# Lecture Notes on
# C++ Multi-Paradigm Programming

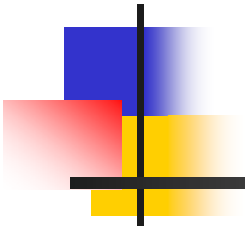
***Bachelor of Software Engineering, Spring 2013***

**Wan Hai**

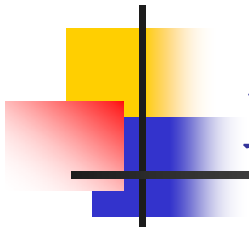
*whwanhai@163.com*

*13512768378*

**Software School, Sun Yat-sen University, GZ**



# IO and File



# 主要内容

- I/O的基本概念
- 文本文件I/O流
- 二进制文件I/O

# 1、何为I/O?

# 2、应用程序、操作系统与I/O


- 在现代通用计算机的架构中，I/O指令属于特权指令，只能由操作系统发出，不能由用户程序发出。
- 用户程序要进行I/O必须利用操作系统提供的接口----系统调用（system calls）。
- 由于系统调用接口的层次太低，用户程序直接使用系统调用不方便，所以高级程序语言编程时一般使用标准库中提供的、更高层次的I/O机制。
- 在C++中，流类就是用于进行I/O操作的高级机制。

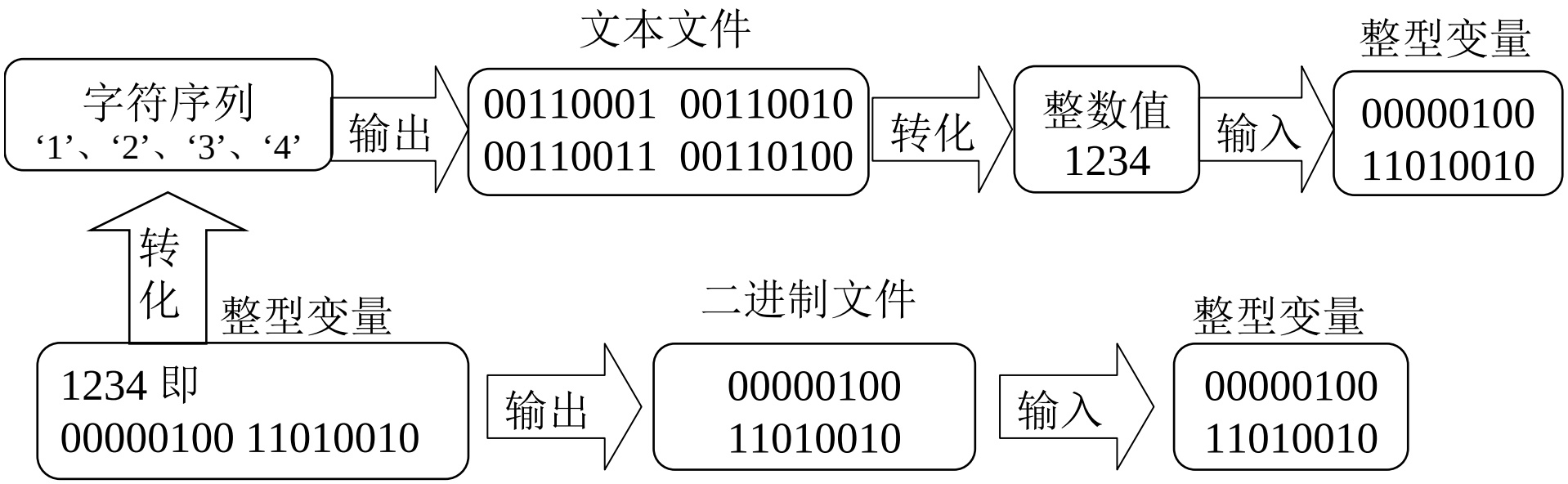
# 3、文本文件I/O Vs. 二进制文件I/O

- 假设有一整型变量，值为**1234**。假设以**2**字节存储整型变量，则该变量在内存中的形式为

  **00000100 11010010**

  现要求输出这个值到文件中保存起来，然后再从这个文件中读取这个值到内存中。

# 3、文本文件I/O Vs. 二进制文件I/O



# 3、文本文件I/O Vs. 二进制文件I/O

- 文本文件输出，需要先将输出值转化为字符序列，然后存储到文件中的内容是这些字符相应的ASCII码。文本文件输入，也需要先将字符序列转化为相应的数值，然后再进入变量中。
- 二进制文件输出，是直接将内存数值（即其二进制形式）输出到文件中保存。因此得到二进制文件，且其内容与内存的形式一样。二进制文件输入，也是直接将文件内容输入到内存中即可，无需转化过程。

# 3、文本文件I/O Vs. 二进制文件I/O

- 除非要输出的内容就是字符，否则保存到文本文件中的内容与内存形式是不同的，而二进制文件的内容则与内存形式相同。
- 一般说来，二进制文件比文本文件体积要小。
- 二进制文件I/O省去了“转化”这道工序，大大节约了文件I/O的时间。
- 二进制文件I/O是文件I/O的主流。
- 文本文件I/O有一个优点：可以很方便的打开文本文件、可以直接看到结果，因此一般用于保存一些简单的结果数据方便查阅。

# 4、文本文件流—例子1

- 例子：
- 先准备好一个纯文本文件**source.txt**，保存在与下面的**FileIO.cpp**同一个文件夹中，其内容为：

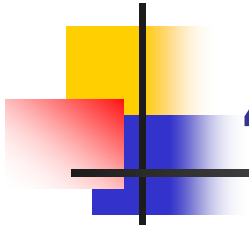
**13 3.14 9**

- 然后编写一程序，把这个文件的内容读入到一个**int**变量、一个**double**变量和**char**变量中。这样这三个变量的值应该分别为**13**、**3.14**、**'9'**。计算前两者之积，然后把结果输出到文件**result.txt**中保存起来。

# 4、文本文件流----例子1

```
//FileIO.cpp                    Chap10_1
//功能：说明利用文件流实现文件I/O
#include <fstream>                    //①
using namespace std;

int main( )
{
    int someInt;
    float someFloat;
    char someChar;
    ifstream  inFile;                 //②
    ofstream  outFile;                //②
    inFile.open("Source.txt");        //③
    outFile.open("Result.txt");       //④
                    :
```

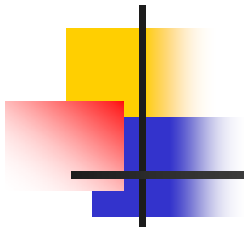
```
inFile >>  someInt
       >> someFloat
       >> someChar;          //⑤

outFile <<  "The answer is: "
        <<someInt*someFloat
        <<endl;              //⑥

inFile.close();              //⑦
outFile.close();             //⑦


return 0;
}
```



# 4、运行结果

- **someInt\someFloat\someChar**的值将分别为**13\3.14\’9’**。
- 在**FileIO.cpp**同一目录下，出现一个新的文件：**result.txt**。将其打开，将看到：

```
The answer is: 40.82
```



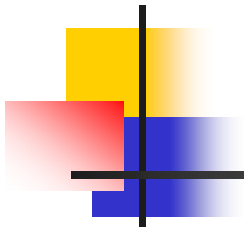
# 4、解释

```
#include <fstream>
```

**Step1:** 由于类**ifstream**和**ofstream**定义在头文件**fstream**中，所以在文件头需要加上预编译指令。

```
ifstream  inFile;
ofstream  outFile;
```

**Step2:** 定义两个对象**inFile**和**outFile**，也称为文件流。前者负责文件输入，后者负责文件输出。



# 4、解释

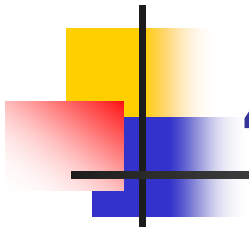
**Step3: open**语句是把文件流与具体的文件相关联起来，使得后续的具体的读写操作作用于这些文件之上。

```
inFile.open("Source.txt");
```

把输入文件流**inFile**与文件**source.txt**关联起来，后面从**inFile**输入数据便是从文件**Source.txt**里读取数据；

```
outFile.open("Result.txt");
```

把输出文件流**outFile**与文件**result.txt**关联起来，后面输出的结果放入**outFile**中最终就会保存到**result**里面。



# 4、解释

**Step4:** 具体的输出输入语句

```
inFile >>  someInt >> someFloat >> someChar;
```

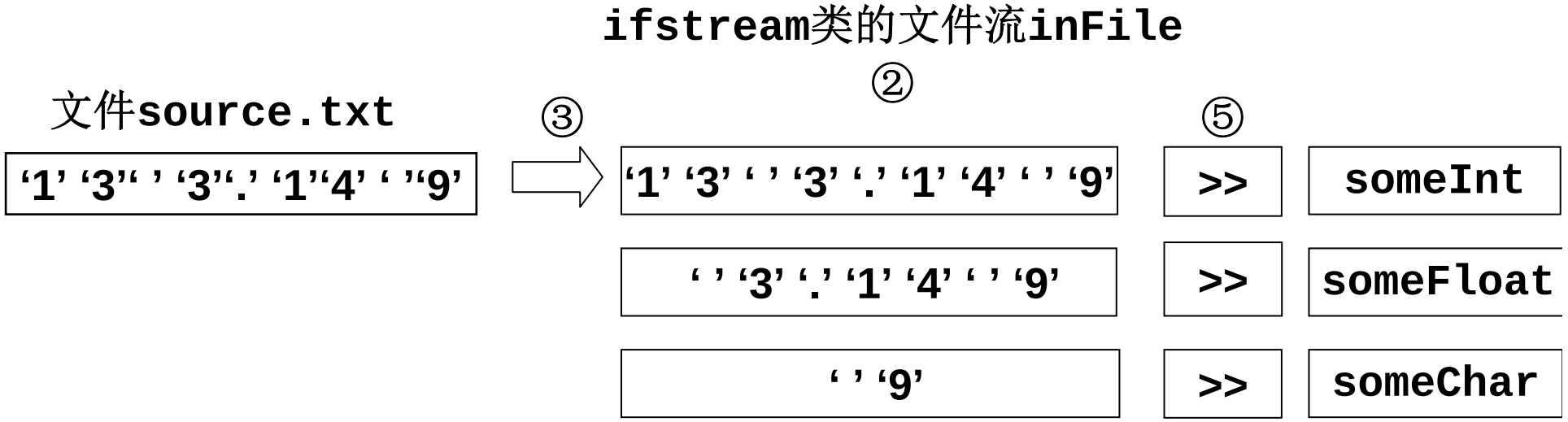
从输入文件流**inFile**中读取数据，置入变量中

```
outFile <<  “The answer is: ” <<someInt*someFloat
        <<endl;
```

把要输出的内容放入输出文件流里面，最终保存到文件**result.txt**里面

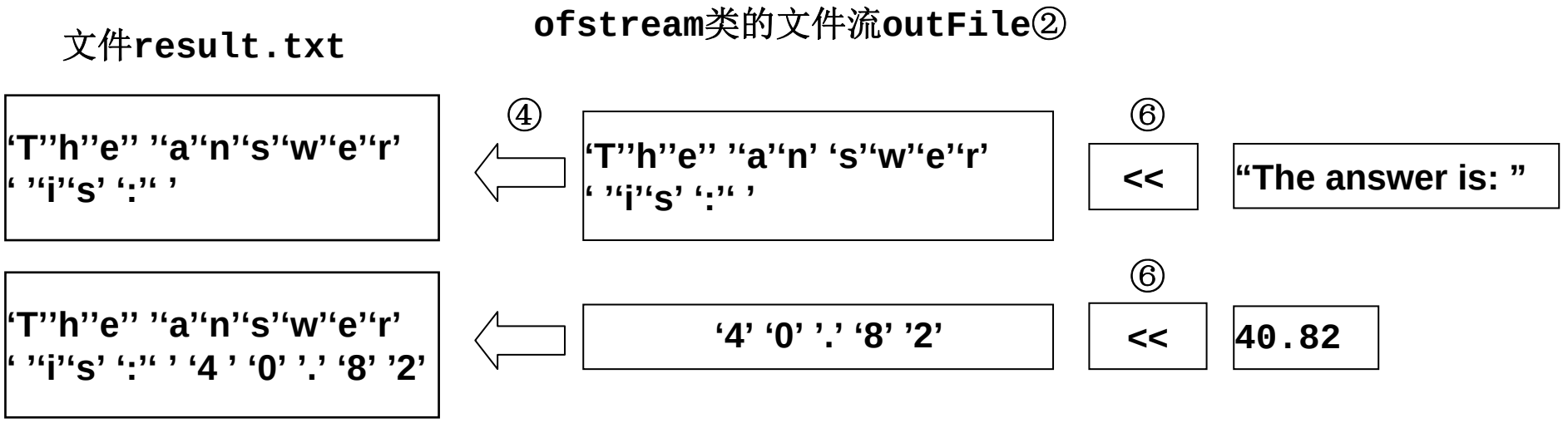
# 4、图解

**inFile >> someInt >> someFloat >> someChar;**



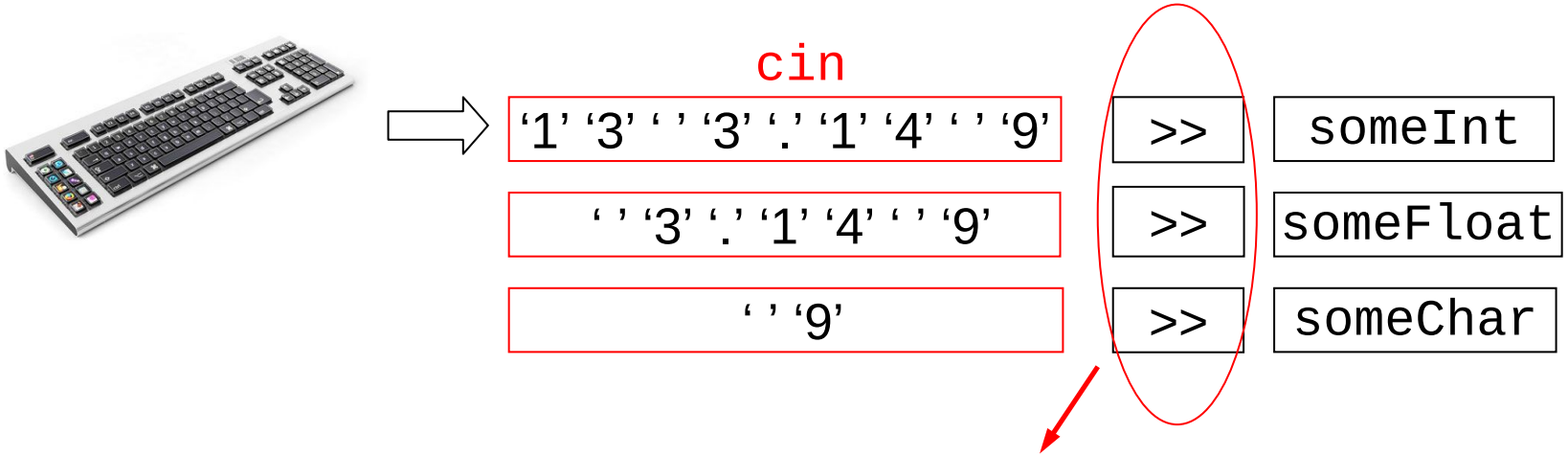
# 4、图解

```
outFile << “The answer is: ” <<someInt*someFloat
        <<endl;
```



# 4、与键盘输入、显示器输出的比较

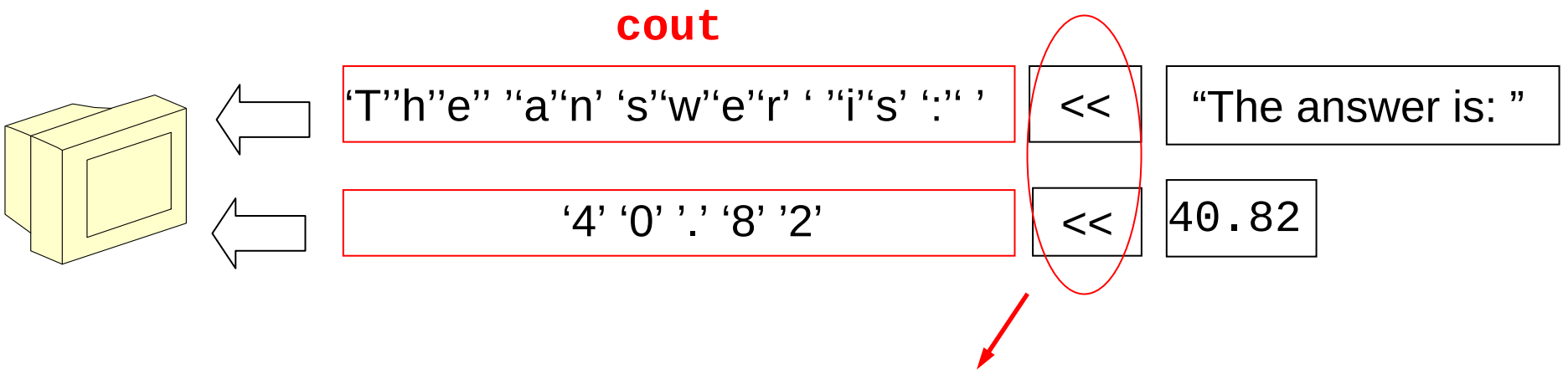
```
cin >> someInt >> someFloat >> someChar;
                    //假设键盘中敲入13 3.14 9↙
```



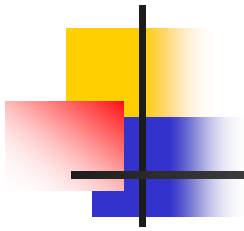

**cin**缺省情况下是装载字符的数据流，输入符”**>>**”负责转化的工作。

# 4、与键盘输入、显示器输出的比较

**cout << “The answer is: ” << someInt*someFloat;**




**cout**是只能装载字符的数据流，输入符”**<<**”要负责转化的工作。

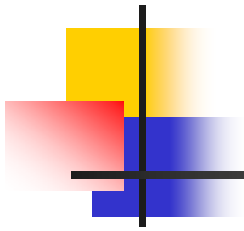


# 5、文本文件流----例子2

- 假设已有纯文本文件**1.txt**，现编一程序**copy**，将**1.txt**拷贝到另一个纯文本文件**2.txt**中。

# 5、文本文件流----例子2

```
//Chap10_2
#include <fstream>
using namespace std;

int main( )
{
    char c;
    ifstream  inFile;

    ofstream  outFile;


    inFile.open("1.txt");
    outFile.open("2.txt");
              :
    inFile >> c;
    while( inFile )
    {
        outFile << c;
        inFile >> c;
    }

    inFile.close();
    outFile.close();

    return 0;
}
```



# 5、文本文件流----例子2

- 运行程序，结果如何？出现什么问题？
- 为什么会出现这样的问题？
- 如何解决这个问题？

# 5、文本文件流----例子2

```
//Chap10_3
#include <fstream>
using namespace std;

int main( )
{
    char c;
    ifstream  inFile;

    ofstream  outFile;


    inFile.open("1.txt");
    outFile.open("2.txt");
              :
    inFile.get(c);
    while( inFile )
    {
        outFile << c;
        inFile.get(c);
    }

    inFile.close();
    outFile.close();

   return 0;
}
```

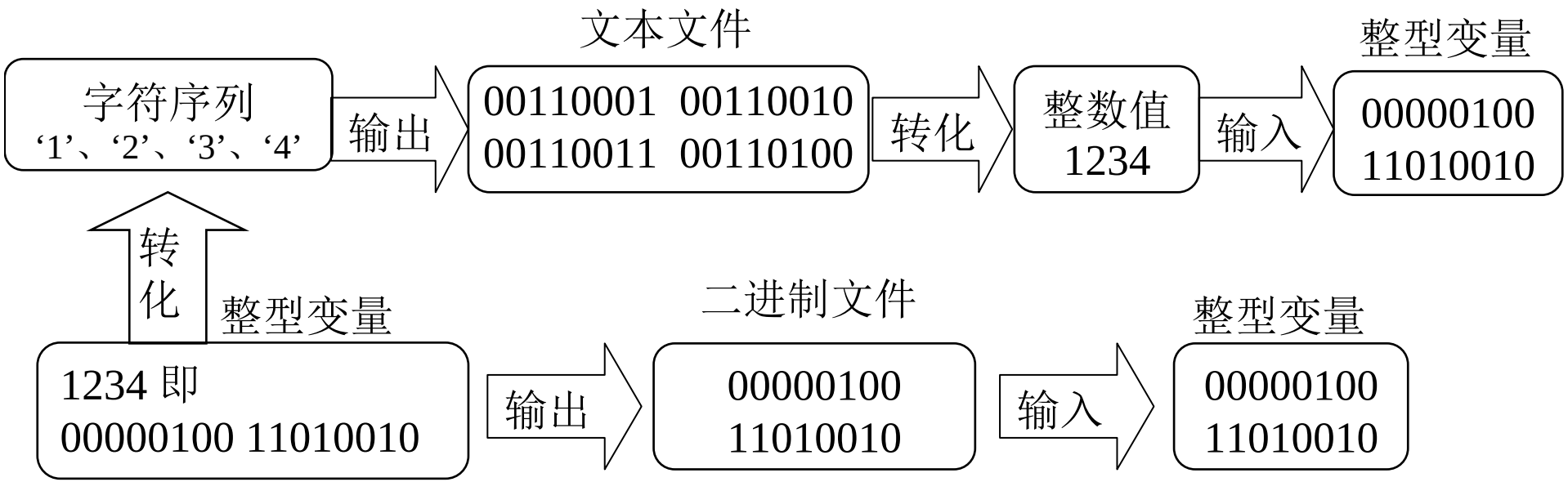
# 6、文本文件I/O Vs. 二进制文件I/O

- 假设有一整型变量，值为**1234**。假设以**2**字节存储整型变量，则该变量在内存中的形式为

  **00000100 11010010**

  现要求输出这个值到文件中保存起来，然后再从这个文件中读取这个值到内存中。

# 6、文本文件I/O Vs. 二进制文件I/O

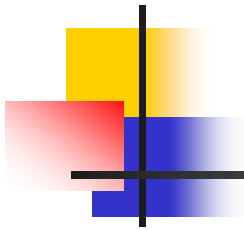


# 6、文本文件I/O Vs. 二进制文件I/O

- 文本文件输出，需要先将输出值转化为字符序列，然后存储到文件中的内容是这些字符相应的**ASCII**码。文本文件输入，也需要先将字符序列转化为相应的数值，然后再进入变量中。
- 二进制文件输出，是直接将内存数值（即其二进制形式）输出到文件中保存。因此得到二进制文件，且其内容与内存的形式一样。二进制文件输入，也是直接将文件内容输入到内存中即可，无需转化过程。

# 6、文本文件I/O Vs. 二进制文件I/O

- 除非要输出的内容就是字符，否则保存到文本文件中的内容与内存形式是不同的，而二进制文件的内容则与内存形式相同。
- 一般说来，二进制文件比文本文件体积要小。
- 二进制文件I/O省去了“转化”这道工序，大大节约了文件I/O的时间。
- 二进制文件I/O是文件I/O的主流。
- 文本文件I/O有一个优点：可以很方便的打开文本文件、可以直接看到结果，因此一般用于保存一些简单的结果数据方便查阅。



# 7、二进制文件I/O

- C：

1）FILE、fopen、fwrite、fread、fclose的用法

2）文件的定位

- C++：

1）利用ifstream/ofstream流

2）文件的定位

# 8、二进制文件I/O(C)

```
//C_File_binary_IO
#include <iostream>
using namespace std;

int main()
{
   int a[10]={1,2,3,4,5,6,7,8,9,10};
   int b[10]={0};
   int i;
   FILE* fp1, *fp2;

   fp1=fopen( "c:\\1.dat", "wb" );
   fwrite(a, sizeof(int), 10, fp1);
   fclose( fp1 );
                  :
```

```
                  :
   fp2=fopen( "c:\\1.dat", "rb" );
   fread( b, sizeof(int), 10, fp2 );
   fclose( fp2 );

   for ( i = 0; i < 10; i++)
         cout << b[i] << "  " ;

   cout << endl << endl;

   return 0;
}
```

# 8、运行结果

- 运行这段程序现屏幕将显示：

**1 2 3 4 5 6 7 8 9 10**

并且在**C**盘下将出现一个文件**1.dat**。

- 这表明数组**a**各元素已写入了这个文件中，且在程序中又将这个文件的内容读取到**b**数组里面。

# 8、解释

```
FILE *fp;
fp = fopen( 文件名， 文件使用方式 );
```

1.定义文件指针fp。

2.利用fopen把fp和某一文件相关联，即将fopen所返回的指针赋给fp，令fp指向该文件。

| 文件使用方式 | 含义 | 文件使用方式 | 含义 |
|---|---|---|---|
| “r”（只读） | 为输入打开一个文本文件 | “r+”（读写） | 为读/写打开一个文本文件(要求文件已存在) |
| “w” | 为输出打开一个文本文件 | “w+”（读写） | 为读/写建立一个新的文本文件 |
| “a” | 向文本文件尾追加数据 | “a+”（读写） | 为读写打开一个文本文件 |
| “rb” | 为输入打开一个二进制文件 | “rb+”（读写） | 为读/写打开一个二进制文件 |
| “wb” | 为输出打开一个二进制文件 | “wb+”（读写） | 为读/写建立一个新的二进制文件 |
| “ab” | 向二进制文件尾追加数据 | “ab+”（读写） | 为读写打开一个二进制文件 |

# 8、解释

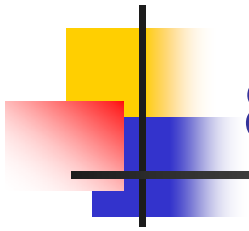
```
FILE *fp;
fp = fopen( 文件名，  文件使用方式 );
```

- 若要写的文件不存在，则将先创建这个文件，然后再往这个文件里面写入数据。
- **“w”:**若要写的文件已存在，则先清空文件的原本内容，然后再写入数据。
- **“a”:**若要写的文件已存在，不清除文件的原本内容，从文件尾续加新数据。
- 如果希望该文件既能读也能写，那么就要带**+**号。

# 8、更多时候，需要判断

- 打开动作并非一定成功。打开失败时**fopen**就不会返回一个文件指针，而是返回**NULL**。

```
if( fp=fopen( 文件名， 文件使用方式 ) == NULL )
{
    ..... //打开文件失败的处理
}
else
{
    ..... //打开文件成功的处理
}
```



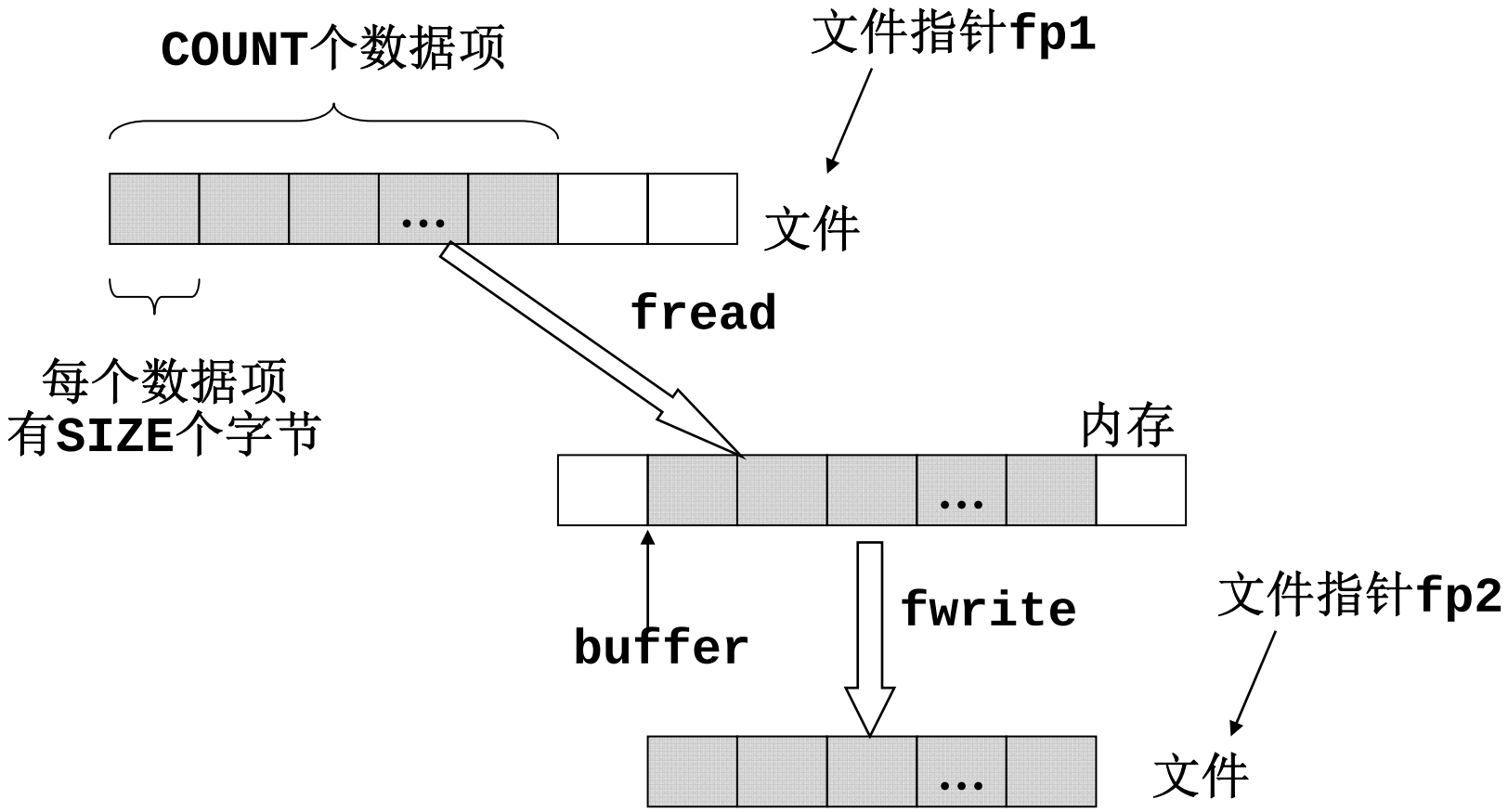
# 8、解释

```
fread( b, sizeof(int), 10, fp1 );

fwrite(a, sizeof(int), 10, fp2);
```

```
fread( buffer, size, count, fp1 );
fwrite( buffer, size, count, fp2 );
```

# 8、解释

```
fread( buffer, size, count, fp1 );
fwrite( buffer, size, count, fp2 );
```



# 8、解释

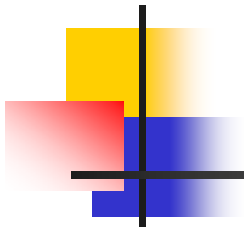
```
fread( buffer, size, count, fp1 );
fwrite( buffer, size, count, fp2 );
```

- 读和写的对象是与**fp1**和**fp2**相关联的文件。这个操作必须与**fopen**中指定的文件使用方式一致。
- **buffer**是指针，指出数据在内存中的起始地址（对读操作，它表示数据输入内存后存放的地址；对写操作，它表示要输出的数据在内存中的存放地址）。
- 总共要读/写**COUNT**个数据项，而每个数据项的大小为**SIZE**个字节。因此总共读/写的数据大小为**COUNT*SIZE**个字节。

# 9、更多时候，需要判断

- **fwrite**返回实际写入（输出）的数据项的个数，而**fread**返回实际读入（输入）的数据项的个数。
- 由于硬件的原因，有时读/写操作并非一定完全成功。因此程序中最好通过其返回值来进行判断。

```
if( fwrite(a, sizeof(int), 10, fp1) != 10 )
{ ... }        //写操作发生错误的处理
else
{ ... }        //写操作成功的处理
```



# 10、文件的定位

- 默认情况下，对文件的读写是按顺序从头到尾进行的。

- 我们可以认为有一个位置指针，它指出了在文件中读写的位置。这个指针每读/写完一个数据项后，就自动移动到下一个位置上。

- 有时，我们可能需要控制这个指针的位置，以一种我们自己设计的顺序来读/写文件。这就是文件的定位。

- 可以利用**fseek**函数进行文件定位。

# 10、文件的定位

1 2 3 4 5 6 7 8 9 10

- 假设修改上例，使得隔个读取**1.dat**里面的数值，即只希望读取**1**、**3**、**5**、**7**、**9**。

**Tips:**

在读取了**1**后，位置指针自动移动到**2**的位置上，这时就应该调用**fseek**改变位置指针使其移动到**3**的位置上，便可以略过**2**。

# 10、文件的定位

```
                :  //此处代码与上例同
fp2=fopen( "c:\\1.dat", "rb" );

for( i = 0; i < 5; i++ )
{
    fread( &b[i], sizeof(int), 1, fp2 );
    fseek( fp2, sizeof(int), SEEK_CUR );
}
fclose( fp2 );

for ( i = 0; i < 10; i++)
     cout << b[i] << "  " ;
            : //此处代码与上例同
}
```

1 2 3 4 5 6 7 8 9 10

运行程序屏幕将显示：

1 3 5 7 9 0 0 0 0 0

# 10、解释

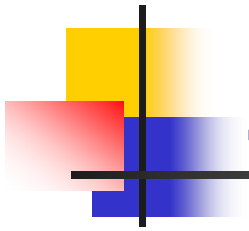
**fseek( 文件指针，位移量(字节数)，起始点 );**

- 使得位置指针从当前位置移动到距离“起始点”为“位移量”的那个位置上。接着的读写将从那个位置开始。
- 起始点可以有

· **SEEK_SET** 表示 文件开始

· **SEEK_CUR**表示 当前位置

· **SEEK_END**表示 文件末尾



# 10、注意

- 利用**fopen**打开文件，若不是追加形式（没有**a**），则位置指针将处于文件头。
- 若带有“**a**”，则位置指针将处于文件尾。每次读写操作后，位置指针将自动移动到下一个位置。
- 进行定位的时候，就必须小心计算好相应于起始点要移动多少个字节，不要犯下多移少移的错误。

# 11、二进制文件I/O(C++)

```
#include <fstream>                                          //Chap10_4
using namespace std;

int main(  )
{
      int a[10]={10,20,30,40,50,60,70,80,90,100};
      int b[10],  i;
      ifstream inFile;                                      //①
      ofstream outFile;                                     //①

      outFile.open( "c:\\1.dat", ios::binary );             //②
      for( i = 0; i < 10; i++ )
    {
          outFile.write( (char*)&a[i], sizeof( a[i] ) );    //③
     }
      outFile.close();                                      //④
```

# 11、二进制文件I/O(C++)

```
    inFile.open(  "c:\\1.dat", ios::binary );          //②
    for( i = 0; i < 10; i++ )
    {
       inFile.read(  (char*)&b[i], sizeof( &b[i] ) ); //⑤
    }
    inFile.close();                                    //④

    for ( i = 0; i < 10; i++)
       cout << b[i] << "  " ;

    cout << endl << endl;

    return 0;
}
```

# 11、 open函数原型

**ifstream的open**

```
void open( const char* szName,
           int nMode = ios::in,   //ios::in表示文本方式输入
           int nProt = filebuf::openprot );
```

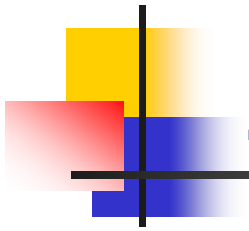
**ofstream的open**

```
void open( const char * szName,
           int nMode = ios::out,   //ios::out表示文本方式输出
           int nProt = filebuf::openprot );
```

**//nMode表示文件输入或输出的方式：二进制还是文本，输入清零还是追加。。。。。所以，有时实参需要“组合”。**

# 11、二进制输出且追加

```
int main( )
{
      int a[10]={10,20,30,40,50,60,70,80,90,100};
      int b[10];
      int i;
      ifstream inFile;
     ofstream outFile;

      outFile.open( "c:\\1.dat",  ios::binary|ios::app  );
      for( i = 0; i < 10; i++ )
    {
       outFile.write( (char*)&a[i], sizeof( a[i] ) );
     }
      outFile.close();
                                        :
```



# 12、文件的定位

文件流.**Fseekg(** 位移量(字节数)，起始点 **)**;

起始点：**ios::beg**

**ios::cur**

**ios::end**

# 12、文件的定位

```
#include <iostream>                                    //Chap10_5
using namespace std;
int main()
{
   int a[10]={10,20,30,40,50,60,70,80,90,100};
  int b[10];
  int i;
  ifstream inFile;
  ofstream outFile;

  outFile.open( "c:\\1.dat", ios::binary );                    //②
  for( i = 0; i < 10; i++ )
  {
      outFile.write( (char*)&a[i], sizeof( a[i] ) );
  }
  outFile.close();                                             //④
```

# 12、文件的定位

```
    inFile.open(  "c:\\1.dat", ios::binary );          //①

    for( i = 0; i < 10; i++ )
    {
        inFile.read(  (char*)&b[i], sizeof( &b[i] ) ); //②
        inFile.seekg( sizeof(int), ios::cur );         //③
    }

    inFile.close();                                    //④

    for ( i = 0; i < 10; i++)
        cout << b[i] << "  " ;

    return 0;
}
```

```
10 20 30 40 50 60 70 80 90 100
```

# 13、动态输入文件名

```
#include <fstream>                                          //c_str
#include <string>
using namespace std;

int main(  )
{
      int a[10]={10,20,30,40,50,60,70,80,90,100};
      int i;
      ifstream inFile;
      string FileName;

      cout  << “Please enter the output file name”;
      cin    >> FileName;
      outFile.open(  FileName.c_str( ), ios::binary );
                              :
}
```