北京工業大學

BEIJING UNIVERSITY OF TECHNOLOGY

C++的流输入和输出

endl	插入一个换行符
ends	插入一个空字符

fixed	让流以**定点表示法**显示数据
scientific	让流以**科学表示法**显示数据

dec	让流以十进制方式解释输入或显示输出
hex	让流以十六进制方式解释输入或显示输出
oct	让流以八进制方式解释输入或显示输出

setprecision	设置小数精度
setw	设置字段宽度
setfill	设置填充字符
setbase	设置基数，与使用 dec、 hex 或 oct 等效
setiosflag	通过类型为 std::ios_base::fmtflags 的掩码输入参数设置标志
resetiosflag	将 std::ios_base::fmtflags 参数指定的标志重置为默认值

setiosflags是 C++ 标准库 <iomanip>中的一个函数，用于**设置输出流的格式标志**。
Setiosflags 函数接受一个 std::ios_base::fmtflags类型的参数，这个参数是你想要设置的标志的**位掩码**。`std::ios_base::fmtflags 是一个位字段类型，用于表示各种格式标志。

以下是一些常用的格式标志：
std::ios_base::dec： 设置输出格式为十进制。
std::ios_base::oct： 设置输出格式为八进制。
std::ios_base::hex： 设置输出格式为十六进制。
std::**ios_base::showbase**： 设置输出格式为显示基数。例如，**十六进制数会以 0x开头**。
std::ios_base::uppercase： 设置输出格式为大写。例如，十六进制数会使用大写字母。你可以使用位或运算符 | 来组合多个标志。
例如，setiosflags(std::ios_base::hex | std::ios_base::showbase)会设置输出格式为十六进制，并显示基数。
setiosflags函数返回一个 std::ios_base&类型的值，这个值可以用于**链式调用**。

std::cout << setiosflags(std::ios_base::hex) << 255;会以十六进制的形式输出 255。

第一部分

CIN & COUT 流

cout << “Enter a line: ” << endl;
char charBuf[10] = {0}; 缓冲区
cin.get(charBuf, 9); // stop inserting at the 9th character

尽可能不要使用 char 数组：只要可能，就应使用 std::string 而不是 char*。

要读取整行输入（包括空白），需要使用 getline( )：
string name;
getline(cin, name);

>> 是会过滤掉不可见字符（如 空格 回车，TAB 等），不想略过空白字符，那就使用 **noskipws 流控制**：cin>>noskipws>>str; 此时输入：空格 test，会输出空格

int cin.get();

istream& cin.get(char& var);

istream& get (char* s, streamsize n);

istream& get (char* s, streamsize n, char delim);

cin.get(array,20);读取一行时，遇到换行符时结束读取，但是不对换行符进行处理，**换行符仍然残留在输入缓冲区**。

第二次由cin.get()将换行符读入变量b，打印输入换行符的ASCII码值为10。这也是cin.get()读取一行与使用getline读取一行的区别所在。

getline读取一行字符时，默认遇到’**\n’时终止**，并且将’**\n’直接从输入缓冲区中删除掉**，不会影响下面的输入处理。

cin.get(str,size);读取一行时，只能将字符串读入C风格的字符串中，即char*，但是C++的getline函数可以将字符串读入C++风格的字符串中，即string类型。

istream& getline(char* s, streamsize count); //默认以换行符结束

istream& getline(char* s, streamsize count, char delim);

```
#include <iostream>
using namespace std;
int main()
{
   char array[20]={0};
   // cin.getline(array,20); //或者指定结束符，使用下面一行
    cin.getline(array, 20, '\n' );
   cout<<array<<endl;
   return 0;
}
```

```cpp
#include <iostream>
int main() {
    int x;
    std::cin >> x;
    if (std::cin.good()) {
        std::cout << "Read successfully: " << x << std::endl;
    } else if (std::cin.eof()) {
        std::cout << "End of file encountered" << std::endl;
    } else if (std::cin.fail()) {
        std::cout << "Non-numeric character entered" << std::endl;
        std::cin.clear();  // Clear the error state
    } else if (std::cin.bad()) {
        std::cout << "Serious error occurred" << std::endl;
    }
    return 0;
}
```

std::cin 是 std::istream 对象，用于从标准输入读取数据。std::istream 对象有一些成员函数可以用来检查其状态：

eof()：如果在读取操作中遇到文件结束符（EOF），则返回 true。

fail()：如果在读取或格式化操作中发生错误（例如，试图读取一个非数字字符到 int 类型的变量），则返回 true。

bad()：如果在读取操作中发生严重错误（例如，读取硬件错误），则返回 true。

good()：如果流处于有效状态，即没有发生任何错误，则返回 true。

第二部分

Fstream 文件流

要使用 fstream、 ofstream 或 ifstream 类，需要使用方法 open( )打开文件：

```
fstream myFile;
myFile.open("HelloFile.txt",ios_base::in | ios_base::out | ios_base::trunc);

if (myFile.is_open())        // check if open() succeeded
{

myFile.close();

}
```

```
Human Input("Siddhartha Rao", 101, "May 1916");
ofstream fsOut ("MyBinary.bin", ios_base::out | ios_base::binary);

if (fsOut.is_open())
{
cout << "Writing one object of Human to a binary file" << endl;
fsOut.write(reinterpret_cast<const char*>(&Input), sizeof(Input)); //write函数需要char*
fsOut.close();
}

ifstream fsIn ("MyBinary.bin", ios_base::in | ios_base::binary);
if(fsIn.is_open())
{
Human somePerson;
fsIn.read((char*)&somePerson, sizeof(somePerson));

cout << "Reading information from binary file: " << endl
cout << "Name = " << somePerson.name << endl;
cout << "Age = "  << somePerson.age << endl;
cout << "Date of Birth = " << somePerson.DOB << endl;
}
```

```
struct Human
{
    Human() {};
    Human(const char* inName, int inAge, const char* inDOB) : age(inAge)
    {
        strcpy(name, inName);
        strcpy(DOB, inDOB);
    }
    char name[30];
    int  age;
    char DOB[20];
};
```

在 C++ 中使用标准库中的 **remove函数来删除文件。**
这个函数在 **<cstdio>**头文件中定义。

```
#include <cstdio>

int main(){
  if (remove("data.txt") == 0)
        {std::cout << "File deleted successfully.\n";    }
 else
        {std::cout << "Failed to delete the file.\n";    }
  return 0;
}
```

在这个示例中，尝试删除名为 "data.txt" 的文件。remove 函数会尝试删除指定的文件
如果成功，它会返回 0；否则，它会返回一个非零值。

```cpp
#include <fstream>
#include <iostream>

int main()
{
    // 使用 fstream 写入文本文件
    std::fstream file("data.txt", std::ios::out);
    if (file.is_open())
    {
        file << "Hello, World!";
        file.close();
    }
    else
    {
        std::cout << "Unable to open file for writing.\n";
    }

    // 使用 fstream 读取文本文件
    file.open("data.txt", std::ios::in);
    if (file.is_open())
    {
        std::string line;
        while (std::getline(file, line))
        {
            std::cout << line << '\n';
        }
        file.close();
    }
    else
    {
        std::cout << "Unable to open file for reading.\n";
    }

    return 0;
}
```

```cpp
#include <sstream>
#include <iostream>
int main() {
    std::stringstream ss;
    ss << "Age: " << 20;
    std::cout << ss.str() << std::endl;  // 输出 "Age: 20"
    return 0;
}
```

```cpp
int main() {
    std::stringstream ss("Hello, World!");
    std::string word;
    while (ss >> word) {
        std::cout << word << std::endl;  // 分别输出 "Hello," 和 "World!"
    }
    return 0;
}
```

```cpp
int main() {
    // 数字转字符串
    int num = 123;
    std::stringstream ss;
    ss << num;
    std::string str = ss.str();
    std::cout << str << std::endl;  // 输出 "123"

    // 字符串转数字
    ss.str("456");
    ss.clear();
    ss >> num;
    std::cout << num << std::endl;  // 输出 456

    return 0;
}
```

1.// 清空 sstream

2.Sstream. Str(“ “);

3.sstream << "third string";

cout << "After clear, strResult is: " << sstream.str() << endl;

```cpp
#include <sstream>
#include <iostream>

using namespace std;

int main()
{
    stringstream sstream;
    int first, second;

    // 插入字符串
    sstream << "456";
    // 转换为int类型
    sstream >> first;
    cout << first << endl;

    // 在进行多次类型转换前，必须先运行clear()
    sstream.clear();

    // 插入bool值
    sstream << true;
    // 转换为int类型
    sstream >> second;
    cout << second << endl;

    return 0;
}
```

在 C++ 中，clear是 std::stringstream`类的一个成员函数，用于**重置流的状态标志**。代码片段中：sstream.clear();这行代码清除了 sstream的错误状态标志。

这意味着，如果之前的操作导致了错误状态（例如，试图从空的流中读取数据），这行代码将会清除这些错误状态，使得流回到良好状态。

在进行流操作（如读取或写入）之前，通常需要确保流是处于良好状态的。如果流处于错误状态，后续的读写操作可能会失败。因此，调用 clear函数是一个好的实践，可以确保流处于可以进行后续操作的状态。

string类型的字符串是不以‘\0’结尾的，因此若str有三个字符，传统C语言的字符串的str[3]是字符‘\0’，但是string类型的只能到str[2]，str[3]是没有定义的，而str. at(3)会提示越界奔溃。

功能	C++ string	C字符数组
定义字符串	string str;	char str[100];
单个字符输出	str[i] / str.at(i)	str[i]
字符串长度	str.length() / str.size()	strlen(str)
读取一行	getline(cin,str)	gets(str)
赋值	str = “Hello”;	strcpy(str,“Hello”);
连接字符串	str = str + “Hello”	strcat(str,“Hello”);
比较字符串	str == “Hello”;	strcmp(str,“Hello”);

c_str()	返回一个以‘/0’结尾的字符数组
data()	以字符数组的形式返回字符串内容，但并不添加’/0’
copy()	字符串的内容复制或写入既有的c_string或字符数组内

```
string str = "Hello World!";
const char* p1 = str.c_str();
const char* p2 = str.data();
const char* p3=new char[10];
str.copy(p3,5,0);
//函数原型：copy(char *s, int n, int pos = 0)
//把当前串中以pos开始的n个字符拷贝到以s为起始位置的字符数组中，返回实际拷贝的数目
```

string还可以方便的改变字符串的容量大小，通过调用成员函数resize()可以重设string的容量。string str="Hello";
str.resize(3);

string可以很方便的查找字符串中的字符或者子串，其是通过成员函数find()和substr()实现的，find()函数是从str第3个位置查起，找到子串后，返回子串的位置；而substr函数从pos位置(子串开始的位置)开始，截取5个字符，赋值给str2，也就是说，str2的内容将是ssdfs。

```
string str = "aaaaddddssdfsasdf";
 size_t pos = str.find("ssdf", 3);   //注意pos的数据类型string::size_type//如果没找到，返回一个特殊的标志npos

if(pos != string::npos)则表示找到。
string str2 = str.substr(pos, 5);
```

```
// 忽略一个字符
std::cin.ignore();

// 忽略 5 个字符
std::cin.ignore(5);

// 忽略直到遇到换行符std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

std::numeric_limits<std::streamsize>::max() 来指定一个非常大的数，这样 ignore 函数将忽略所有字符，直到遇到换行符。这对于清除输入流中剩余的字符（例如，用户输入了多余的字符）是非常有用的。

cin和cout解锁

代码（写在main函数开头）：

```
ios::sync_with_stdio(false);
cin.tie(0),cout.tie(0);
```

注意：cin cout解锁使用时，不能与 scanf,getchar, printf,cin.getline()混用，一定要注意，会出错。

这两行代码通常在 C++ 程序中使用，以提高 I/O 效率。

ios::sync_with_stdio(false);：这行代码用于取消 C++ 标准流（如 cin 和 cout）与 C 标准流（如 scanf 和 printf）之间的同步。默认情况下，这两种类型的流是同步的，以确保它们可以混合使用并且结果是正确的。但是，这种同步会导致一些额外的性能开销。如果你的程序只使用 C++ 标准流，或者只使用 C 标准流，你可以取消这种同步，以提高 I/O 效率。

cin.tie(0),cout.tie(0);：这两行代码用于取消 cin 和 cout 之间的绑定。默认情况下，cin 和 cout 是绑定的，这意味着每次从 cin 读取数据之前，都会先刷新 cout 的缓冲区。这是为了确保在交互式程序中，任何提示信息都会在读取输入之前显示出来。但是，这种绑定也会导致一些额外的性能开销。如果你的程序不需要这种交互式的行为，你可以取消这种绑定，以提高 I/O 效率。

注意，取消流的同步和绑定可能会改变程序的行为，特别是在多线程环境中，或者当程序混合使用 C++ 标准流和 C 标准流时。因此，只有当你确定取消同步和绑定不会导致问题，并且确实需要提高 I/O 效率时，才应该使用这两行代码。

s.replace(pos,n,str)把当前字符串从索引pos开始的n个字符替换为str
s.replace(pos,n,n1,c)把当前字符串从索引pos开始的n个字符替换为n1个字符
cs.replace(it1,it2,str)把当前字符串[it1,it2)区间替换为str it1 ,it2为迭代器哦

通过stl的transform算法配合tolower 和toupper 实现。有4个参数，前2个指定要转换的容器的起止范围，第3个参数是结果存放容器的起始位置，第4个参数是一元运算

```
string s;
transform(s.begin(),s.end(),s.begin(),::tolower);      //转换小写
transform(s.begin(),s.end(),s.begin(),::toupper);      //转换大写
```

代码	含义
`s.find (str, pos)`	在当前字符串的pos索引位置（默认为0）开始，查找子串str，返回找到的位置索引，-1表示查找不到子串
`s.find (c, pos)`	在当前字符串的pos索引位置（默认为0）开始，查找字符c，返回找到的位置索引，-1表示查找不到字符
`s.rfind (str, pos)`	在当前字符串的pos索引位置开始，反向查找子串s，返回找到的位置索引，-1表示查找不到子串
`s.rfind (c,pos)`	在当前字符串的pos索引位置开始，反向查找字符c，返回找到的位置索引，-1表示查找不到字符
`s.find_first_of (str, pos)`	在当前字符串的pos索引位置（默认为0）开始，查找子串s的字符，返回找到的位置索引，-1表示查找不到字符
`s.find_first_not_of (str,pos)`	在当前字符串的pos索引位置（默认为0）开始，查找第一个不位于子串s的字符，返回找到的位置索引，-1表示查找不到字符
`s.find_last_of(str, pos)`	在当前字符串的pos索引位置开始，查找最后一个位于子串s的字符，返回找到的位置索引，-1表示查找不到字符
`s.find_last_not_of ( str, pos)`	在当前字符串的pos索引位置开始，查找最后一个不位于子串s的字符，返回找到的位置索引，-1表示查找不到子串

ptrdiff_t 是一种有符号整数类型，它的大小足以存储两个指针之间的差值。这种类型在 **<cstddef>** 头文件中定义。

在 C++ 中，ptrdiff_t 主要用于表示两个指针之间的差值，这在进行指针运算时非常有用。例如，当你有两个指向数组元素的指针，并且你想知道这两个元素在数组中的距离时，你可以使用 ptrdiff_t。

```
#include <iostream>
#include <cstddef>
int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int* p1 = &arr[0];
    int* p2 = &arr[3];
    ptrdiff_t diff = p2 - p1;
    std::cout << diff << std::endl;  // 输出 3
    return 0;
}
```

uintptr_t 是一种整数类型，其宽度足以**存储一个指针的值**。这种类型在 <cstddef> 或 <cstdint> 头文件中定义。

在 C++ 中，uintptr_t 主要用于在需要将**指针转换为整数进行某些操作**（例如，某些哈希函数或位操作）时，保证转换的安全性和可移植性。

```
#include <iostream>
#include <cstddef>
int main() {
    int num = 10;
    int* ptr = &num;
    uintptr_t intPtr = reinterpret_cast<uintptr_t>(ptr);
     std::cout << "The integer representation of the pointer: " << intPtr <<
         std::endl;

    int* originalPtr = reinterpret_cast<int*>(intPtr);
    std::cout << "The original value: " << *originalPtr << std::endl;

    return 0;
}
```

```
#include <unistd.h>
#include <iostream>
int main() {
    char buffer[128];
    ssize_t bytesRead = read(STDIN_FILENO, buffer, sizeof(buffer) - 1);

if (bytesRead >= 0) {
        buffer[bytesRead] = '\0';  // Null-terminate the string
        std::cout << "Read " << bytesRead << " bytes: " << buffer << std::endl;
    } else {
        std::cerr << "Read error" << std::endl;
    }
    return 0;
}
```

```cpp
#include <iostream>

int main() {
    char buffer[128];
    std::cin.read(buffer, sizeof(buffer) - 1);
    std::streamsize bytesRead = std::cin.gcount();
    buffer[bytesRead] = '\0';  // Null-terminate the string
    std::cout << "Read " << bytesRead << " bytes: " << buffer << std::endl;
    return 0;
}
```

代码	含义
`bitset < n >a`	a有n位，每位都为0
`bitset < n >a(b)`	a是unsigned long型u的一个副本
`bitset < n >a(s)`	a是string对象s中含有的位串的副本
`bitset < n >a(s,pos,n)`	a是s中从位置pos开始的n个位的副本

```cpp
#include<bits/stdc++.h>
using namespace std;
int main() {
    bitset<4> bitset1;       //无参构造，长度为 4 ，默认每一位为 0

    bitset<9> bitset2(12);  //长度为9，二进制保存，前面用 0 补充

    string s = "100101";
    bitset<10> bitset3(s);    //长度为10，前面用 0 补充

    char s2[] = "10101";
    bitset<13> bitset4(s2);    //长度为13，前面用 0 补充

    cout << bitset1 << endl;    //0000
    cout << bitset2 << endl;    //000001100
    cout << bitset3 << endl;    //0000100101
    cout << bitset4 << endl;  //0000000010101
    return 0;
}
```

```
bitset<4> foo (string("1001"));
bitset<4> bar (string("0011"));
cout << (foo^=bar) << endl;// 1010 (foo对bar按位异或后赋值给foo)
cout << (foo&=bar) << endl;// 0001 (按位与后赋值给foo)
cout << (foo|=bar) << endl;// 1011 (按位或后赋值给foo)
cout << (foo<<=2) << endl;// 0100 (左移2位，低位补0，有自身赋值)
cout << (foo>>=1) << endl;// 0100 (右移1位，高位补0，有自身赋值)
cout << (~bar) << endl;// 1100 (按位取反)
cout << (bar<<1) << endl;// 0110 (左移，不赋值)
cout << (bar>>1) << endl;// 0001 (右移，不赋值)
cout << (foo==bar) << endl;// false (1001==0011为false)
cout << (foo!=bar) << endl;// true  (1001!=0011为true)
cout << (foo&bar) << endl;// 0001 (按位与，不赋值)
cout << (foo|bar) << endl;// 1011 (按位或，不赋值)
cout << (foo^bar) << endl;// 1010 (按位异或，不赋值)
```

代码	含义
`b.any()`	b中是否存在置为1的二进制位，有 返回true
`b.none()`	b中是否没有1，没有 返回true
`b.count()`	b中为1的个数
`b.size()`	b中二进制位的个数
`b.test(pos)`	测试b在pos位置是否为1，是 返回true
`b[pos]`	返回b在pos处的二进制位
`b.set()`	把b中所有位都置为1
`b.set(pos)`	把b中pos位置置为1
`b.reset()`	把b中所有位都置为0
`b.reset(pos)`	把b中pos位置置为0
`b.flip()`	把b中所有二进制位取反
`b.flip(pos)`	把b中pos位置取反
`b.to_ulong()`	用b中同样的二进制位返回一个unsigned long值

- ios_base::app：　附加到现有文件末尾，而不是覆盖它。
- ios_base::ate：　切换到文件末尾，但可在文件的任何地方写入数据。
- ios_base::trunc：　导致现有文件被覆盖，这是默认设置。
- ios_base::binary：创建二进制文件（默认为文本文件）。
- ios_base::in：　以只读方式打开文件。
- ios_base::out：　以只写方式打开文件。

```
while (myFile.good())
{
    getline (myFile, fileContents);
  cout << fileContents << endl;
}
```

读取用getline

```
class Widget { ... };
class SpecialWidget: public Widget { ... };
void update(SpecialWidget *psw);
SpecialWidget    sw;                    // sw 是一个非 const 对象。
const SpecialWidget& csw = sw;          // csw 是 sw 的一个引用 是const对象
update(&csw);                           // 错误!不能传递一个 const SpecialWidget* 变量
                                        // 给一个处理 SpecialWidget*类型变量的函数
update(const_cast<SpecialWidget*>(&csw)); // 正确，csw 的 const 被显示地转换掉
                                          csw 和 sw 两个变量值在 update函数中能被更新
update((SpecialWidget*)&csw);           // 同上，但用了一个更难识别的 C 风格的类型转换
Widget *pw = new SpecialWidget;
update(pw); // 错误！ pw 的类型是 Widget*，但是update 函数处理的是 SpecialWidget*类型
update(const_cast<SpecialWidget*>(pw));// 错误！ const_cast 仅能被用在影响
                                       // constness or volatileness 的地方上。,
                                       // 不能用在向继承子类进行类型转换。
```

```
update(dynamic_cast<SpecialWidget*>(pw));
                                    // 正确，传递给 update 函数一个指针
                                    // 是指向变量类型为 SpecialWidget 的 pw 的指针
                                    // 如果 pw 确实指向一个对象,否则传递过去的将使空指针。
void updateViaRef(SpecialWidget& rsw);
updateViaRef(dynamic_cast<SpecialWidget&>(*pw));
                                    //正确。 传递给 updateViaRef 函数SpecialWidget pw 指针，如
                                      果 pw确实指向了某个对象否则将抛出异常
```

```
int firstNumber, secondNumber;
double result = dynamic_cast<double>(firstNumber)/secondNumber;
                                                    // 错误！没有继承关系
const SpecialWidget sw;
update(dynamic_cast<SpecialWidget*>(&sw));     // 错误! dynamic_cast 不能转换掉 const。
```

Thanks To your Careful Guidance