C++语言程序设计

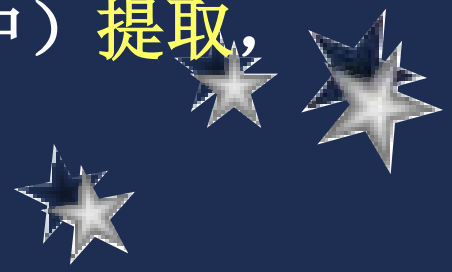
第十一章 流类库与输入/输出

本章主要内容

- I/O流的概念
- 输出流
- 输入流
- 输入/输出流

I/O流的概念

- 当程序与外界环境进行信息交换时，存在着两个对象，一个是程序中的对象，另一个是文件对象。
- 流是一种抽象，它负责在数据的生产者和数据的消费者之间建立联系，并管理数据的流动。
- 程序建立一个流对象，并指定这个流对象与某个文件对象建立连接，程序操作流对象，流对象通过文件系统对所连接的文件对象产生作用。
- 读操作在流数据抽象中被称为（从流中）提取，写操作被称为（向流中）插入。



输出流

输出流

- 最重要的三个输出流是
  - ostream
  - ofstream类支持磁盘文件输出
  - ostringstream类向字符串写入数据

输出流对象

输出流

- 预先定义的ostream类对象:
  - cout 标准输出
  - cerr 标准错误输出，没有缓冲，发送给它的内容立即被输出。
  - clog 类似于cerr，但是有缓冲，缓冲区满时被输出。

输出流对象

输出流

- ofstream类支持磁盘文件输出
- 如果在构造函数中指定一个文件名，当构造这个文件时该文件是自动打开的
  - ofstream myFile("filename");
- 可以在调用默认构造函数之后使用open成员函数打开文件

```
ofstream myFile; //声明一个静态文件输出流对象
myFile.open("filename");
    //打开文件，使流对象与文件建立联系
```

- 在构造对象或用open打开文件时可以指定模式
  - ofstream myFile("filename", ios_base::out | ios_base::binary);

插入运算符（<<）

输出流

- 插入(<<)运算符是所有标准C++数据类型预先设计的。
- 用于传送字节到一个输出流对象。

控制输出格式

输出流

- 控制输出宽度
  - 为了调整输出，可以通过在流中放入setw操纵符或调用width成员函数为每个项指定输出宽度。
  - setw和width都不截断数值
  - setw和width仅影响紧随其后的域，在域输出结束后宽度恢复默认值（必要的宽度）
- 例11-1 使用width控制输出宽度

```
#include <iostream>
using namesoace std;
int main() {
   double values[] = { 1.23, 35.36, 653.7, 4358.24 };
   for(int i = 0; i < 4; i++) {
      cout.width(10);
      cout << values[i] << endl;
   }
   return 0;
}
```

输出结果：

```
      1.23
     35.36
     653.7
   4358.24
```

例：使用*填充

输出流

- 缺省为空格
- 使用setfill操纵符或fill成员函数设置填充字符
- 持久影响

```
#include <iostream>
using namespace std;
int main() {
  double values[]={ 1.23, 35.36, 653.7, 4358.24 };
  cout.fill('*');
  for(int i = 0; i < 4; i++) {
    cout.width(10);
    cout << values[i] << '\n';
  }
  return 0;
}
```

输出结果：

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

例11-2使用setw指定宽度

输出流

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
int main() {
   double values[] = { 1.23, 35.36, 653.7, 4358.24 };
   string names[] = { "Zoot", "Jimmy", "Al",
   "Stan" };
   for (int i = 0; i < 4; i++)
      cout << setw(6) << names[i]
      << setw(10) << values[i] << endl;
   return 0;
}
```

输出结果：

```
  Zoot      1.23
 Jimmy     35.36
    Al     653.7
  Stan  4358.24
```

对齐方式

输出流

- 输出流默认为右对齐文本
- **setiosflags操作符来设置格式标志，持久影响**

  smanip setiosflags (ios_base::fmtflags mask);
- **resetiosflags操作符来清除格式标志**

  smanip resetiosflags (ios_base::fmtflags mask);
- 掩码**mask**可用位或**|**进行组合：
  - 对齐：ios_base::left, ios_base::right
  - 进制：ios_base::dec, ios_base::oct, ios_base::hex
  - 浮点：ios_base::fixed, ios_base::scientific
  - 其他：ios_base::skipws 等

例11-3设置对齐方式

输出流

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
int main() {
   double values[] = { 1.23, 35.36, 653.7, 4358.24 };
   string names[] = { "Zoot", "Jimmy", "Al", "Stan" };
   for (int i=0;i<4;i++)
       cout << setiosflags(ios_base::left)
            << setw(6) << names[i]
            << resetiosflags(ios_base::left)
            << setw(10) << values[i] << endl;
   return 0;
}
```

输出结果:

```
Zoot         1.23
Jimmy       35.36
Al          653.7
Stan      4358.24
```

精度

输出流

- 浮点数输出精度默认值是6
- setprecision操纵符或precision成员函数来设置精度
- 3466.9768显示为3466.98
  - 若设置了ios_base::fixed，定点格式（没有指数部分）：3466.976800
  - 若设置了ios_base::scientific，科学格式：3.466977e+003
  - 若都未设置，精度值确定总的有效位数，否则确定小数点后小数位数

例11-4控制输出精度

输出流

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
int main() {
   double values[] = { 1.23, 35.36, 653.7, 4358.24 };
   string names[] = { "Zoot", "Jimmy", "Al", "Stan" };
   for (int i=0;i<4;i++)
     cout << setiosflags(ios_base::left)
     << setw(6) << names[i]
     << resetiosflags(ios_base::left)
     << setw(10) << setprecision(1) << values[i]
     << endl;
   return 0;
}
```

输出结果：

```
Zoot          1
Jimmy      4e+001
Al         7e+002
Stan       4e+003
```

进制

输出流

- 默认进制是dec（十进制）
- dec、oct和hex操纵符设置输入和输出的默认进制

文件输出流成员函数

输出流

- 输出流成员函数有三种类型：
  - 与操纵符等价的成员函数。
  - 执行非格式化写操作的成员函数。
  - 其它修改流状态且不同于操纵符或插入运算符的成员函数。

文件输出流成员函数

输出流

- **open函数**
  把流与一个特定的磁盘文件关联起来。
  需要指定打开模式。默认模式：ios_base::out
- **put函数**
  把一个字符写到输出流中。
- **write函数**
  把内存中的一块内容写到一个文件输出流中
- **seekp和tellp函数**
  操作文件输出流的内部指针
- **close函数**
  关闭与一个文件输出流关联的磁盘文件
- **错误处理函数**
  在写到一个流时进行错误处理

例11-5向文件输出

输出流

```
#include <fstream>
using namespace std;
struct Date {
   int mondy, day, year;
};
int main() {
   Date dt = { 6, 10, 92 };
   ofstream file("date.dat", ios_base::binary);
   file.write(reinterpret_cast<char *>(&dt),
   sizeof(dt));
   file.close();
   return 0;
}
```

二进制输出文件

输出流

- 以通常方式构造一个流，然后使用setmode成员函数，在文件打开后改变模式。
- 使用ofstream构造函数中的模式参量指定二进制输出模式
  ios_base::binary，默认是文本模式

字符串输出流ostringstream

输出流

- 用于构造字符串
- 功能
  - 支持ofstream类的除open、close外的所有操作
  - str函数可以返回当前已构造的字符串
- 典型应用
  - 将数值转换为字符串

例11-6 字符串输出流应用

输出流

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
template <class T>
inline string toString(const T &v) {
   ostringstream os;  //创建字符串输出流
   os << v;           //将变量v的值写入字符串流
   return os.str();   //返回输出流生成的字符串
}
int main() {
   string str1 = toString(5);
   cout << str1 << endl;
   string str2 = toString(1.2);
   cout << str2 << endl;
   return 0;
}
```

输出结果：

5

1.2

输入流

输入流

- 重要的输入流类：
  - istream类最适合用于顺序文本模式输入。**cin是**其实例。
  - ifstream类支持磁盘文件输入。
  - istringstream类支持从字符串中读取数据

输入流对象

输入流

- 如果在构造函数中指定一个文件名，在构造该对象时该文件便自动打开。

```
ifstream myFile("filename");
```

- 在调用默认构造函数之后使用open函数来打开文件。

```
ifstream myFile;//建立一个文件流对象
myFile.open("filename",);
  //打开文件"filename"
```

- 打开文件时可以指定模式

```
ifstream myFile("filename", ios_base::in |
  ios_base::binary);
```

提取运算符（>>）

输入流

- 提取运算符(>>)对于所有标准C++数据类型都是预先设计好的。
- 是从一个输入流对象获取字节最容易的方法。
- ios类中的很多操纵符都可以应用于输入流。但是只有少数几个对输入流对象具有实际影响，其中最重要的是进制操纵符dec、oct和hex。

输入流成员函数

输入流

- open函数把该流与一个特定磁盘文件相关联。
- get函数的功能与提取运算符（>>）很相像，主要的不同点是get函数在读入数据时包括空白字符。
- getline的功能是从输入流中读取多个字符，并且允许指定输入终止字符，读取完成后，从读取的内容中删除终止字符。
- read成员函数从一个文件读字节到一个指定的内存区域，由长度参数确定要读的字节数。
  如果给出长度参数，当遇到文件结束或者在文本模式文件中遇到文件结束标记字符时结束读取。

输入流成员函数

输入流

- seekg函数用来设置文件输入流中读取数据位置的指针。
- tellg函数返回当前文件读指针的位置。
- close函数关闭与一个文件输入流关联的磁盘文件。

例11-9 从文件读二进制记录

输入流

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

struct SalaryInfo {
   unsigned id;
   double salary;
};

int main() {
   SalaryInfo employee1 = { 600001, 8000 };
   ofstream os("payroll", ios_base::out |
   ios_base::binary);
   os.write(reinterpret_cast<char *>(&employee1),
   sizeof(employee1));
   os.close();
```

输入流

```
    ifstream is("payroll", ios_base::in | ios_base::binary);
    if (is) {
        SalaryInfo employee2;
        is.read(reinterpret_cast<char *>(&employee2),
                sizeof(employee2));
        cout << employee2.id << " " << employee2.salary << endl;
    } else {
        cout << "ERROR: Cannot open file 'payroll'." << endl;
    }
    is.close();

    return 0;
}
```

字符串输入流istringstream

输入流

- 用于从字符串读取数据
- 在构造函数中设置要读取的字符串
- 功能
  - 支持ifstream类的除open、close外的所有操作
- 典型应用
  - 将字符串转换为数值

例11-12 字符串输入流应用

输入流

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
template <class T>
inline T fromString(const string &str) {
    istringstream is(str);      //创建字符串输入流
    T v;
    is >> v;                    //从字符串输入流中读取变量v
    return v;                   //返回变量v
}
int main() {
    int v1 = fromString<int>("5");
    cout << v1 << endl;
    double v2 = fromString<double>("1.2");
    cout << v2 << endl;
    return 0;
}
```

输出结果：

```
5
1.2
```

输入输出流

输入输出流

- iostream类对象可以是数据的源或目的
- fstream类支持磁盘文件输入和输出。
- stringstream类支持面向字符串的输入和输出

小结与复习建议

- 主要内容
  - I/O流的概念、输出流、输入流、输入/输出流。
- 达到的目标
  - 理解I/O流的概念，学会使用I/O流类库实现文件输入/输出及格式控制。