

信息学竞赛中可能用到的 C++ 语法

虽然我们总是习惯写“C/C++”，其实大家都知道 C 与 C++ 是两种不同的语言：C++ 是 C 的“超集”——它“几乎”完全兼容 C 的语法，同时也提供非常多特有的语法与概念。

本文并不想全面介绍 C++ 的语法与概念——那足够写一本一寸多厚的专业书籍——只是将信息学竞赛中可能要用到的 C++ 语法做个简要介绍，它们的加入将使您原有的 C 程序更简洁、清晰，书写更方便。

首先要说明的一点是：C++ 程序源文件的扩展名是“cpp”，而不是 C 程序的“c”，这也是多数编译器识别一个源程序是 C++ 程序还是 C 程序的标准。由于 C++ 是 C 的“超集”，因此您可以在“*.cpp”文件中同样书写 C 程序而且“几乎”不会遇到什么问题。特别强调：本文中介绍的语法，都是只能在“*.cpp”文件中使用的 C++ 语法，C 语言的编译器不会编译通过。（本文中所有源程序都在 Dev-C++ 中编译通过。）

一、输入输出流。

C++ 中对输入输出部分做了很大的修改，C 语言中使用 scanf、fscanf、printf、fprintf 进行的输入输出工作都由对“流(stream)”的操作代替了。您可以向一个输出流中用“<<”符号“插入”一些内容——相当于输出，也可以用“>>”符号从一个输入流中“提取”一些内容——相当于输入。

[例 1]（从键盘读入及向屏幕输出）从键盘读入两个整数，由小到大排序后输出，用空格分开。

[源程序]

```
#include <iostream> //要使用输入输出流必须的头文件,注意没有“.h”
using namespace std; //要使用 cin 与 cout 必须包括的命名空间
int main(int argc, char *argv[])
{
    int a, b;
    cin>>a>>b; //cin 是标准输入流,此语句从键盘输入两个整数,依次存入 a、b 中
    if(a<b)
        cout<<a<<' '<<b<<endl; //cout 是标准输出流,输出到屏幕
    else
        cout<<b<<' '<<a<<endl; //依次将“<<”连接的各项输出,endl 表示换行
    return EXIT_SUCCESS;
}
```

若从键盘输入：

4 3

则输出

3 4

通过上面的例子，相信您已经对输入输出流的使用有了一个大概的了解。下面一个例子演示如何利用输入输出流对文本文件进行读写操作，您也将从这个例子开始逐渐看到 C++ 语法的方便之处。

[例 2]（文件的输入输出）从文件 in.txt 中输入两个整数，由小到大排序后输出到 out.txt 中，用空格分开。

[源程序]

```
#include <fstream> //要使用文件输入输出流必须的头文件
using namespace std; //要使用 ifstream 与 ofstream 必须的命名空间
ifstream inf("in.txt"); //定义文件输入流 inf,并关联到 in.txt
ofstream outf("out.txt"); //定义文件输出流 outf,并关联到 out.txt
int main(int argc, char *argv[])
{
    int a, b;
```

```
inf>>a>>b; //从 inf 中输入两个整数
if(a<b)
    ouf<<a<<' '<<b<<endl; //向 ouf 输出结果。
else
    ouf<<b<<' '<<a<<endl;
return EXIT_SUCCESS;
}
```

如果 in.txt 的内容为:

4 3

则 out.txt 的内容为:

3 4

由例 2 可以看出,对文本文件的输入输出与标准输入输出从格式上讲是一致的,只是需要定义一下文件输入输出流,且定义时只需说明文件名即可,文件的打开与关闭动作是 C++自动完成的,用户不必处理,这样就不会出现忘记关闭文件而丢失数据的情况了。

二、“//”开始的单行注释。

上面的两个例子中其实已经使用这个语法了,一个语句行中从“//”开始后面的文字都是注释——换行后就不是了,因此这是“单行注释”,与常用的“/*”、“*/”不同。

三、有了 bool 类型。

C++也有“真正”的布尔类型了,就是“bool”,布尔型的变量可以有两个值 true 和 false。其中“bool”、“true”和“false”都成为了 C++的关键字。

四、可以按需要随时定义变量。

在 C 语言中,变量只能定义在所有可执行语句之前,这样,每当我们定义一个新变量时,都要向前翻。在 C++中就不用,您可以“随时随地”地定义变量,当然,同一作用域中的变量仍然是不准重名的。这在下面的例子中可以看到。

五、结构体(struct)的名称、共用体(union)的名称及枚举(enum)的名称都可以直接用做类型名而定义变量。

直接用一个例子来说明即可。

[例 3](结构体语法)从文件 in.txt 中读入十个学生的姓名与成绩,将学生信息按成绩由大到小排序后输出到 out.txt 中,每个学生一行。

[源程序]

```
#include <fstream>
using namespace std;
#define N 10
ifstream inf("in.txt");
ofstream ouf("out.txt");
struct stu //定义结构体 stu
{
    char name[20];
    int score;
};
int main(int argc, char *argv[])
{
    int i;
```

```
stu data[10]; //结构体名称“stu”可以直接用于定义变量
for(i=0;i<N;i++) //输入
    inf>>data[i].name>>data[i].score;
int j; //在使用前定义变量
for(i=0;i<N-1;i++) //以 score 为键值的大数上浮的冒泡排序
    for(j=N-1;j>i;j--)
        if(data[j].score>data[j-1].score)
        {
            stu t;
            t=data[j];
            data[j]=data[j-1];
            data[j-1]=t;
        }
for(i=0;i<N;i++) //输出
    outf<<data[i].name<<' '<<data[i].score<<endl;
return EXIT_SUCCESS;
}
```

如果 in.txt 的内容为:

```
zhang 80
wang 74
li 95
zhao 96
chen 90
xiao 100
qian 80
sun 66
liu 75
dong 80
```

则 out.txt 的内容为:

```
xiao 100
zhao 96
li 95
chen 90
zhang 80
qian 80
dong 80
liu 75
wang 74
sun 66
```

六、new 与 delete。

new 与 delete 都是 C++ 的关键字, 用于动态申请与归还内存, 比 C 语言中原有的 malloc 与 free 函数要方便。

[例 4] (new 与 delete) new 与 delete 的练习。

[源程序]

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    int *p;
    p=new int; //为指针 p 申请一个整数的空间, new 后的类型必须与*p 的类型一致
    *p=3;
    cout<<*p<<endl;
    delete p; //归还空间, 不必指出归还多少
    return EXIT_SUCCESS;
}
```

程序最终输出:

3

new 与 delete 的另一个用途就是动态数组——即事先无法确定大小的数组, 使用时按需分配空间, 这里要用到 C 语言的一个概念: 一维数组名实际是指针变量。

[例 5] (动态数组) 文件 in.txt 中有 n 个学生的姓名和成绩, 请将学生信息按成绩由大到小排序后输出到 out.txt 文件中。每个学生一行。in.txt 文件的结构为: 首行一个整数, 表示学生个数 n, 以下 n 行每行一个姓名与成绩, 用空格分开。

[源程序]

```
#include <fstream>
using namespace std;
ifstream inf("in.txt");
ofstream outf("out.txt");
struct stu
{
    char name[20];
    int score;
};
int main(int argc, char *argv[])
{
    int i, n;
    stu *data; //定义一个 stu 的指针, 用于分配数组
    inf>>n;
    data=new stu[n]; //分配 n 个 stu 的空间, 由 data 指向
    for(i=0; i<n; i++)
        inf>>data[i].name>>data[i].score; //使用时与静态数组没区别
    int j;
    for(i=0; i<n-1; i++)
        for(j=n-1; j>i; j--)
            if(data[j].score>data[j-1].score)
            {
                stu t;
                t = data[j];
                data[j] = data[j-1];
                data[j-1] = t;
            }
    for(i=0; i<n; i++)
        outf<<data[i].name<<" "<<data[i].score<<endl;
    return 0;
}
```

```
t=data[j];
data[j]=data[j-1];
data[j-1]=t;
}
for(i=0;i<n;i++)
    ouf<<data[i].name<<' '<<data[i].score<<endl;
delete [] data; //回收空间, 注意回收数组空间要用[], 不必指定归还大小
return EXIT_SUCCESS;
}
```

七、引用变量

在 C 语言中, 函数的参数可以用数值变量或指针来传递。在 C++ 中, 函数的参数有了第三种传递方法——引用传递。效果上等同于 PASCAL 中的 var 变量, 可以在函数中修改变量的实际参数的值。比如下面的例子就使用了引用参数。

[例 6] (引用参数) 写一个函数, 交换两个变量的值。

[源程序]

```
#include <iostream>
using namespace std;
void swap(int &a, int &b) //引用参数, 形参前加&符号表示引用
{
    int t;
    t=a; //交换变量的值
    a=b;
    b=t;
}
int main(int argc, char *argv[])
{
    int a, b;
    cin>>a>>b;
    cout<<a<<' '<<b<<endl;
    swap(a, b); //实现变量值的交换
    cout<<a<<' '<<b<<endl;
    return EXIT_SUCCESS;
}
```

如果输入

2 3

则输出

2 3

3 2

八、结构体的构造函数与析构函数

构造函数和析构函数原本是“类”中的概念, 但在结构体中也可以使用。大家知道结构体中可以包括各种各样的变量, 称做它的“成员”。在 C++ 中, 结构体里面甚至可以包括函数, 称为它的“方法”。其中最重要的两个方法就是构造函数与析构函数。构造函数是在定义结构体变量 (或用 new 分配空间时) 被自动运行的函数, 而析构函数则是当变量退出作用域 (或用 delete 归还空间) 时被自动运行的函数。因此在

构造函数中通常放入初始化的代码, 比如指针清 NULL 之类, 而在析构函数中, 大多放入清理代码, 比如归还申请的空间等。由于这两种函数是被自动运行的, 因此可以避免结构体成员没有初始化、内存没有回收等问题。

构造函数和析构函数可以象定义变量一样写在结构体中, 但名字是确定的: 构造函数必须与结构体名称相同, 析构函数必须是结构体名称前加 “~” 符号。

下面通过一个例子来说明。

[例 7] (结构体中的构造函数与析构函数) 从输入文件 in.txt 中读入 n ($n \geq 10000$) 个整数, 然后查找 m ($m \geq 5000$) 个整数是否在其中, 并将结果输出到 out.txt 中。in.txt 的结构: 首行两个整数, 分别表示 n 和 m , 其后是 $n+m$ 行, 每行一个整数。前 n 行是原始整数, 后 m 行是待查整数。out.txt 中输出 m 行, 每行一个整数, 表示待查的整数, 空格后是 “IN” 或 “OUT”, 表示该数是否在原始集合中, 输出时请保持输入时 m 个数的前后次序。

[算法分析]

由于原始集合较大, 采用顺序查找明显是不行的。因此决定使用二叉搜索树 (BST, 当然也可以考虑快速排序后的二分或用哈希表)。

[源程序]

```
#include <fstream>
using namespace std;
struct node //二叉树结点结构体
{
    int data;
    node *left, *right;

    node() //构造函数, 使初始化工作自动完成, 函数名与结构体名相同
    {
        left=NULL;
        right=NULL;
    }

    ~node() //析构函数, 使清理工作自动完成, 函数名为结构体名前加 “~”
    {
        left=NULL;
        right=NULL;
    }
};

node *root=NULL; //BST 的根
int n, m;
ifstream inf("in.txt");
ofstream ouf("out.txt");
void build() //构造 BST
{
    int i, d;
    inf>>d;
    root=new node; //先读入一个, 建个 “最小的树”
```

```
root->data=d;
for(i=0;i<n-1;i++) //读入剩余的 n-1 个数, 插入到 BST 中
{
    inf>>d;
    node *temp;
    temp=root;
    while(true)
        if(d>temp->data)//向右枝查找
        {
            if(temp->right!=NULL)
                temp=temp->right;
            else
            {
                temp->right=new node; //插入
                temp=temp->right;
                temp->data=d; //只处理 data 即可, left 与 right
                                //已经在 new 时通过构造函数自动设置为 NULL 了。
                break;
            }
        }
    else if(d<temp->data) //向左枝查找
    {
        if(temp->left!=NULL)
            temp=temp->left;
        else
        {
            temp->left=new node; //插入
            temp=temp->left;
            temp->data=d;
            break;
        }
    }
    else //出现相同整数, 则不再处理
        break;
}

bool search(int d) //查找 BST
{
    node *temp;
    temp=root;
    while(temp!=NULL)
        if(d==temp->data)//找到
            break;
```

```
    else if(d>temp->data)//向右枝查找
        temp=temp->right;
    else//向左枝查找
        temp=temp->left;
    if(temp==NULL)
        return false;
    else
        return true;
}

void clear(node *&root) //删除 BST
{
    if(root==NULL)
        return;
    clear(root->left);
    clear(root->right);
    delete root;//归还空间, 自动执行析构函数
}

int main(int argc, char *argv[])
{
    inf>>n>>m;
    build();
    int i, d;
    for(i=0;i<m;i++)
    {
        inf>>d;
        if(search(d))
            ouf<<d<<' '<<"IN"<<endl;
        else
            ouf<<d<<' '<<"OUT"<<endl;
    }
    clear(root);
    return EXIT_SUCCESS;
}
```

以上是在编写 OI 程序时可能会用到的 C++ 语法, 虽然并不复杂, 却可以使编写过程更加方便, 减少出错的概率。