C++

笔记本：	我的第一个笔记本		
创建时间：	2020/9/14 19:01	**更新时间：**	2020/9/16 22:02
作者：	1272209351@qq.com		

C++

C语言之父-----里奇

语言（C\C++）、数据结构、操作系统（linux）、网络、数据库、项目
c++98、c++11

C++98 里边有63个关键字

asm	do	if	return	try	continue
auto	double	inline	short	typedef	for
bool	dynamic_cast	int	signed	typeid	public
break	else	long	sizeof	typename	throw
case	enum	mutable	static	union	wchar_t
catch	explicit	namespace	static_cast	unsigned	default
char	export	new	struct	using	friend
class	extern	operator	switch	virtual	register
const	false	private	template	void	true
const_cast	float	protected	this	volatile	while
delete	goto	reinterpret_cast			

命名空间----namespace

一、命名空间的三种方式：
1、（1）、常用方式；（2)、命名空间可以嵌套；
（3）同一个工程中可以定义多个名字相同的命名空间

```cpp
namespace N1
{
    int a = 10;
    int b = 20;

    int Add(int left, int right)
    {
        return left + right;
    }
}
```

```cpp
// 2. 命名空间可以嵌套
namespace N2
{
    int a = 10;
    int b = 20;

    int Sub(int left, int right)
    {
        return left - right;
    }

    namespace N3
    {
        int c = 10;
        int d = 20;

        int Mul(int left, int right)
        {
            return left*right;
        }
    }
```

```cpp
// 3. 在同一工程中，可以定义多个名字相同的命名空间
// 不会冲突
// 编译器会将多个相同名称的命名空间合并成一个
namespace N1
{
    int Div(int left, int right)
    {
        return left / right;
    }
}
```

相同作用域中不能出现相同的变量名字
相同名称的多个命名空间中：也不能出现相同的名字 因为编译器会将多个相同名称的命名空间最终合并成一个

```cpp
namespace N
{
    int a = 10;

    //嵌套的方式----该种方式不会冲突
    // 相当于外层N命名空间中又包含了一个N的命名空间
    namespace N
    {
        int b = 10;
        int a = 10;
    }
}


int main()
{

    return 0;
}
```

2、命名空间的访问方式：

：： 作用域运算符

```
namespace N
{
    int a = 10;
    int b = 20;

    int Add(int left, int right)
    {
        return left + right;
    }
}

int a = 20;

int main()
{
    int a = 30;

    printf("%d\n", a);
    return 0;
}
```

```
int main()
{
    int a = 30;

    // 就近原则
    printf("%d\n", a);

    // 如果访问全局作用域中的a
    // ::作用域运算符
    // ::a  明确说明要访问全局作用域中的a
    printf("%d\n", ::a);
    return 0;
}
```

```
// 访问N命名空间中的a
printf("%d\n", N::a);
```

```
// 该场景：对N命名命名空间中某些成员访问的非常频繁

using N::a;

int main()
{
    // 访问N命名空间中的a
    printf("%d\n", N::a);
    printf("%d\n", N::a);
    printf("%d\n", N::a);
    printf("%d\n", N::a);
    printf("%d\n", N::a);
    printf("%d\n", N::a);
    printf("%d\n", N::a);

    // 为了写代码简单，想要直接访问N命名空间中的a
    printf("%d\n", a);
    return 0;
}
```

using N : : a
该语句加上之后相当于将N 命名空间之内的 a 当成当前文件的一个全局变量来使用

```
// 该场景：对N命名命名空间中某些成员访问的非常频繁
// 优点：写代码简单了
// 缺点：如果该文件中有相同名称的变量或者函数，就会产生冲突
// 如果产生冲突，怎么办？----按照方式1使用即可

// 该条语句加上之后，相当于将N命名空间中的a当成当前文件的一个全局变量来使用
using N::a;


// 如果该文件中也有一个a，必然会产生冲突，只能按照方式1来进行使用
//int a = 10;

int main()
{
    int a = 10;
```

C++中的输入和输出：
C语言的输入输出方式在C++中依旧可以使用-----兼容

```
#pragma  warning (disable:4996)
#include<iostream>
using namespace std ;

cin >> endl;
cout << endl;
```

3、C语言中标准输入输出：
scanf/getchar/gets.....
printf/putchar/puts.....

C++中：
cin>>endl
cout<<endl

注意：std 命名空间是C++语言提供的
cin 和 cout 包含在std 命令空间当中，但不是说std 命名空间中只包含了 cin 和 cout

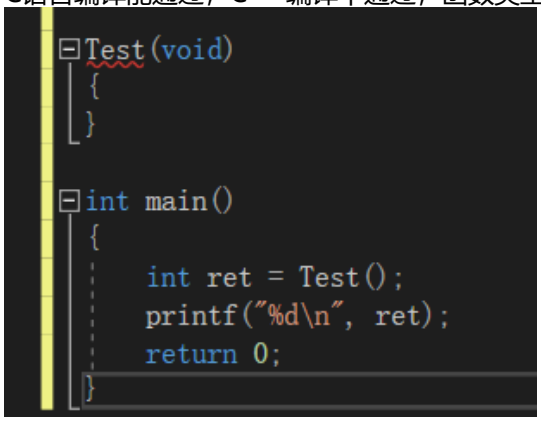
4、C / C++ 的区别

```
// 没有返回值，也没有参数
// 在C语言中：以下代码可以通过编译
// 在C++语言中：以下代码不能通过编译
// C++编译器比C语言编译器语法检测更加严格
void TestFunc()
{

}

int main()
{
    TestFunc();
    TestFunc(10);
    TestFunc(10, 20);
    TestFunc(10,20,30);
    return 0;
```

C语言编译能通过，C++编译不通过，函数类型缺失



```
Test(void)
{
}

int main()
{
    int ret = Test();
    printf("%d\n", ret);
    return 0;
}
```

C和C++关于函数方面的区别:
（1）函数返回值类型
（2）函数参数返回值类型
结论：C++编译器比C语言对函数参数检测更加严格
（3）c++定义函数可以带参数值

```
void TestFunc(int a, int b)
{
    printf("%d %d", a, b);
}

int main()
{
    TestFunc(10, 20);
    TestFunc(10, 20);
    TestFunc(10, 20);
    TestFunc(10, 20);
    TestFunc(10, 20);
    TestFunc(10, 20);
    TestFunc(10, 20);
    TestFunc(10, 20);
    TestFunc(10, 20);

    // 上述调用中，每次调用传递的都是同一组实参
    // 不想传递，但是让该函数执行起来后，还可以拿到10,20
    TestFunc();
    return 0;
}
```

可以在函数形参定义的同时进行赋值（C++能正常运行）

缺省参数：在定义函数同时给参数带上默认值

```
// 缺省参数：在定义函数时，可以给函数的参数待上一些默认值
// 在调用该函数时，如果没有指定实参则采用该默认值，否则使用指定的实参。
void TestFunc(int a=10, int b=20)
{
    printf("%d %d\n", a, b);
}
```

形参 a、b 都带有默认值，用户调用函数时，如果没有传递实参则a 和 b 使用默认值，如果用户传递了实参就使用传递的实参

缺省参数的分类：

（1）全缺省参数：每一个参数都带有默认值

```
void TestFunc(int a=1, int b=2, int c=3)
{
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "c = " << c << endl;
}

int main()
{
    TestFunc();             // 1 2 3
    TestFunc(10);           // 10 2 3?   1   2   10?
    TestFunc(10, 20);
    TestFunc(10, 20, 30);   // 10   20   30
    return 0;
}
```

```
a = 1
b = 2
c = 3
a = 10
b = 2
c = 3
a = 10
b = 20
c = 3
a = 10
b = 20
c = 30
```

（2）半缺省参数：部分参数带有默认值

```
void TestFunc(int a, int b = 2, int c = 3)
{
    cout << a << " " << b << " " << c << endl;
}
```

```cpp
// 2. 半缺省参数：部分参数带有默认值
// a      b      c
// 0      1      1      代码可以通过编译
// 1      0      1      代码编译失败
// 1      1      0      代码编译失败
// 0      0      1      可以通过编译
void TestFunc(int a, int b, int c=1)
{
    cout << a << " " << b << " " << c << endl;
}

int main()
{
    return 0;
}
```

结论：缺省参数只能从右往左依次给出，不能间隔着赋值

缺省参数可以在声明位置给出，也可以在定义位置给出，但是不能两个位置同时给出

5、函数重载

定义：在相同作用域中，函数名字相同 参数列表不同（个数、类型、类型次序）

C语言不支持函数重载

参数列表不同的几种形式：
参数个数不同；参数列表不同；参数类型次序不同

```cpp
// 函数重载

int Add(int left, int right)
{
    return left + right;
}

double Add(double left, double right)
{
    return left + right;
}


int main()
{
    Add(10, 20);
```

```cpp
// 1. 参数个数不同
// 2. 参数列表不同
void TestFunc()
{}

void TestFunc(int a)
{}

void TestFunc(char c)
{}

void TestFunc(int a, char b)
{}

void TestFunc(char a, int b)
{}
```

注意：如果两个函数仅仅只是返回值类型不同则不能构成函数重载

函数重载一定是在编译阶段具体来确定应该调用哪一个函数

函数重载的调用原理：

（1）编译器在编译阶段，会对函数实参类型进行推演，根据推演的结果找类型匹配的函数进行调用；

（2）如果有类型完全匹配的函数则直接进行调用；

（3）如果没有类型完全匹配的函数则会进行隐式类型转换，如果隐式类型转换后有对应函数则进行直接调用，如果没有对应的函数则会报错（或者说转换之后具有二义性------报错）

```cpp
int main()
{
    // 单纯从调用位置来看---只看到了一个Add
    Add(10, 20); // int, int--->

    Add(1.2, 3.4);// double, double--->Add(double, double)

    Add('1', '2'); // char, char--->Add(char,char)--->char和int之间可以进行隐
    return 0;
}
```

如果未定义char类型，会调用int 类型----隐式转换

```
// 编译阶段，对形参类型进行推演：int, double
// Add(int, double)-->该函数没有找到
// 发现：int和double之间可以进行隐式类型转换  int--->double  double--->int
// Add(int, int) || Add(double, double) 转换之后发现有两个函数都可以
// 编译器不知道到底应该调用哪一个方法了
// 编译报错
Add(1, 2.2);
```