C++第一讲

一、课前绪论

1> C++是C语言的扩充，C的所有语法，C++几乎都能使用

2> C语言是面向过程语言，C++是半面向过程半面向对象的，C#是完全面向对象语言

3> C++既可以做面向过程的程序设计，也可以做面向对象的程序设计

1.1 面向对象概念

所谓面向对象，就是将实现某一事物的所有的方法（行为）和属性（成员）都封装到一个整体，这个整体我们称之为类，并且给用户提供一个公共的接口，让用户对该类实例出来的对象进行操作。

例如：一台风扇的生平

天热了，我想要一台风扇，有两种实现方式：

方式一、自己制作风叶、底座、马达、罩子、电线，纯手动制作出来，每个细节全部执行一遍，心知肚明

方式二、到风扇店里买一台风扇，只需要安装说明书正常使用遥控器即可

1.2 C++对C的兼容

1> C++能够实现几乎所有的C功能，但是，C++的编译器相比于C更加严格。

```
#include <iostream>

using namespace std;

void change(void *p)
{
    int *q = (int *)p;     //在C++中，万能指针必须强转后，才能赋值给其他指针变量

    *q = 1314;
}


int main()
{
    int a = 520;

```

```
    change(&a);

    return 0;
}
```

2> C++文件的后缀是.cpp .C .cxx .cc

3> C++的头文件，不以.h结尾，在C++中想用C语言的头文件，可以将对应的.h去掉，在前面加个c即可

4> 在linux下，编译器是g++:

g++ 文件名 -o 可执行文件名

二、第一个C++程序

2.1 hello word

```
#include <iostream>
//包含输入输出的头文件

using namespace std;
//使用系统提供的标准的命名空间
//using关键字，表明使用命名空间的关键字
//namespace 定义命名空间的关键字
//std系统提供的标准命名空间：cout cin   endl

int main()
{
    cout << "Hello World!" << endl;
    //输出语句，cout是标准输出流对象 << 插入运算符，配合cout一起使用
    return 0;
}
```

2.2 输出流对象cout

1> 来自ostream的一个类对象，目前可以理解成跟C的printf一样。

2> 输出数据时，会自动识别类型，不需要格式控制符

```
#include <iostream>

using namespace std;

int main()
{
```

```
    cout << "Hello World!" << "ni hao" << endl;      //输出字符串

    int a = 520;
    cout<<a<<endl;           //输出整形数据

    char b = 'G';
    cout<<b<<endl;           //输出字符数据

    float c = 3.14;
    double d = 1.234;
    cout<<"c = "<<c<<"   d = "<<d<<endl;           //输出浮点型数据

    return 0;
}
```

练习：使用循环输出斐波那契前20项

1 1 2 3 5 8 13 21 。。。

```
#include <iostream>

using namespace std;

int main()
{
    int a = 1;
    int b = 1;         //前两项
    int c;

    cout<<a<<"\t"<<b<<"\t";         //输出前两项

    for(int i=3; i<=20; i++)
    {
        c = a+b;           //推出第三项
        cout<<c<<"\t";

        //更新前两项
        a = b;
        b = c;
    }

    return 0;
}
```

2.3 输入流对象cin

1> 来自于istream的类对象
2> 相当于c中的scanf

```
#include <iostream>

using namespace std;

int main()
{
    char c;
    cout<<"请输入c：";
    cin>>c;                    //输入字符数据
    cout<<" c = "<<c<<endl;

    int a;
    cin>>a;                //整形数据输入
    cout<<a<<endl;

    double b;
    float f;
    cin>>b>>f;             //输入浮点型数据
    cout<<"b = "<<b<<"    f = "<<f<<endl;

    return 0;
}

```

练习：提示并输入一个字符，判断该字符，如果是大写输出对应的小写字母，如果是小写，输出对应的大写，如果是其他字符都输出*

```
#include <iostream>

using namespace std;

int main()
{

    char ch;      //定义字符变量
    cout<<"请输入一个字符：";
    cin>>ch;

    //对ch进行判断

```

```
    if(ch>='a' && ch<='z')
    {
        ch -= 32;
        cout<<"该字符对应的大写为：'<<ch<<endl;
    }else if(ch>='A' && ch<='Z')
    {
        ch += 32;
        cout<<"该字符对应的小写为：'<<ch<<endl;
    }else
    {
        cout<<"*"<<endl;
    }

    return 0;
}
```

2.4 cout格式化输出（了解）

```
#include <iostream>
#include<iomanip>

using namespace std;

int main()
{
    int num = 64;

    cout<<"dec:"<<dec<<num<<endl;           //十进制数据
    cout<<"hex:"<<hex<<num<<endl;           //十六进制数据
    cout<<"oct:"<<oct<<num<<endl;           //八进制数据

    //设置宽度
    cout<<setw(5)<<num<<endl;
    cout<<setfill('*')<<setw(5)<<num<<endl;

    //对于小数保留精度
    //有效数字：从左边第一个不为0的数字开始算起
    double pi = 3.14159265358;
    cout<<"pi = "<<pi<<endl;            //默认最多保留6个有效数字
    cout<<"pi = "<<setprecision(10)<<pi<<endl;          //设置保留10位有效数字

    return 0;
}
```

三、命名空间

3.1 C++中的名字

变量名、函数名、结构体名、类名、数组名、枚举名。。。

3.2 命名空间的作用

解决多人协同开发时，命名冲突问题，有了命名空间后，每个人可以建立自己的名字空间，此时，使用时就不会产生名字冲突问题了

3.3 std命名空间的使用方式

1、 每次使用时，都加上命名空间名和作用域限定符

例如：std::cout<<"hello world"<<std::endl;

2、 在使用之前，将该名字进行提前声明，后面再用的时候，就无需加命名空间，但是没有声明的名字，还是要加上命名空间

例如：using std::cout;

cout<<"hello world"<<std::endl;

3、 提前将整个命名空间全部声明，后期用该空间中的名字时，都不需要加命名空间名和作用域限定符了

例如：using namespace std;

cout<<"hello world"<<endl;

```
#include <iostream>

using namespace std;      //第三种，将整个命名空间全部声明，后面再用时，都不用加命名空间名

using std::cout;       //第二种使用方式

int main()
{
    //int cout = 520;

    std::cout << "hello world" << std::endl;     //命名空间第一种使用方式，每次使用时都加上命名空间名和作用域限定符

    cout<<"hello"<<std::endl;          //方式二

    int a ;
    cin>>a;
    cout<<a<<endl;

```

```
19
20      return 0;
21  }
```

3.4 定义自己的命名空间

格式

```
namespace 空间名{
    名字1;
    名字2;
    。。。
}
```

```
#include <iostream>

using namespace std;

//声明属于zpp的命名空间
namespace zpp {
    int age;
    char sex;
    void show();          //声明函数
}

void zpp::show()          //函数定义
{
    cout<<"age = "<<age<<"   sex = "<<sex<<endl;
}

using zpp::age;    //方式二

using namespace zpp;    //方式三


int main()
{
    zpp::show();         //使用命名空间中的函数名

    zpp::age = 18;         //方式一、使用自己定义的名字空间

    age = 20;          //方式二、已经在上面提前声明了
    sex = 'M';          //方式三、上面已经将整个命名空间全部声明了

```

```
    show();         //使用命名空间中的函数名

    return 0;
}

```

3.5 多个命名空间中名字冲突问题

1> 多个命名空间中使用同一名字，需要在产生冲突的名字前面加上命名空间名和作用域限定符

2> 当命名空间中的名字与全局变量（匿名空间中的名字）冲突时，也要在使用的时候，加上命名空间名和作用域限定符，匿名空间中的名字使用：

：：匿名空间中的名字

3> 当命名空间中的名字与局部变量同名时，优先使用局部变量，如果非要使用命名空间中的名字，也必须加上命名空间名和作用域限定符

```
#include <iostream>

using namespace std;

namespace zpp {
    int age;
    char sex;
    double score;
}

namespace zhangsan {
    int age;
}

//声明命名空间
using namespace zpp;
using namespace zhangsan;

char sex;                   //全局变量的数据会放到匿名空间中

int main()
{
    zpp::age = 18;     //此时需要加上命名空间名和作用域限定符
    zhangsan::age = 20;

    zpp::sex = 'F';         //使用的是命名空间中的自定义变量
    ::sex = 'M';             //使用的是全局变量（匿名空间中）的变量

```

```
    cout<<"zpp::sex = "<<zpp::sex<<"     ::sex = "<<::sex<<endl;

    //定义局部变量
    double score;
    score = 99.99;           //优先使用局部变量
    zpp::score = 88.88;       //使用的是命名空间中的变量

    return 0;
}
```

3.6 名字空间的嵌套定义

```
#include <iostream>

using namespace std;


namespace A {
    int a;
    namespace B {

        int b;
    }
}

//using namespace A;


int main()
{
    A::a = 520;

    A::B::b = 1314;          //需要用作用域限定符一级一级找到对应数据

    return 0;
}
```

3.7 命名空间的总结

1> 使用方式有三种：

每次使用时，都加上命名空间名和作用域限定符

提前将某个名字进行声明，后期再使用时，就无需加命名空间名和作用域限定符

提前将整个命名空间全部声明，后面用到该命名空间中的名字都无需添加

2> 命名冲突：

多个命名空间的名字冲突

命名空间中的名字和匿名空间中的名字冲突

命名空间中的名字和局部变量冲突

3> 命名空间中可以封装函数，一般只在命名空间中声明，在外部定义，定义时在函数名前加空间名和作用域限定符，而不是在函数类型前加

4> 命名空间允许嵌套定义，要使用最深层空间中的名字的话，需要使用作用域限定符一级一级找到最低级进行使用

四、C++中的字符串

4.1 C++支持两种风格的字符串

1> c风格字符串特点：由字符数组存储，以'\0'作为结尾

例如：char name[20] = "zhangpp";

2> C++风格字符串，本质上是类对象

使用要求：需要加上标准命名空间std

有时也要加上#include

4.2 string类型的赋值和初始化

1> 一个变量的赋值和初始化

```
#include <iostream>
#include<string>

using namespace std;

int main()
{
    string s1;          //此时定义了一个字符串变量
    cout<<s1<<endl;           //随机值
    s1 = "hello";
    cout<<"s1 = "<<s1<<endl;

```

```
    string s2 = "world"; //定义一个s2变量，并且用字符串进行初始化
    cout<<"s2 = "<<s2<<endl;

    cout<<"*********************************"<<endl;
    string s3("world");          //定义一个s3变量，并且用字符串进行初始化
    cout<<"s3 = "<<s3<<endl;
    cout<<"*********************************"<<endl;

    string s4(5,'a');       //定义一个s4变量，用五个字符a初始
    cout<<"s4 = "<<s4<<endl;


    return 0;
}
```

2> 多个变量的赋值和初始化

```
#include <iostream>

using namespace std;

int main()
{
    string s1 = "hello";
    cout<<"s1 = "<<s1<<endl;

    //定义一个新的s2字符串变量，用s1初始化
    string s2(s1);
    cout<<"s2 = "<<s2<<endl;

    //定义一个新的s3字符串变量，用s1初始化
    string s3 = s2;
    cout<<"s3 = "<<s3<<endl;

    //定义一个s4，用s1赋值
    string s4;
    s4 = s1;              //在C语言中不允许，只能用strcpy
    cout<<"s4 = "<<s4<<endl;

    //定义一个s5，用s1连接"world"后赋值
    string s5 = s1+"world"+s2;      //此时的+实现了C的strcat
    cout<<"s5 = "<<s5<<endl;


```

```
    return 0;
}
```

4.3 C风格和C++风格的字符串互换

1> C风格字符串转C++风格字符串，无需操作，无缝衔接
2> C++风格的字符串转C的字符串需要调用成员函数c_str()

```
#include <iostream>
#include<cstring>

using namespace std;

int main()
{
    string s = "hello";
    char name[20];

    //将s中的字符串，赋值到name中
    strcpy(name, s.c_str());
    cout<<name<<endl;

    return 0;
}

```

4.4 string类中两个重要函数

1> size()等价于C语言中的strlen函数
2> empty():判空函数

```
#include <iostream>

using namespace std;

int main()
{
    string s1 = "hello world";
    string s2;


```

```
    cout<<"s1的长度："<<s1.length()<<endl;         //11
    cout<<"s2的长度："<<s2.size()<<endl;         //0

    if(s1.empty())
    {
        cout<<"s1为空";
    }else
    {
        cout<<"s1 = "<<s1<<endl;
    }

    ////////////////////////////////////////////////
    if(!s2.empty())
    {
        cout<<"s2 = "<<s2<<endl;
    }else
    {
        cout<<"s2为空";
    }

    return 0;
}

```

4.5 string类型的比较

不需要调用str系列函数，直接可以使用关系运算符进行比较

```
#include <iostream>

using namespace std;

int main()
{
    string s1 = "hello";
    string s2 = "world";

    if(s1>s2)         //在C++中可以使用关系运算符进行字符串的比较
    {
        cout<<"s1>s2"<<endl;
    }else if (s1<s2) {
        cout<<"s1<s2"<<endl;
    }else {
        cout<<"相等"<<endl;
    }
```

```
18
19      return 0;
20  }
```

4.6 string类型的成员访问 at()

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       string s = "hello world";
8
9       s.at(2) = 'G';           //可以更改数据   .at()函数比下标访问更加
    安全，因为有下标越界处理
10      s[1] = 'E';
11
12      for(int i=0; i<s.size(); i++)
13      {
14          //cout<<s[i]<<" ";
15          cout<<s.at(i)<<" ";
16      }
17
18      return 0;
19  }
```

4.7 string数据的输入

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       string s1;
8       string s2;
9
10      cout<<"请输入s1:";
11      cin>>s1;              //不能输入带空格的字符串
12      cout<<"s1 = "<<s1<<endl;
13      getchar();
14
15      //想要输入带空格的字符串，需要调用一个全局函数 getline(cin, s1);
```

```
    getline(cin, s2);
    cout<<"s2 = "<<s2<<endl;

    return 0;
}
```

练习：提示并输入一个字符串，将该字符串中的大写字母变成小写字母，小写字母变成大写字母，其余所有字符都变成"*"后，输出结果

```
#include <iostream>

using namespace std;

int main()
{
    string s1;

    cout<<"请输入一个字符串:"<<endl;
    getline(cin,s1);

    int i = 0;
     if(s1.empty())
        {
            cout<<"字符串为空"<<endl;
        }
        else
        {
             for(i=0;i<s1.size();i++)
            {
                    if(s1.at(i) >= 'A' && s1.at(i) <= 'Z')
                    {
                        s1.at(i) += 32;
                    }
                    else if(s1.at(i) >= 'a' && s1.at(i) <=
'z')
                    {
                        s1.at(i) -= 32;
                    }
                    else
                    {
                        s1.at(i) = '*';
                    }
            }
        }

```

```
36
37      cout<<s1<<endl;
38
39      return 0;
40  }
```

五、bool类型

1> C++支持bool类型，C语言不支持bool类型

2> bool类型的值有两个：1（true）、0（false）

3> 在所有整数中，除了0表示假，其他都是真，包括负数

4> 在C++中，bool类型，默认用数字表示真假，如果想要用单词表示，需要在前面加个boolalpha,之后所有的bool类型，都是用单词表示，如果还想要用数字表示，在前面加个noboolalpha,以后，都是用数字表示的了

5> bool类型的数据，本质上只需要1bit大小就可以存储了，但是，系统还是给其分配了1Byte大小，原因是，内存分配的基本单位是字节

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       bool b1 = 1;
8       cout<<"b1 = "<<b1<<endl;       //1
9
10      bool b2 = 0;
11      cout<<"b2 = "<<b2<<endl;       //0
12
13      bool b3 = 10;
14      cout<<"b3 = "<<b3<<endl;       //1
15
16      bool b4 = -10;
17      cout<<"b4 = "<<b4<<endl;       //1
18
19      /*******************************************************/
20      cout<<"b1 = "<<boolalpha<<b1<<endl;       //true
21      cout<<"b2 = "<<b2<<endl;       //false
22      cout<<"b3 = "<<b3<<endl;       //ture
23      cout<<"b4 = "<<b4<<endl;       //true
24
25      /*******************************************************/
26      cout<<"b1 = "<<noboolalpha<<b1<<endl;       //1
27      cout<<"b2 = "<<b2<<endl;       //0
```

```
28      cout<<"b3 = "<<b3<<endl;       //1
29      cout<<"b4 = "<<b4<<endl;       //1
30       /*********************************************************/
31      bool b5 = true;
32      cout<<"b5 = "<<b5<<endl;       //1
33      bool b6 = false;
34      cout<<"b6 = "<<b6<<endl;       //0
35
36
   /*************************************************************/
37      cout<<"sizeof(bool) = "<<sizeof(bool)<<endl;       //1字节
38
39      return 0;
40  }
41
```

六、引用（reference）

6.1 引用概念

相当于给变量起个别名，例如：宋江的别名叫及时雨、孝义黑三郎。。。

6.2 定义引用

定义格式：类型名 &引用名 = 引用的目标名；

例如：int a = 520;

int &r = a;

&的用途：

1、一个&表示：按位与

2、两个&&表示：逻辑与

3、一个&表示：取地址运算

4、表示引用

区分引用和取地址：看一下左侧有没有数据类型，如果左侧有数据类型，表示引用，其他都是取地址

引用的要求：

1、定义引用时，必须使用引用目标为其初始化

2、系统不会为引用单独开辟空间，引用与其目标是同一空间

3、定义引用时，类型必须与其目标保持一致

4、一个目标，可以定义多个引用

5、引用的目标，一旦指定，就不能进行更改

6.3 引用的基本使用

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 520;

    int &r = a;
    cout<<"a = "<<a<<"   r = "<<r<<endl;
    cout<<"&a = "<<&a<<"   &r = "<<&r<<endl;

    int b = 1314;
    r = b;            //该语句并不是将引用r的目标更换成b，而是用b给r进行赋值
    cout<<"a = "<<a<<"   r = "<<r<<endl;
    cout<<"&a = "<<&a<<"   &r = "<<&r<<"    &b = "<<&b<<endl;

    return 0;
}
```

6.4 引用做形参

```cpp
#include <iostream>

using namespace std;

void swap_1(int m, int n)
{
    int t;
    t=m, m=n, n=t;

    cout<<"m = "<<m<<"  n = "<<n<<endl;      //1314  520
}

void swap_2(int *m, int *n)
{
    int t;
    t=*m, *m=*n, *n=t;

    cout<<"*m = "<<*m<<"  *n = "<<*n<<endl;      //1314  520
}
```

```
20
21
22  void swap_3(int &m, int &n)
23  {
24      int t;
25      t=m, m=n, n=t;
26
27      cout<<"m = "<<m<<"  n = "<<n<<endl;       //520  1314
28  }
29
30
31  int main()
32  {
33      int a = 520;
34      int b = 1314;
35
36      swap_1(a,b);          //调用交换函数
37      cout<<"a = "<<a<<"  b = "<<b<<endl;        //520  1314
38
39      swap_2(&a,&b);          //调用交换函数
40      cout<<"a = "<<a<<"  b = "<<b<<endl;        //1314  520
41
42      swap_3(a,b);          //调用交换函数
43      cout<<"a = "<<a<<"  b = "<<b<<endl;        //520  1314
44
45      return 0;
46  }
47
```

6.5 常引用

6.6 引用做返回值

6.7 结构体中的引用成员（了解）

七、引用和指针的区别