<stdio.h>中定义，指的是许多标准I/O函数中使用的内部缓冲区的大小。它同时也是你缓冲区大小的理想选择。

## example1：把一行从标准输入拷贝到标准输出中

#ifndef USE\_CPLUSPLUS

#include <stdio.h>

int main()

{

char s[BUFSIZ];

while(gets(s))

puts(s);

return 0;

}

#else

#include <iostream>

#include <string>

using namespace std;

int main()

{

string s;

while(getline(cin, s))

cout << s << endl;

}

#endif

## example2：允许选择重定向的拷贝过滤器

#include <stdio.h>

#include <stdlib.h>

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

{

char s[BUFSIZ];

// 打开可选择的输入文件

if (argc > 1)

{

if (freopen(argv[1], "r", stdin) == NULL)

{

return EXIT\_FAILURE;

}

}

// 打开可选择的输出文件

if (argc > 2)

{

if (freopen(argv[2], "w", stdout) == NULL)

{

return EXIT\_FAILURE;

}

}

while(gets(s))

puts(s);

return EXIT\_SUCCESS;

}

## example3：通过显示的文件指针来拷贝文件

fgets区别于gets的地方在于fgets需要知道缓冲区的大小，gets只是假设有足够的空间。它一直读到比那些字符少1时为止，这样它能附加一个'\0',并且它能在所返回的字符串中保留这个换行字符(如果有空间的话是这样，而gets总是丢弃换行字符)。它们各自配套的输出函数像所期望的那样执行：puts添加一个换行符给输出，而fputs则不。

#ifndef USE\_CPLUSPLUS

#include <stdio.h>

#include <stdlib.h>

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

{

if (argc == 3)

{

char s[BUFSIZ];

FILE \*inf, \*outf;

if ((inf = fopen(argv[1], "r")) == NULL)

{

return EXIT\_FAILURE;

}

if ((outf = fopen(argv[2], "w")) == NULL)

{

return EXIT\_FAILURE;

}

while(fgets(s, BUFSIZ, inf))

fputs(s, outf);

fclose(inf);

fclose(outf);

return EXIT\_SUCCESS;

}

else

{

return EXIT\_FAILURE;

}

}

#else

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

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

{

if (argc == 3)

{

string s;

ifstream inf(argv[1]);

ofstream outf(argv[2]);

if (!inf || !outf)

{

return EXIT\_FAILURE;

}

while(getline(inf, s))

outf << s << endl;

}

}

#endif

## example4： 带有错误检测的 copy3

fgets和gets在进行到文件结尾或出错时都返回NULL，通常不需要有任何另外的错误检查。然而，应该显示地检查输出错误。因为在PC系统中用完磁盘空间是很常见的。通过调用ferror来做错误检验。由于文件I/O是缓冲的，因此必须在进行错误检查之前刷新输出缓冲。一旦文件的错误状态被设置了，它保持这个状态不变直到通过调用clearerr或者rewind来复位它。

while(fgets(s, BUFSIZ, inf))

{

fputs(s, outf);

fflush(outf);

if (ferror(outf))

{

return EXIT\_FAILURE;

}

}

# 二进制文件

## example5： 拷贝二进制文件

在MSDOS的文本模式下，内存中的每一个换行字符在输出设备中都被"\r\n"替换，然而在输入时与这个过程相反(这是从过去的CP/M时代继承下来的)。此外，CTRL+Z被解释成文件结束，因此在文本模式下不可能跳过Ctrl+Z来读入数据。在二进制模式下没有那么多的翻译产生 -- 在内存中的数据和在磁盘中的数据是一样的。

fread和fwrite分别对数据块进行读和写操作，它们返回已经成功处理的数据块(不是字节)的数量。在这个例子中，各项恰巧是以字节出现的(即长度为1的模块)。当fwrite返回值小于各项要求的数量时，就知道发生了写错误，在这种情况下没有必要显示地调用ferror。任何由fwrite写入的非字符数据都在输出设备中以二进制模式存储，肉眼通常看不见它。

read和write方法与fread、fwrite作用相似。istream::gcount返回“所得到的的数”，即由istream::read传递的字节数。

#ifndef USE\_CPLUSPLUS

#include <stdio.h>

#include <stdlib.h>

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

{

if (argc == 3)

{

char buf[BUFSIZ];

FILE \*inf, \*outf;

if ((inf = fopen(argv[1], "rb")) == NULL)

{

return EXIT\_FAILURE;

}

if ((outf = fopen(argv[2], "wb")) == NULL)

{

return EXIT\_FAILURE;

}

while(!feof(inf))

{

int nitems = fread(buf, 1, BUFSIZ, inf);

if (fwrite(buf, 1, nitems, outf) != nitems)

{

return EXIT\_FAILURE;

}

}

fclose(inf);

fclose(outf);

return EXIT\_SUCCESS;

}

else

{

return EXIT\_FAILURE;

}

}

#else

#include <iostream>

#include <fstream>

#include <stdlib.h>

using namespace std;

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

{

if (argc == 3)

{

char buf[BUFSIZ];

ifstream inf(argv[1], ios::in | ios::binary);

ofstream outf(argv[2], ios::out | ios::binary);

while(inf)

{

inf.read(buf, BUFSIZ);

outf.write(buf, inf.gcount());

if (!outf)

{

return EXIT\_FAILURE;

}

}

return inf.fail() ? EXIT\_FAILURE : EXIT\_SUCCESS;

}

else

{

return EXIT\_FAILURE;

}

}

#endif

# 记录处理

函数fread和fwrite适合处理有固定长度记录的文件。

在打开模式请求中的“+”表示更新模式，这意味着允许文件的输入和输出。然而，必须通过调用fflush或者一些文件定位命令如fseek和rewind，来分隔输入操作和输出操作。

命令fseek把读/写光标定位在从文件开头开始数的给定数量的字节上(SEEK\_SET)，或从文件结尾开始数的给定数量的字节上(SEEK\_END),或从当前位置开始数的给定数量的字节上(SEEK\_CUR)。

命令rewind等价于fseek(f, 0L, SEEK\_SET)。

传递给fseek的任意字节定位只有在二进制模式下才有意义，因为在文本模式下可能存在未知的嵌入式字符。

函数ftell返回在一个文件中光标的当前位置，这个值可以被传递给fseek以返回到那个位置。(这种fseek和ftell的同步应用也可以在文本模式下使用)。

由于fseek和ftell为了文件定位而采用一个长整型参数，它们被自己能正确遍历文件的大小所限制。如果系统支持更大的文件定位值，可以通过库函数fgetpos和fsetpos随机地遍历这些文件，它们与fpos\_t类型的值进行交易，fpos\_t类型可能是整型值也可能不是。

## example6：处理有固定长度记录的文件

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAXRECS 10

struct record

{

char last[16];

char first[11];

int age;

};

static char\* get\_field(char \*, char \*);

int main()

{

int nrecs;

char s[81];

struct record recs[MAXRECS], recbuf;

FILE\* f;

// 小心地保存记录

for (nrecs = 0; nrecs < MAXRECS && get\_field("Last", s); ++nrecs)

{

strncpy(recs[nrecs].last, s, 15)[15] = '\0';

get\_field("First", s);

strncpy(recs[nrecs].first, s, 10)[10] = '\0';

get\_field("Age", s);

recs[nrecs].age = atoi(s);

}

// 将记录写入文件

if ((f = fopen("recs.dat", "w+b")) == NULL)

{

return EXIT\_FAILURE;

}

if (fwrite(recs, sizeof(recs[0]), nrecs, f) != nrecs)

{

return EXIT\_FAILURE;

}

// 定位于最后一条记录

fseek(f, (nrecs - 1) \* sizeof(struct record), SEEK\_SET);

fread(&recbuf, 1, sizeof(struct record), f);

printf("last: %s, first: %s, age: %d\n", recbuf.last, recbuf.first, recbuf.age);

// 定位于第一条记录

rewind(f);

fread(&recbuf, 1, sizeof(struct record), f);

printf("last: %s, first: %s, age: %d\n", recbuf.last, recbuf.first, recbuf.age);

return EXIT\_SUCCESS;

}

static char\* get\_field(char\* prompt, char\* buf)

{

// 为输入域提示

fprintf(stderr, "%s: ", prompt);

return gets(buf);

}