基本输入输出流

Andrew Huang<bluedrum@163.com>

课程内容

- C库的printf/scanf的缺点
- 输出流
  - 插入操作符
  - 输出操纵符
- 输入流
  - 提取操作符
  - 输入操纵符

C 库中printf/scanf 的缺点

- 在C 库中用printf/scanf 也实现了标准输入输出的流机制．但有一些缺点
- 缺点1：非类型安全
  - 函数原型是使用编译系统对函数调用时进行必要的类型检查，免除了许多错误．
  - 但printf/scanf 无法利用这一强大工具，因此它们所期望的参数个数和类型的信息包含在第一个格式字符串里，编译器无法利用它进行检查

```
#include <stdio.h>
int j=10;
float =2.3;
void fn(){//输入错误类型,但无法检测出来
   printf(%d",f); scanf("%d",&f);
   scanf("%d",j); printf("%d","abcde");
   printf("%d,%d\n",1);
}
```

C 库中printf/scanf 的缺点

- 缺点2：无法扩展
  - 除了printf() 和scanf() 知道如何输入输出已知的基本数据类型值，在C++ 程序中大量类对象，其输入输出格式是未预先定义．
  - printf() 和scanf() 不能识别，也无法扩展识别用户定义的对象

```
class A{/* */}
A a;
//...
void fn(){
   printf("%?",a); //?表示不知道用什么格符   scanf("%?",&a);
}
```

输入/输出流的概念 3-1

- C++将输入和输出看作字节流
- 输入来自标准输入设备（键盘），或从其他输入设备重新定向而来
- 输出发送到标准输出设备或其他输出设备
- C++为每一个流关联一个缓冲区

输入/输出流的概念 3-2

- 流是字符集合或数据流的源或目的地

- 有两种流
  - 输出流,在类库里称为ostream
  - 输入流,在类库里称为istream
  - 有时了同时处理输入和输出,由istream和ostream派生双向流iostream.

输入/输出流的概念 3-3

- 预定义的流在iostream.h中定义

C++名字	对应设备	C对应名字	默认设备
cin	标准输入流	stdin	键盘
cout	标准输出流	stdout	屏幕
cerr	标准错误流（非缓冲）	stderr	屏幕
clog	标准错误流（缓冲)	stdprn	打印机

输出流

输出流

- ostream类包含为输出操作定义的函数
- 标准流上的输出是使用cout对象实现的
  - cout是一个全局对象,为类ostream的一个实例
  - ostream cout;
- 插入操作符
  - ostream类为实现输出重载了操作符 “<<”
  - 可以是任意复杂的表达式，系统可自动计算其值并传给插入符；
  - 指针类型的地址值，默认为十六进制显示，用long强制转换后才可以十进制显示；
  - 字符指针的地址值的输出格式为：**(void *)s**或**void *(s)**，此时仍为十六进制格式；

示例：

```
cout << variablename;
```

- variablename可以是任何基本数据类型
- 插入操作符右边的内容到左边的流对象中
- 屏幕是默认的输出流

```
#include <ostream.h>
int main()
{ int i=8; int * p=&i;
    cout <<“i=“<<i<<“,addr=“<<p;
     cout <<"i="<<(void *)i<<",addr="<<(int)p; }
```

一个更复杂的输出例子

```
#include <iostream.h>
#include <string.h>
void main()
{
      char *str="Hello";
      int a=100;
      int *pa=&a;
      cout<<"*pa="<<*pa<<endl;
      cout<<"&pa="<<&pa<<" or "<<long(&pa)<<endl;
      cout<<"The string is "<<str<<endl;
      cout<<"The address is "<<(void *)str
                <<" or "<<long((void *)str)<<endl;
}
```

输出流

- ostream类还提供其他输出函数
  - put() – 输出字符
    - ostream& ostream::put(char c);
  - write() – 输出字符串
    - ostream& ostream::write(char *buf,int n);
- 这些成员函数既可用于文本流，也可用于二进制流，尤其适用于二进制流；

cout.write 的例子

```
#include <iostream.h>
void main()
{
	cout<<'a'<<', '<<'b'<<'\n';
	cout.put('a').put(', ').put('b').put('\n');
	char c1='A',c2='B';
	cout.put(c1).put(c2).put('\n');
}
```

```
#include <iostream.h>
#include <string.h>
void PrintString(char *s)
{
	cout.write(s,strlen(s)).put('\n');
	cout.write(s,6)<<"\n";
}
void main()
{
	char str[]="I love you?";
	cout<<"The string is: "<<str<<endl;
	PrintString(str);
	PrintString("this is a string");
}
```

转义字符

- printf 提供输出转义字符的功能
  - 如\n,\r,\\,\¡±,\¡¯
- cout 也支持上述转义符号，并支持endl() 来作换行符

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

int main()
{

cout<<"\n\"Least          said\n\t\tsoonest
mended.\"\n\a"<<endl;
return 0;
}
```

清空输出缓冲区

- C 标准库使用 fflush(stdout) 清空;
- cout 也提供flush 来清空缓冲区
  - cout << "C++ 的 I/O 需要刷新。"<< flush;

格式化输出

- printf提供各种格式化输出控制符,如宽度,小数位,对齐.
- cout 也提供相应机制控制这些格式.

- width()函数或setw(int w)用于设置输出的字段宽度，默认是0
- fill()函数用于设置填充字符，默认是空格。如果指定的宽度大于实际的输出，C++用空格填充多余的位置
- precision()函数或setprecision(int d)将精度位数设置为d，默认是6

控制输出宽度

- 控制输出宽度

  1 、在流中放入setw 操纵符，

  2 、width 成员函数是设置cout 的每一项缺省输出宽度

```
#include <iostream.h>
void main()
{       double values[ ] = {1.23,35.36,653.7,4358.24};

cout <<"default width = "<<cout.width ()<<    endl;
       for(int i=0;i<4;i++)
       {     cout.width(10);
             cout << values[i] <<endl;
       }
}
```

输出结果:

```
      1.23
     35.36
     653.7
  4358.24
```

控制输出宽度(2)

- 用setw(int n) 设置每一个输出项的宽度

```
void main()
{   double values[ ] = {1.23,35.36,653.7,4358.24};
      char *names[ ] = {"Zoot", "Jimmy", "Al", "Stan"};
      for (int i=0;i<4;i++)
                  cout << setw(6) << names[i] << setw(10);
                  cout << values[i] << endl;
}
```

输出增充字符

- fill()函数用于设置填充字符，默认是空格。如果指定的宽度大于实际的输出，C++用空格填充多余的位置

```
#include <iostream.h>
void main()
{       double values[ ] = {1.23,35.36,653.7,4358.24};
   for(int i=0;i<4;i++)
      {     cout.width(10);
            cout.fill('*');
            cout << values[i] <<endl;
      }
}
```

输出结果:

```
******1.23
*****35.36
*****653.7
***4358.24
```

控制浮点数值显示

- 使用setprecision(n)可控制输出流显示浮点数的数字个数。C++默认的流输出数值有效位是6。

- 如果setprecision(n)与setiosflags(ios::fixed)合用，可以控制小数点右边的数字个数。setiosflags(ios::fixed)是用定点方式表示实数
- 如果与setiosnags(ios::scientific)合用， 可以控制指数表示法的小数位数。setiosflags(ios::scientific)是用指数方式表示实数。

控制浮点数值显示(2)

```
//下面的代码分别用浮点、定点和指数方式表示一个实数：
#include <iostream.h>
#include <iomanip.h> //要用到格式控制符

void main()
{
      double amount = 22.0/7;
      cout <<amount <<endl;
      cout <<setprecision(0) <<amount <<endl
        <<setprecision(1) <<amount <<endl
        <<setprecision(2) <<amount <<endl
        <<setprecision(3) <<amount <<endl
        <<setprecision(4) <<amount <<endl;

      cout <<setiosflags(ios::fixed);
      cout <<setprecision(8) <<amount <<endl;

      cout <<setiosflags(ios::scientific) <<amount <<endl;
      cout <<setprecision(6); //重新设置成原默认设置
}
```

输出结果:

```
.14286
3
3
3.1
3.14
3.143
3.14285714
3.14285714e+00
```

设置左右对齐输出

- 默认时，I/O 流左对齐显示的内容。使用头文件iomanip.h 中的setiosflags(ios::left)和(ios::right) 标志，可以控制输出对齐。

```
#include <iostream.h>
#include <iomanip.h>
void main()
{   double values[ ] = {1.23,35.36,653.7,4358.24};
      char *names[ ] = {"Zoot", "Jimmy", "Al", "Stan"};
          for (int i=0;i<4;i++)
                  cout
<<setiosflags(ios::left)<<setw(6)<<names[i]

<<resetiosflags(ios::left)<<setw(10)<<values[i]<<endl;
}
```

输出8 进制和16 进制数

- 3个常用的控制符是hex，oct和dec，它们分别对应16进制、8进制和10进制数的显示。这3个控制符在iostream.h头文件中定义
- 对应 printf的%x,%o,%d

```
#include <iostream.h>

void main()
{
      int number=1001;
      cout <<"Decimal:" <<dec <<number <<endl
      <<"Hexadecimal:" <<hex <<number <<endl
      <<"Octal:" <<oct <<number <<endl;
}
```

输出流

- ios 类包含一个用来控制多种格式特征的函数 setiosflags ()

- setiosflags() 函数使用下列枚举型常量作为参数，必须在它们前面使用类名和作用域解析操作符 (::)
  - showbase - 在输出的八进制数字前加 “0”，在输出的十六进制数字前加 “0x”
  - showpoint - 总是用一个小数点和尾随零显示浮点数
  - uppercase - 对十六进制的输出使用大写字母 (A-F)
  - showpos - 使用前导“+” 显示正数

强制显示小数和正数符号

```
#include <iostream.h>
#include <iomanip.h>

void main()
{
//显示小数点
cout <<10.0/5 <<endl;
cout <<setiosflags(ios::showpoint)
        <<10.0/5 <<endl;

//显示符号
  cout <<10 <<" " <<-20 <<endl;
  cout <<setiosflags(ios::showpos)
        <<10 <<" " <<-20 <<endl;
}
```

输入流

输入流 5-1

- istream 类包含为输入操作定义的函数
- 来自标准流的输入是使用cin对象实现的
- cin与标准输入设备（键盘）相关联
- 提取操作符
  - istream 类为实现输入重载了操作符“>>”

输入流 5-2

示例：

```
int variablename;
cin >> variablename;
```

- 等待用户输入
- 从键盘的数据将存储在变量variablename
- 提取操作时，空白符（空格、tab键、换行符）只用于字符的分隔符，而本身不作为从输入流中提取的字符；

```
#include <iostream>
using namespace std;
main ()
{
int a,b;
cin>>a>>b;
cout<<a+b<<endl;
}
```

```
#include <iostream>
using namespace std;
main ()
{//仍然有内存越界问题
char a[6];
cin>>a;
cout<<a<<endl;
}
```

get 方法

- 使用get()获取一个字符
  - istream& istream::get(char& c);
  - int istream::get();
- get(char &ch) – 将输入的字符存储在ch中。它获取输入的下一个字符，即使它是一个空白字符,或者回车符

```
#include <iostream>
using namespace std;
main ()
{
char ch;
ch=cin.get(); //或者cin.get(ch);
cout<<ch<<endl;
}
```

get 方法(2)

- cin.get(char *buffer, int size,char delim=¡®\n¡¯)
  - 或者读size ¨C1 个字符入buffer,或者遇到¡®\n¡¯;在buffer最后加¡®\0¡¯,分隔符留在输入流.

```
#include <iostream>
using namespace std;
main ()
{
        char a[6];
        cin.get(a,sizeof(a));
        cout<<a<<endl;
}
```

getline() 方法

- 使用getline() 获取多个字符
  - istream& istream::getline(char *buf,int Limit,Deline='\n');
  - getline()最多可读取Limit-1个字符
- getline() 函数结束操作的条件：
  - 从输入流中读取Limit-1 个字符后；
  - 从输入流中读取换行符或其他终止符后；
  - 从输出流中读取到文件或输入流结束符后；
- get()和 getline()的区别
  - **get()** 不丢弃换行符（**\n** ）**,** 仍然把它保存在输入缓冲区中，**getline()** 读取换行符并将起抛弃

getline 实例

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   const int ARRAY_SIZE=10;
   char array[ARRAY_SIZE];

   cin.getline(array, ARRAY_SIZE); //输入一行
   cout<<array<<endl;
   return 0;
}
```

read

- 使用cin.read作无格式读取一串字符
  - 包括空格, 回车符,Tab 均会被读入
- 与其相对应了cout有write 方法输出指定的字符串

read 实例

```
#include <iostream>
using namespace std;
main ()
{
const int SIZE = 80;
char buffer[SIZE];
cin.read(buffer, 20);
cout.write(buffer, cin.gcount());
        //gcount返回上次读入的字节数
cout << endl;}
```

```
输入：  Using the read, write and gcount member
输出:    Using the read, write
```

格式化输入控制

- 操纵符提供了格式化数据的简单方法
  skipws - 在提取(“>>”)时跳过空白字符

使用ignore() ，peek(),putback();

- cin.ignore(8,'\n'); 其含义是：忽略换行符（\n ）qi 前面8 个字符。
- int istream::peek();// 只是查看，并不提取下一个字符
- istream& istream::putback(char *,int);// 则向输入流中插入一个字符
- int istream::gcount()// 返回上次输入的字符数

样例

```
#include<iostream>;
int main()
{
    char chcin;
    cout<<"Enter a string:";

    while(cin.get(chcin)) //取得流中的字符
     {
      //if  的功能是把  '!',则换成  '*'
      if(chcin == '!')
            cin.putback('*');
      else          cout<<chcin;

      // second while功能是删除  '#'
        while(cin.peek() == '#')
            cin.ignore(1,'#');
     }
    return 0;
}
```

忽略空格

- 输入操纵符
  操纵符提供了格式化数据的简单方法

  skipws - 在提取(“>>”)时跳过空白字符
  cin. setf(ios::skipws);

忽略空格，样例

```
#include <iostream.h>
int main()
{
    int count = 0;
    char ch;
    cin. setf(ios::skipws);//输入空格将不计数
// cin. unsetf(ios::skipws);
    cin >> ch;
    while(ch != '.') {
        count++;
        cin >> ch;
    }
    cout << endl;
    cout << "共有：" << count << "个字符" << endl;
    return 0;
}
```

iostream.h 与iostream 的区别

iostream.h 与iostream 的区别

- iostream.h 是为兼容旧有系统而保留的版本，是从C 继承下来. 而iostream 是新的C++ 标准支持的.
- 在VC++ 6.0 两者都支持，但在VC++ 7.0 ，只能使用最新的iostream
- 明确是C++ 程序的情况下，最好使用C++ 标准头文件

```
//旧的标准
#include <iostream.h>
int main(){
   cout<<“hello!”<<endl;
}
```

```
//C++标准库
#include <iostream>
using namespace std;
int main(){
   cout<<“hello!”<<endl;
  std::cout<<“other output!”;
}
```

标准C++ 库

- 在C++ 标准化后，为了避免名字污染，引入名字空间的概念，将 C++ 标准库中的所有名字都封装到 std 中，为了使以前的程序能继续使用，所以新的头文件都去掉了后缀名.h
  - <iostream.h> 变成<iostream>
  - <complex.h> 变成 <complex>
- 对原有的C 库头文件，不仅取消了后缀名，而且还在名字要加一个c
  - <string.h> 变成<cstring>
  - <stdio.h> 变成<cstdio>
- 前常要在代码前加入使用std 空间代码
  - using namespace std;

关于C 函数与C++ 标准流混用问题

- 在很多情况,不建议混用c标准库函数和C++标准流.下例用getchar和与旧版本cout混用产生问题
- 最好是cin与cout配合使用
- 标准C++库(<iostream>)工作正常,但也不推荐混用

```
#include <iostream.h>
#include <stdio.h>

void main()
{
  cout <<1<<¡°hello¡±; //加入<<endl两者都执行正常
  getchar(); //执行这里不显示
  //cin.get(); //执行这个显示
}
```

课堂练习

- 1.请完全用C++标准库的cout写出一个打印一个内存buffer的函数,请用16进制显示(中等)
  - 每个行首打印出buffer地址,如打印0x001268F0的buffer,应该有如下效果
  - 0x001268F0 FF 89 EA 85 38 23 ...
  - 函数有三个参数,开始地址,打印长度和每行显示个数,其中最后一个参数缺省值为16
  - 请写一个标准C版本,与两者对比一下.
- 2.已写一个程序,用cin.read接收一个字符串,回车结束,然后在屏幕重新显示
  - 注意不能使用cin.get或cin.getline