2020/05/05_C++_第1课_输入输出流

笔记本: C++
创建时间: 2020/5/5 星期二 15:14
作者: ileemi
标签: C++ auto, C++输入输出流

简单介绍C++

为什么要学C++
开发效率更快，语法层面支持面向对象编程思想

全面兼容C语言的语法
1、改进了C语言的语法(函数重载、const、内联、bool、类型推导)
2、面向对象编程(封装、类型、多态)
3、版本(98、03、11、14、17、19、20b)

输入输出流

1、简单的C++对C语法的改进
2、C++里面的输入输出流

简单的C++对C语法的改进

1、C++类型的检查更加严格，了解即可

```
int nTest1 = 999;
int nTest2 = 3.17f;                                    //从“float”转换
到“int”，可能丢失数据
int nTest3 = 33.33;                                    //从“double”转换
到“int”，可能丢失数据
int nTest4 = { 6 };
//需要类型严格匹配
//int nTest5 = { 3.12 };                               //从“double”转换
到“int”需要收缩转换
```

```
//char ch = 0x001277fc ;          //“初始化”：从“int”到“char”截断，“初始化”：
截断常量值
char ch = { 0x001277fc };
int n6{ 9 };
```

2、类型占位符 ->类型推导 (11 14语法)

根据 = 右边的数值类型，推导出左侧变量的类型

```
auto nTest = 10;
auto dblTest = 3.14;
auto szTest = "Hello World";
//a1 = "Hahaha";  //报错，无法从“const char [7]”转换为“int，因为x是int类型，不能赋值char* 类型
//auto a4;  //报错，类型包含“auto”的符号必须具有初始值设定项，因为没有数值可以来给t推导类型

/*
类型：list<pair<int, char *>> :: interator
变量名：itr
*/
list<pair<int, char *>> :: interator  itr = NULL;
auto itr = NULL;

//类似的类型推导 decltype，根据程序中已经存在的变量类型去推导对应的类型
decltype(nTest) nTest2 = 100;                    //根据nTest的类型，推导出int类型，将nTest2也视为int类型
decltype(dblTest) dblTest2 = 7.77;
decltype(szTest) szTest2 = 999;          //报错，无法从“int”转换为“const char *”，类型不匹配

//应用场景，在类型比较复杂的时候使用
```

3、C++空指针 nullptr (11语法)

```
C语言为指针赋初值的使用方式：
//空指针
int* pTest = NULL;
int nTest = NULL;                        //此处将指针类型的NULL赋值给int类型的变量 nTest，可读性不好

C++中的空指针
//C++引入nullptr，专门用来区分0、NULL
int* pTest2 = nullptr;              //nullptr的类型为nullptr_t，能够隐式的转换为任何指针
```

```
//int nTest2 = nullptr;              //无法从“nullptr”转换为“int”
```

C语言中的NULL是一个宏，进行预处理时会对齐进行文本替换

```
//空指针
int* p = NULL;
```

vcruntime.h

```
229     #ifdef __cplusplus
230         #define NULL 0
231     #else
232         #define NULL ((void *)0)
233     #endif
234 #endif
235
236 #if defined _M_X64 || defined _M_ARM || defined _M_ARM64
```

行: 230 字符: 17 列: 22 空格 CRLF

C++：nullptr是一个空指针类型，使用时会做语法类型的检查，编译阶段做的事

4、范围迭代(11 14语法)

```
int nAry[] = { 1, 2, 3, 5, 6, 7, 8, 9, 10,11 };
for (int i = 0; i < sizeof(nAry) / sizeof(nAry[0]); i++)
{
    printf("%d ", nAry[i]);
}
printf("\r\n");
//auto nVal: nAry
for (int nVal : nAry)
{
    printf("%d ", nVal);
}
printf("\r\n");
for (auto nVal : {11, 10, 9, 8, 7, 6, 5, 3, 2, 1})
{

    printf("%d ", nVal);
}
```

C++的输入输出流 cout

C++头文件不需要加.h

C++的输出流

std :: cout << "Hello World" << std :: endl;

std ->命名空间

没有std，需要在main函数之前添加 using namespace std;

添加后就可以使用以下面的语句进行输出：

cout << "Hello World" << endl;

endl ->换行符

```
std::cout << "Hello World!" <<std:: endl;
std::cout << 100 << std::endl;
std::cout << 99.9f << std::endl;
std::cout << 3.14 << std::endl;
在main函数之前添加 using namespace std; 后就可以在每次输出数据前添加 std

cout << "Hello World" << endl;
cout << 100 << endl;
cout << 99.9f << endl;
cout << 3.14 << endl;


//使用cout同属输出多项数据
cout << "Hello World!" << 100 << 99.9f <<   3.14
                << endl;
//输出结果：Hello World!10099.93.14

//在数据间进行换行
cout << "Hello World!" << endl
    << 100 << endl
    << 99.9f << endl
    <<   3.14
    << endl;

//还可以在输出数据时，数据间空空格隔开或使用 '\t'
cout << "Hello World!" << " "
    << 100 << " "
    << 99.9f << " "
    <<   3.14 << " "
    << endl;
```

使用cout将数据进行数据转换

```
//C++的 cout默认输出 十进制数据
cout << 0x001277fc << endl;          //输出数据：1210364

//使用cout调用setf函数，将数值转换成对应的进制数值
//setf ->设置特定格式标志
cout.setf(ios::hex, ios::basefield);
cout << 0x1277fc << endl;          //输出数据：1277fc


//        //输出数值的8进制
```

setf 函数的使用方法：

```
//以八进制形式输出数值的8进制
cout.setf(ios::oct, ios::basefield);
cout << 0x12ff7c << endl;              //输出进制数据4577574
return 0;

//将数据以不同进制方法显示
//使用cout调用setf函数，将需要输出的数据按照指定的格式输出
cout.setf(ios::hex, ios::basefield);
cout << 999 << endl;          //输出十六进制数据：1277fc

//输出数值的8进制
cout.setf(ios::oct, ios::basefield);
cout << 999 << endl;              //输出进制数据4577574

//输出数值的8进制
cout.setf(ios::dec, ios::basefield);
cout << 999 << endl;              //输出进制数据999

//使用cout 调用setf函数当设置对应的进制数后，会影响到后面所有cout输出

//输出数值的8进制
cout.setf(ios::oct, ios::basefield);
cout << 999 << endl;        //4577574
cout << 16699 << endl;  //40473
cout << 999 << endl;    //1747

//同时还有一个取消之前设置的进制操作
//使用cout调用unsetf即可，操作如下
cout.setf(ios::oct, ios::basefield);
cout << 999 << endl;              //输出进制数据4577574

cout.unsetf(ios::oct);

cout << 16699 << endl;    //16699
cout << 999 << endl;              //输出进制数据999
```

cout默认输出 十进制数据，使用setf可以设置其他进制，使用unsetf可以取消其他进制，并恢复默认的十进制

```
  //不调用函数就可以直接将数值的输出对应的进制数值，操作如下：

cout << hex << 999 << " "
        << dec << 999 << " "
        << oct << 999 << " "
```

```
        << endl;                //3e7 999 1747

cout << hex << 999 << 0x12ff7c << endl;          //3e712ff7c 同样有后遗症
```

浮点数相关的操作:

```
        //科学计数法
cout << 3.18 << endl;//3.18
cout << scientific << 6.183232 << endl;                  //6.183232e+00
6.183232*10
cout << scientific << 22323.3232 << endl;          //2.232332e+04
2.232332 * 10    =
2.232332e
cout << scientific << 0.0000123 << endl;            //1.230000e-05 =
1.230000 * 10^-5


//浮点精度
cout << fixed <<1235.123 << endl;
//1235.123000

//显示浮点数后多少位的数据
cout.precision(2); //显示小数点后2位
cout << fixed << 3.1425926 << endl;                              //3.14
```

```
//对齐和宽度

cout.width(20);
cout.fill('*'); //空白处使用 * 进行填充
cout << right << "Hello World" << endl; //*********Hello World


//showbase ->生成为整数输出指示数字基底的前缀
cout << showbase << hex << 999 << " "
    << showbase << dec << 999 << " "
    << showbase << oct << 999 << " "
    << endl;

//程序运行结果：0x3e7 999 01747


/*
uppercase ->启用浮点和十六进制整数输出中大写字符的使用，对，十六进制以
```

```
为的进制，字符串没有效果
*/
cout << uppercase << "Hello World" << endl;
cout << hex << 999 << endl;
cout << uppercase << scientific << 98.9996 << endl;
/*
程序运行结果：
*********Hello World
0x3e7 999 01747
Hello World
0X3E7
9.899960E+01
*/
```

C++的输入流 cin

```
    //cin的基本需要
    int nNum = 0;
      float fltTest = 0.0f;
      double dblTest = 0.0;
      char szBuf[256];
      cin >> nNum >> fltTest >> dblTest >> szBuf;
      cout << "nNum: " << nNum << " "
           << "fltTest: " << fltTest << " "
           << "dblTest: " << dblTest << " "
           << "szBuf: " << szBuf << " "
           << endl;

      /*
      20
      3.14
      99.999
      HelloWorld
      nNum: 20 fltTest: 3.14 dblTest: 99.999 szBuf: HelloWorld
      */

#endif
```

```
  char szBuf[256] = { 0 };
   int nCount = 0;
```

```
    cin >> szBuf >> nCount;                          //自动截获输入中的空
格，导致达不到预期的结果
    cout << szBuf << " " << nCount << endl;
   // cin的基本使用
```

C++清理缓冲区可以使用下面两句：

```
获取缓冲区内还有多少数据
int nCount = cin.rdbuf()->in_avail;                  //获取缓冲区内还有多
少数据
cin.ignore(nCount);                                                    //
将缓冲去内的无用数据进行相应的处理

//示例：
    char szTest[32];
    int nNumber;
    cin >> szTest; //Hello World
    //清掉cin缓冲区中的数据
    int nCout = cin.rdbuf()->in_avail();
    cin.ignore(nCout);

    cin >> nNumber; //输入999
    cout << szTest << " " << nNumber << endl; //空格，回车，tab键都会截
断
    //输出：Hello 999
...

/*
从控制台每次读一行数据，如果一行中有空格和tab键，同样会被读入
getLine
原型：
basic_istream&  getline(  char_type*  s,  std::streamsize  count  );
s-指向要存储字符到的字符串的指针(例：要输入的字符数组首地址)
count-s  所指向的字符串的大小(例：字符数组的大小)
*/
cin.getline(szBuf, sizeof(szBuf));

用法：
        char szAry[32] = { '\0' };
        cout << "请输入一串字符(32字节以内)：";
        //从流释出字符，直至行尾或指定的分隔符 delim
        cin.getline(szAry, sizeof(szAry));
        cout << "字符数组cAry内保存的数据为：";
        cout << szAry << endl;
        return 0;
```

```
...
```