·C++对C的扩充

·C++的输入输出

流名	含义	隐含设备
cin	标准输入	键盘
cout	标准输出	屏幕
cerr	标准出错输出	屏幕
clog	Cerr的缓冲格式	屏幕

在头文件iostream中定义。

·用const定义常变量

```
const float PI 3.1416; //定义PI为一个常变量，其值无法被修改
```

·函数的重载

简单地来说：一个函数名多用。重载前提，编译器可以分辨使用者是调用的哪个函数。

例子：

```
int find_max(int a,int b);
float find_max(float a,float b);
int find_max(int a);
//int find_max(int a=0,int b);//错误示例

int find_max(int a,int b)
{
    return a?b:a>b;
}
float find_max(float a,float b)
{
    return a?b:a>b;
}
int find_max(int a)
{
    return a;
}

void chongzai()
{
    int a=1,b=2;
```

```
    float c=1.1,d=1.2;
    cout<<"调用int(a,b)，返回值:"<<find_max(a,b)<<endl;
    cout<<"调用float(c,d)，返回值:"<<find_max(c,d)<<endl;
    cout<<"调用int(a)，返回值:"<<find_max(a)<<endl;
}
```

·函数模板

函数模板：建立一个通用函数，其函数类型和形参类型不具体指定，用一个虚拟的类型来代表，这个通用函数就是函数模板。凡是函数体相同的函数都可以用这个模板来代替，不需要定义多个函数，只需要在模板中定义即可。

模板用法：

```
template<typename T>
//+通用函数定义
//或
template<class T>
//+通用函数定义
```

例子：

```
/***函数模板***********/
void moban();
template<class T>
T fmax(T a,T b)
{
    return a?b:a>b;
}

void moban()
{
    int a=2,b=3;
    float c=1.3,d=5.6;
    cout<<"int模板，返回值："<<fmax(a,b)<<endl;
    cout<<"float模板,返回值："<<fmax(c,d)<<endl;
}
```

·有默认参数的函数

如有一函数声明：

```
float area(float r=6.5);
//调用：
area();//相当于area(6.5)
area(7);//相当于area(7)
```

如果有多个形参，可以使每个形参有一个默认值，也可以只对一部分形参指定默认值。

注意：实参与形参的结合是从左至右的，第个实参必然与第一个形参结合，第二个实参必然与第二个形参结合。因此指定默认值的参数必须放在形参表列中的**最右端**！

```
void f1(float a,int b=0,int c,int d=4);//错误!
void f2(float a,int c,int b=0,int d=4);//正确
```

注意：一个函数不能既作为重载函数，又作为有默认参数的函数。因为若如此做，编译器无法识别使用者要调用哪个函数。

·※变量的引用※

1、引用的概念

在C++中，变量的引用就是变量的别名。建立引用的作用是为变量再起一个名字，以便在需要时可以方便、间接地引用该变量。对一个变量的引用的所有操作，实际上都是对其所代表的（原来的）变量的操作。

```
int a;//整形变量a
int &b=a;//定义b为一个整型变量的引用，它被初始化为a
```

经过上述声明后，a和b的作用相同，都代表同一变量。

注意：此处的&不代表地址!

注意：由于引用不是独立的变量，编译系统不给他单独分配存储单元，因此在建立引用时只有声明，没有定义，只是声明它和原有某一变量的关系。

当声明一个变量的引用后，在本函数执行期间，该引用一直与其代表的变量相联系，不能再作为其他变量的别名!

```
int a1,a2;
int &b=a1;//使b称为a1的引用
int &b=a2;//又企图使b成为a2的引用，这是错误的!
```

2、引用的简单使用

```
void yinyong()
{
    int a=3;
    int &b=a;//b为a的引用
    int &c=b;//c为b的引用
    a=4;//改变a
    b=6;//改变b
    c=10;//改变c
    cout<<"a="<<a<<endl;
    cout<<"b="<<b<<endl;
    cout<<"c="<<c<<endl;
}
```

显然，输出结果均为10，因为a这个变量有2个别名b和c

3、关于引用的简单说明

（1）引用不是一种独立的数据类型！它必须与某一种类型的数据相联系。声明引用时必须指定它代表的是哪个变量，即对它初始化。例如：

```
int a;
int &b=a;//正确，使b称为a的别名。
int &b;//错误！！
float a;int &b=a;//错误！类型错误！
```

（2）引用与其代表的变量共享同一内存单元，系统并不为引用另外分配存储空间。实际上，编译系统使引用和其代表的变量都具有相同的地址。

（3）如何区别地址符号与引用？方法：&a的前面又类型符时（如int &a），则它必然是对引用的声明；如果前面没有类型符，则此时的&为取地址符号。

（4）对引用的初始化，可以用一个变量名，也可以用另一个引用。如：

```
int a=3;
int &b=a;//b为a的别名
int &c=b;//c为a的别名。此时a有两个别名：b和c
```

（5）引用在初始化后不能再被重新声明为另一变量的别名。如：

```
int a=3,b=4;
int &c=a;//正确。c为a的别名
c=&b;//错误。企图将c改变为b的别名。
int &c=b;//企图重新声明c为整形变量b的别名，错误。
```

4、将引用作为函数参数

C++之所以增加“引用”，主要是利用它作为函数参数，以扩充函数传递数据的功能。

※传递变量的别名（引用）

例子：

```
void fswap(int &a,int &b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}

void yinyong()
{
    int a1=5;
    int b1=10;
    cout<<"交换"<<endl;
    fswap(a1,b1);
    cout<<a1<<"  "<<b1<<endl;
}
```

程序分析：对引用型形参的初始化是在函数调用时通过虚实结合实现的。当主函数调用fswap时由实参把变量名传给形参。这时，a称为a1的别名，b称为b1的别名，对a和b的更改就是对实参a1和b1的更改，因此在fswap函数中实现了调换，输出结果也为调换后的结果。

实际上，实参传递给形参的是实参的地址，也就是使形参a和变量a1具有同样的地址，从而使a和a1共享同意单元。这里为了便于理解，我们说把变量a1的名字传给引用变量a，使a成为i的别名。

5、对引用的进一步说明

（1）不能建立void类型的引用，如：

```
void &a=b;//错误！！
```

（2）不能建立引用的数组

（3）可以将变量的引用的地址赋给一个指针，此时指针指向的是原来的变量，如：

```
int a=3;//声明一个整型变量。
int &b=a;//声明b为a的引用（别名）
int *p=&b;//指针p指向变量a的引用b，相当于指向a，合法。与int *p=&a;作用相同。
```

如果输出*p的值，就是b的值，也就是a的值。但是不能定义指向引用类型的指针变量。这是因为引用不是一种独立的数据类型。

（4）可以建立指针变量的引用，如：

```
int i=5;
int *p=&i;//p指向i
int* &a=p;//a为一个指向整型变量的指针变量的引用，初始化为p。a就是p的别名。
```

（5）※可以用const对引用加以限定，不允许改变该引用的值。如：

```
int i=5;
const int &a=i;//声明常引用，不允许改变a的值
a=3;//此处改变了a的值，错误！
i=3;//不阻止改变引用所代表的变量的值，所以合法。
```

这一特征再使用引用作为函数形参时时有用的，因为有时希望保护形参的值不被改变。

（6）※可以用常量或表达式对引用进行初始化，但此时必须用const作声明，如：

```
int i=5;
const &a=i+3;//合法
```

上两条语句在编译时，系统将其转换为：

```
int temp=i+3;
const int &a=temp;
```

临时变量在内部实现，用户无法访问。（详情见书P23）

·内联函数

内联函数：嵌入到主调函数中的函数称为内置函数，内联函数，内嵌函数。这是为了提高效率。若某一个函数需要频繁调用，则累计时间会很长，把它写成内联函数可以提高效率。

例子：

```
    inline void print_menu()
    {
        cout<<"1、函数的重载示例"<<endl;
        cout<<"2、函数模板示例"<<endl;
        cout<<"3、有默认参数的函数示例"<<endl;
        cout<<"4、变量的引用示例"<<endl;
        cout<<"5、作用域与运算符示例"<<endl;
        cout<<"6、字符串变量示例"<<endl;
        cout<<"7、动态分配/撤销内存示例"<<endl;
        cout<<"e、退出"<<endl;
    }
```

·命名空间

1、使用命名空间的目的：

（1）目的：多人编程时避免命名重复。

2、命名空间的定义

C++中提出了命名空间的概念：

1.命名空间将全局作用域分成不同的部分，
2.不同命名空间中的标识符可以同名而不会发生冲突
3.命名空间可以发生嵌套
4.全局作用域也叫默认命名空间

语法：

```
namespace Name
{
    namespace Internal
    {
        /*...*/
    }
    /*...*/
}
```

C++命名空间的使用：

使用整个命名空间：using namespace name;
使用命名空间中的变量：using name::variable
使用默认命名空间中的变量: ::variable

例子：

```
#include <stdio.h>
#include <iostream>
namespace First
{
```

```
    int i = 0;
}

namespace Second
{
    int i = 1;

    namespace Internal //嵌套命名空间
    {
        struct P  //嵌套命名空间
        {
            int x;
            int y;
        };
    }
}

int main()
{
    using namespace First; //使用整个命名空间
    using Second::Internal::P;  //使用嵌套的命名空间

    printf("First::i = %d\n", i);
    printf("Second::i = %d\n", Second::i);  //使用命名空间中的变量
    P p = { 2, 3 };

    printf("p.x = %d\n", p.x);
    printf("p.y = %d\n", p.y);
    return 0;
}
```

·作用域运算符

每一个变量都有其有效的作用域，只能在变量的作用域内使用该变量，不能直接使用其他作用域中的变量。

```
#include <iostream>
using namespace std;
float a=13.5;
int main()
{
    int a=5;
    cout<<a;
    return 0;
}
```

上面程序中，输出的a为局部的a，即为5，若想输出全局的a，需要作用域运算符“::”，它能指定所需要的作用域。“::a”表示默认作用域中的变量a。

·字符串变量

1、定义字符串变量

头文件+定义

```
#include <string.h>

int main()
{
    string str1;
    string str2="aoligei";
    return 0;
}
```

2、对字符串变量的赋值

（1）直接使用”=“，（已经过运算符重载）

```
string str1,str2;
str1="aoligei";//将”aoligei“赋值给str2
str2=str1;//用一个字符串变量给另一个字符串变量赋值。
```

注意：在进行上述代码第三行时，字符串长度可以被改变。

（2）单独赋值

```
string str="aoligei";
str[5]='a';
str[6]='n';
//上述代码执行功能：将str：aoligei改为aoligan
```

3、字符串变量的输入输出

直接输入输出

```
cin>>string1;
cout<<string2;
```

4、字符串变量的运算

对string类对象，可以不用strcat、strcmp等函数处理，直接用简单的运算符就可以

（1）字符串复制用赋值号

```
str1=str2;
```

（2）字符串连接用加号

```
str1=str2+str3;
```

（3）字符串比较直接用关系运算符

```
if(str1>str2)
{
    //...
}
```

5、字符串数组

不仅可以用string定义字符串变量，也可以用string定义字符串数组。

```
string name[5];
string name_str[3]={"Gzh","Gogo","lllgogogzh"};
```

·动态分配/撤销内存的运算符

C++提供了比较简便而且功能较强的运算符new和delete来取代malloc和free函数（为了与C语言兼容，仍保留这两个函数）。

（1）new

```
new int;//开辟一个存放整数的空间，返回一个指向整型数据的指针
new int(1000);//开辟一个存放整数的空间，并指定该整数的初始值为1000
new int[1000];//开辟一个存放数组的空间，该数组有1000个元素，返回一个指向整型数据的指针
float *p=new float(3.1416);//开辟一个存放float类型数据的空间，初值为3.1416，将返回的指针值赋给p
```

（2）delete

用法：delete []指针变量

```
int *p=new int(5);
int *parr=new int[10];
delete p;//对单独数据的操作
delete []parr;//对数组的操作，前面加一个方括号
```