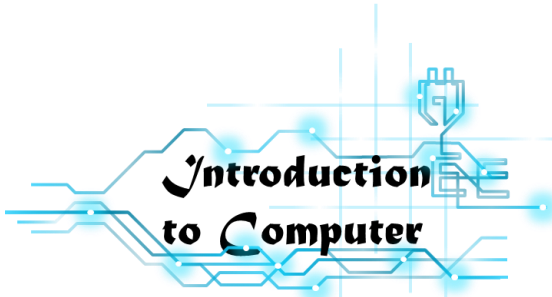
[C++] Loading Files

TA: 電子一 謝明倫

yans@media.ee.ntu.edu.tw

2014/03/01

臺灣大學



目錄

- 讀寫檔案模式
- C++ library 讀/寫檔方法
- 附註

讀寫檔案模式

- 除了讀檔(in)與寫檔(out)的分別之外，還另有一些模式。其中最重要的，是區分
  - Text(文字模式) ← default(預設選項)
  - Binary(原始資料)
- 差異:Text模式會視當前的處理系統對內容的解釋做**修改**，Binary模式則是純粹拿到原始資料。
  - 差異尤其發生在”換行”上, (可自行google: windows linux new line)
  - 所以好好一張圖選錯模式開可能 讀不到or多讀到 某些資料

- **基本原則: 非文字檔一率使用Binary模式讀/寫**

所以文字檔用text囉? 不一定...看狀況耶...

C++ library 讀/寫檔

- 利用fstream (ifstream讀, ofstream寫)
  - .open(const char* filename, ios_base::openmode mode)

constant	access
ios::in	File open for reading.
ios::out	File open for writing.
ios::binary	Operations are performed in binary mode rather than text.

```
#include<fstream>
…
ifstream pic1;
ofstream pic2;
```

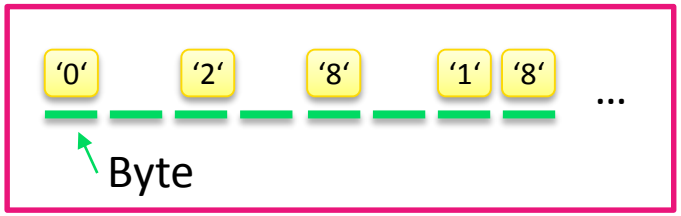
```
pic1.open(picPath,ios::in);//讀檔且text模式
pic1.open(picPath,ios::in|ios::binary);//讀檔且binary模式

pic2.open(picPath,ios::out);//寫檔且text模式
pic2.open(picPath,ios::out|ios::binary);//寫檔且binary模式
```

C++ library 讀/寫檔

- 開啟檔案後...
  - 讀(in)
    - **Text mode: operator>>**: 用法跟大家用cin時一模一樣
    - **Binary mode: .read(某個pointer, size)**:
      自檔案讀取size這麼多bytes到pointer裡面
      - 詳細說明 http://www.cplusplus.com/reference/istream/istream/read/
  - 寫(out)
    - **Text mode: operator<<**: 用法跟大家用cout時一模一樣
    - **Binary mode: .write(某個pointer, size)**:
      自pointer讀取size這麼多bytes寫到檔案裡面
      - 詳細說明http://www.cplusplus.com/reference/ostream/ostream/write/
- 執行完記得.close()以關閉檔案

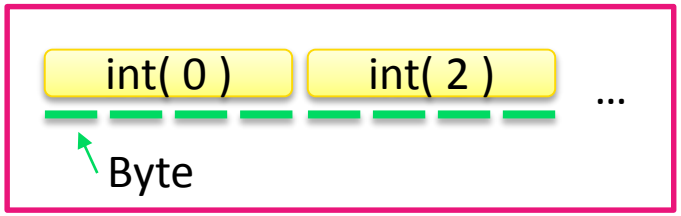
Example (Text)




```cpp
#include<iostream>
#include<fstream>
using namespace std;
int main(){
    ofstream fout;
    fout.open("file.xx",ios::out);
    for(int i=0;i<10;++i)
        fout << 2*i*i << ' ';
    fout.close();

    ifstream fin;
    fin.open("file.xx",ios::in);
    for(int i=0,temp;i<10;++i){
        fin >> temp;
        cout << temp << endl;
    }
    fin.close();
    return 0;
}
```

Example (Binary)




```cpp
#include<iostream>
#include<fstream>
using namespace std;
int main(){
    int arr[10],arr2[10];
    for(int i=0;i<10;++i) arr[i]= 2*i*i;
    ofstream fout;
    fout.open("file.xx",ios::out|ios::binary);
    fout.write( (char*)arr, 10*sizeof(int) );
    fout.close();

    ifstream fin;
    fin.open("file.xx",ios::in|ios::binary);
    fin.read( (char*)arr2, 10*sizeof(int) );
    fin.close();
    for(int i=0,temp;i<10;++i)
        cout << arr2[i] << endl;
    return 0;
}
```

整理

	讀檔(in)	寫檔(out)
Text	ifstream fin; fin.open(name, ios::in); fin>>	ofstream fout; fout.open(name, ios::out); fout<<

Binary	ifstream fin;
	fin.open(,ios::in|ios::binary);
	fin.read(pointer,size);
		ofstream fout;
		fout.open(,ios::out|ios::binary);
		fout.write(pointer,size);

當然也都要記得.close()

Q:如果open的時候mode混用了會怎麼樣?
A: =口= 可以自己試試看... 我猜是要看compiler的喜好...

附註

附註

- 看到4Bytes數字型資料請多利用fin.read((char*)p_uint,4)
  - 如果讀成4個char或unsigned char的話可能會有Little Endian問題
  - http://en.wikipedia.org/wiki/Little_Endian
- 看到2Bytes數字型資料請多利用fin.read((char*)p_ushort,2)
  - 除了第一筆是兩個char(非數字型)

Shift	*Name*	*Size*
0x00	Identifier (ID)	2
0x02	File Size	4
0x06	Reserved	4
0x0A	Bitmap Data Offset	4
0x0E	Bitmap Header Size	4
0x12	Width	4
0x16	Height	4
0x1A	Planes	2
0x1C	Bits Per Pixel	2

- 後面pixel資料用unsigned char array來接

- uint iarr[2] = {0x0A0B0C0D,0x1A1B1C1D};
  - 實際在Little Endian記憶體的存法是： 0D 0C 0B 0A 1D 1C 1B 1A
- uchar* pch=(uchar*)(iarr);
  - 注意pch[0]==='\x0D',pch[1]==='\x0C‘ 依此類推
- [錯誤] 所以如果read(pch,8);

```
uint jarr[2]={0,0};
for(int k=0;k<2*4;++k){
        jarr[k/4] = jarr[k/4]<<4;
        jarr[k/4] = jarr[k/4]&pch[k];
} //jarr變成{0x0D0C0B0A,0x1D1C1B1A}
```

- [正確] uint* jarr=new uint[2]; read((char*)jarr,8);
  - jarr[0]===0x0A0B0C0D;
  - 有點類似 jarr= (uint*)(pch=(uchar*)iarr); 的感覺
  - 當然如果[錯誤]作法裡面手動shift順序改一下也是會ok的

附註

- C library請自行參考↓本文不說明之
  - http://www.cplusplus.com/reference/cstdio/?kw=cstdio
  - binary讀寫檔案請在fopen的mode裡多加”b”