c++要点总结

1 类和对象

类是抽象的，对象是具体的。

类中定义数据成员和成员函数。

main()函数中类的名称跟着对象的名称。或者跟一个数组，需要写明需要多少个元素。

main()函数中类的名称跟一个指针等于一个new出来的内存，也可以申请一个数组。但这种情况切记将内存释放掉。

2 访问对象的各种成员

```
int main()
{
        TV tv;
        tv.type=0;//通过点来访问成员函数
        tv.changeVol();
        return 0;
}
```

```
int main()
{
        TV *p=new TV();
        p->type=0;
        p->changeVol();
        delete p;
        p=NULL;
        return 0;
}
```

例子

```
int main()
{
        TV *p=new TV[5];
        for(int i=0;i<5;i++)
        {
                p[i]->type=0;
                p[i]->changeVol();
        }
        delete []p;
        p=NULL;
        return 0;
}
```

坐标类

数据成员：x和y；

成员函数：分别打印x和y的值

```
#include<iostream>
#include<cstdlib>
using namespace std;
class Coordinate
{
public:
        int x;
        int y;
        void printx()
        {
                cout<<x<<endl;
        }
        void printy()
        {
                cout<<y<<endl;
        }
};
int main()
{
        Coordinate coor;
        coor.x=10;
        coor.y=20;
        coor.printx();
        coor.printy();
        //Coordinate *p=new Coordinate();
```

```
        //if(NULL=p)
        //{return 0;}
        //p->x=100;
        //p->y=200;
        //p->printx();
        //p->printy();
        //delete p;
        //p=NULL;
        return 0;
}
```

3 string

string是一个字符串类型。

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
        string name="ZhangSan";
        string hobby("football");
        cout<<name<<hobby<<endl;
        return 0;
}
```

string的初始化方式

string s1;//s1为空串

string s2("ABC");//用字符串字面值初始化s2

string s3(s2);//将s3初始化为s2的一个副本

string s4(n,'c');//将s4初始化为字符'c'的n个副本

string的常用操作

s.empty()//若s为空串，则返回true，否则返回false,s是一个成员变量

s.size()//返回s中字符的个数

s[n]//返回s中位置为n的字符，位置从0开始

s1+s2//将两个串连成新串，返回新生成的串

s1=s2//把s1的内容替换为s2的副本

v1==v2//判定相等，相等返回true,否则返回false

v1!=v2//判定不等，不等返回true,否则返回false

例子

string s1=" hello";

string s2("world");

string s3=s1+s2;

string s4="hello"+s2;

string s5="hello"+s2+"world";

string s6="hello"+"world";//错误

实例

```
#include<iostream>
#include<cstdlib>
#include<string>
using namespace std;
int main()
{
        string name;
        cout<<"Please input you name:";
        getline(cin,name);//用户直接敲回车就会将空串给name,如果敲得字符串就会把字符串给name
        if(name.empty())
        {
            cout<<"input is NULL"<<endl;
                return 0
        }
        if(name=="imooc")
        {
                cout<<"you are a administrator"<<endl
        }
```

```
        cout<<"hello"+name<<endl;
        cout<<"your name length :"<<name.size()<<endl;
        cout<<"your name first letter is:"<<name[0]<<endl;//因为name[0]是一个char类型的
        //所以不用+直接连接
        return 0;
}
```

4 数据的封装

面向对象：谁做什么

```
class Student
{
public:
        void setAge(int _age){age= _age}//设置年龄的值
        int getAge(){return age;}//获得年龄的值
private:
        string name;
        int age;
        ......
};
```

封装的好处

```
class Car
{
public:
        int getWheelCount(){return m_iWheelCount;}
private:
        int m_iWheelCount;
};
```

实现数据封装例子

```
#include<iostream>
#include<cstdlib>
#include<string>
using namespace std;
class Student
{
public:
        void setName(string _name)
        {
                m_strName=_name;
```

```
        }
        string getName()
        {
                return m_strName;
        }
        void setGender(string _gender)
        {
                m_strGender= _gender
        }
        string getGender()
        {
                return m_strGender;
        }
        int getScore()
        {
                return m_iScore;
        }
        void initScore()
        {
                m_iScore=0;
        }
        void study(int _score)
        {
                m_iScore+=_score;
        }
private:
        string m_strName;
        string m_strGender;
        int m_iScore;
};
int main()
{
Student stu;
stu.initScore();
stu.setName("zhangsan");
stu.setGender("女");
stu.study(5);
stu.study(3);
cout<<stu.getName()<<" "<<stu.getGender()<<" "<<stu.getScore()<<endl;
```

5 类内定义和内联函数

关于内联函数

关键字：inline

```
inline void fun()
{
        cout<<"Hello"<<endl;
}
```

类内定义会首位当做内联函数进行处理，虽然没有写inline。

同文件类外定义

例子

```
class Car
{
public:
        void run();
        void stop();
        void changeSpeed();
};
void Car::run(){}//表明这个函数是属于Car这个类的。
void Car::stop(){}
void Car::changeSpeed(){}
```

分文件类外定义

需要先建立一个.h类型的头文件，建议与文件名一致，在该文件中声明了类的成员函数，在另外一个文件中，把所有的成员函数进行定义，用大括号，要包含头文件

例子:同文件类外定义

```
#include<iostream>
#include<stdlib>
#include<string>
using namespace std;
class Teacher
{
public:
void setName(string _name)
string getName();
void setGender(string _gender)
string getGender();
void setAge(int _age);
int getAge();
void teach();//声明的时候把所有的成员函数都罗列出来
private:
```

```
string m_strName;
string m_strGender;
int m_iAge;
};
void Teacher::setName(string _name)
{
        m_strName=_name;
}
string Teacher::getName()
{
        return m_strName;
}
void teacher::setGender(string _gender)
{
        m_strGender=_gender;
}
string Teacher::getName()
{
        return m_strGender;
}
void Teacher::setAge(int _age)
{
        m_iAge=_age;
}
int Teacher::getAge()
{
        return m_iAge;
}
void Teacher::teach()
{
        cout<<"现在上课..."<<endl;
}
int main()
{
        Teacher t;
        t.setName("孔子");
        t.setGender("男");
        t.setAge(30);
        cout<<t.getName()<<" "<<t.getAge()<<" "<<t.getGender();
        t.teach();
        return 0;
}
```

例子：分文件类外定义

头文件添加新项代码头文件.h使用类名作为头文件名字新建出来teacher.h;然后再新建项选c++文件，名字也叫teacher,点添加。然后把teacher对于类的声明部分剪切到teacher.h里，然后将类的定义部分剪切到teacher.cpp里，再将类的头文件包含进来。

```
#include "teacher.h"
```

6 构造函数详解

内存分区

栈区：

```
int x=0;int *p=NULL;
```

堆区：

```
int *p=new int[20];
```

全局区：存储全局变量及静态变量

常量区：

```
string str="hello";
```

代码区：存储逻辑代码的二进制

例子

类在实例化之前是不会占用堆或者栈中的内存的，实例化之后，每个对象都会在栈上开辟一块内存用来存储各自的数据，但是不同的变量占据着不同的内存，而逻辑代码却只编译出一份放在代码区，供所有对象使用

7 对象初始化

坦克

```
class Tank
{
        private:
                int m_iPosX;
                int m_iPosY;
        public:
                void init()
                {m_iPosX=0;m_iPosY=0;}
```

```
};
int main()
{
        Tank t1;
        t1.init();
        Tank t2;
        t2.init();
        return 0;
}
```

构造函数的规则和特点

构造函数在对象实例化时被自动调用

构造函数与类同名

构造函数没有返回值

构造函数可以有多个重载形式，实例化对象时仅用到一个构造函数，当用户没有定义构造函数时，编译器自动生成一个构造函数。

构造函数的定义

```
class Student
{
public:
        Student(){m_strName="jim";}//构造函数没有参数，函数名和类名相同，
        //构造函数前没有任何返回值，构造函数内部对数据成员进行赋值
private:
        string m_strName;
}

class Student
{
public:
        Student(string name)//这个构造函数的作用是用户实例化student这个对象的时候
        //可以传进来一个name,这个name就可以给数据成员一个初始值，从而初始化m_strName
        {m_strName=name;}
private:
        string m_strName;
};
```

构造函数重载

```
class Student
{
public:
        Student(){m_strName="jim";}
        student(string name)
        {m_strName=name;}
private:
        string m_strName;
};
```

代码展示构造函数使用方法

```
#include<iostream>
#include<cstdlib>
#include<string>
#include"Teacher.h"//声明Teacher类
using namespace std;
int main()
{
        Teacher t1;//调用的是无参的构造函数
        Teacher t2("Merry","15");
        cout<<t1.getName()<<" "<<t1.getAge<<endl;
        cout<<t2.getName()<<" "<<t2.getAge<<endl;
        return 0;
}
```

teacher.h

```
#include<iostream>
#include<string>
using namespace std;
class Teacher
{
public:
        Teacher();//无参构造函数
        Teacher(string name,int age);//有参的构造函数,可以赋初始值
        void setName(string name);
        string getName();
        void setAge(int age);
        int getAge();
private:
        string m_strName;
        int m_iAge;
};
```

teacher.cpp

```
#include "Teacher.h"
Teacher::Teacher()//定义无参的构造函数
{
        m_strName="jim";
        m_iAge=5;
        cout<<"Teacher()"<<endl;
}
Teacher::Teacher(string name,int age)
{
        m_strName=name;
        m_iAge=age;
        cout<<"Teacher(string name,int age)"<<endl;
}
void Teacher::setName(string name)
{
        m_strName=name;
}
string Teacher::getName()
{
        return m_strName;
}
void Teacher::setAge(int age)
{
        m_iAge=age;
}
int Teacher::getAge()
{
        return m_iAge;
}
```

8 默认构造函数

```
int main()
{
        Student stu1();//实例化stu1
        Student *p=NULL;//用p指向了一块空
        p=new Student();//从堆中实例化了一个对象，并用指针去指向它
        return 0;
}

class Student
{
```

```
public:
        Student(){}
        Student(string name="jim")//在这两种情况下实例化对象时都不用给构造函数传递实参，
        //我们把这种在实例化对象时不需要传递参数的构造函数称为默认构造函数
private:
        string m_strName;
};
```

9 构造函数初始化列表

```
class Student
{
public:
        Student():m_strName("jim"),m_iAge(10){}//初始化列表进行初始化，
        //在构造函数的后边用一个冒号隔开，多个数据成员中间用逗号隔开，赋值只能用括号。
private:
        string m_strName;
        int m_iAge;
};
```

初始化列表特性

初始化列表先于构造函数执行

初始化列表只能用于构造函数

初始化列表可以同时初始化多个数据成员

代码展示初始化列表

```
#include<iostream>
#include<cstdlib>
#include<string>
#include "Teacher.h"
using namespace std;
int main()
{
        Teacher t1("Merry",12);//会替换掉默认值
        cout<<t1.getName()<<" "<<t1.getAge()<<endl;
        return 0;
}
```

teacher.h

```
#include<iostream>
#include<string>
using namespace std;
class Teacher
{
public:
        Teacher(string name="jim",int age=1);//两个参数，
        //都有默认值使该构造函数变成默认构造函数
        void setName(string name);
        string getName();
        void setAge(int age);
        int getAge();
private:
        string m_strName;
        int m_iAge;
};
```

teacher.cpp

```
#include"Teacher.h"
Teacher::Teacher(string name,int age):m_strName(name),m_iAge(age)//因为在构造函数声明的时候
//已经加了默认值，所以不在定义的时候加默认值
{
        cout<<"Teacher(string name,int age)"<<endl;
}
void Teacher::setName(string name)
{
        m_strName=name;
}
string Teacher::getName()
{
        return m_strName;
}
void Teacher::setAge(int age)
{
        m_iAge=age;
}
int Teacher::getAge()
{
        return m_iAge;
}
```

10 拷贝构造函数

例子

```
class Student
{
public:
        Student()//默认的构造函数
        {cout<<"Student";}
private:
        string m_strName;
};
int main()
{
        Student stu1;
        Student stu2=stu1;
        Student stu3(stu1);//调用的拷贝构造函数
        //拷贝构造函数定义时与其他的构造函数基本相同，只是参数上
        //有严格的要求
        return 0;
}
```

定义格式

```
//类名（const 类名&变量名）
class Student
{
public:
        Student(){m_strName="jim";}
        Student(const Student&stu){}//拷贝构造函数
private:
        string m_strName;
};
```

如果没有定义拷贝构造函数则系统自动生成一个默认的拷贝构造函数

当系统直接初始化或赋值初始化实例化对象时系统自动调用拷贝构造函数

11 构造函数总结

构造函数分为无参构造函数和有参构造函数，所有的无参构造函数都是默认构造函数，而有参构造函数又分为参数带默认值和参数无默认值，对于有默认值的有参构造函数，如果所有的参数都有默认值，则它就是一个默认的构造函数。

系统自动生成的函数，有普通构造函数和拷贝构造函数，如果我们自定义了，系统都不会自动生成。

而初始化列表都跟在构造函数的后边，无论是跟着那种构造函数，初始化列表都有很重要的地位

12 析构函数

定义格式

类名（）

例子

```
class Student
{
public:
        Student(){cout<<"Student"<<endl;}
        ~Student(){cout<<"~Student"<<endl;}//析构函数不可以加任何参数
private:
        string m_strName;
}
```

13 对象数组

```
int main()
{
        Coordanate coord[3];
        coord[1].m_iX=10;
        Coordinate *p=new Coordinate[3];
        p[0].m_iY=20;
        p->m_iY=20;
        delete []p;
        p=NULL;
        return 0;
}
```

14 对象成员

```
class Coordinate
{
public:
        Coordinate(int x,int y);
private:
        int m_iX;
        int m_iY;
};
class Line
{
public:
        Line(int x1,int y1,int x2,int y2);
private:
        Coordinate m_coorA;
        Coordinate m_coorB;
};
Line::Line(int x1,int y1,int x2,int y2):
        m_coorA(x1,y1),m_coorB(x2,y2)
{
        cout<<"Line"<<endl;
}
int main()
{
Line *p=new Line(2,1,6,4);
delete p;
p=NULL;
return 0;
}
```

15 深拷贝浅拷贝

```
class Array
{
public:
        Array(){m_iCount=5;}
        Array(const Array&arr)
        {m_iCount=arr.m_iCount;}//把arr1的m_iCount赋值给arr2
private:
        int m_iCount;
};
int main()
{
        Array arr1;
        Array arr2=arr1;
        return 0;
```

```
}

class Array
{
public:
        Array(){m_iCount=5;m_pArr=new int[m_iCount];}
        Array(const Array&arr)
        {m_iCount=arr.m_icount;
         m_pArr=new int[m_iCount];
         for(int i=0;i<m_iCount;i++)
         {m_pArr[i]=arr.m_pArr[i];}}//当进行对象拷贝时，
         //不是简单的做值的拷贝，
         //而是将堆中内存的数据也进行拷贝就是深拷贝
private:
        int m_iCount;
        int *m_pArr;//一个指针
};
int main()
{
Array arr1;
Array arr2=arr1;
return 0;
}
```

16 对象指针

例子

```
class Coordinate
{
public:
        int m_iX;
        int m_iY;
}
Coordinate *p=new Coordinate;//p实际上就指向对象中的第一个元素m_iX;
int main()
{
        Coordinate *p=new Coordinate;
        p->m_iX=10;//(*p).m_iX=10;
        p->m_iY=20;//(*p).m_iY=20;
        delete p;
        p=NULL:
        return 0;
}
```

17 对象成员指针

对象的指针作为另外一个类的数据成员。

```
class Coordinate
{
public:
        Coordinate(int x,int y);
private:
        int m_iX;
        int m_iY;
};
class Line
{
public:
        Line();
        ~Line();
private:
        Coordinate *m_pCoorA;
        Coordinate *m_pCoorB;
};
//初始化:
Line::Line():m_pCoorA(NULL),m_pCoorB(NULL)
{
}
//或
Line::Line()
{
        m_pCoorA=NULL;
        m_pCoorB=NULL;
}
//更多是
Line::Line()
{
        m_pCoorA=new Coordinate(1,3);
        m_pCoorB=new Coordinate(5,6);
}
Line::~Line()
{
        delete m_pCoorA;
        delete m_pCoorB;
}
```

18 this指针

例子

```
class Array
{
public:
        Array(int _len){len=_len;}
        int getLen(){return len;}
        void setLen(int _len){len=_len;}
private:
        int len;
};
```

this指针就是指向对象自身数据的指针。相当于它所在对象的地址。

```
Array arr1    this<->&arr1
```

例子

```
class Array
{
public:
        Array(int len){this->len=len;}//来表达数据成员的len
        int getLen(){return len;}
        void setLen(int len){this->len=len;}
private:
        int len;
}
```

例子

```
class Array
{
public:
        Array(T *this,int _len){this->len=_len;}
        int getLen(T *this){return this->len;}
        void setLen(T *this,int _len){this->len=_len;}
private:
 int len;
};
int main()
{
        Array arr1(this,10);->len=10->this->len=10
        arr1.getLen(this);->return len->return this->len
        Array arr2(this,20);->len=20->this->len=20
        arr2.setLen(this,30);->len=30->this->len=30
        return 0;
}
```

19 常对象成员和常成员函数

```
class Coordinate
{
public:
        Coordinate(int x,int y);
private:
        const int m_iX;//常成员
        const int m_iY;
};
coordinate::Coordinate(int x,int y):m_iX(x),m_iY(y)
{
}
```

常成员函数

```
class Coordinate
{
public:
        Coordiante(int x,int y);
        void changeX() const;//常成员函数，不能修改数据成员的值
        void changeX();//两个成员函数互为重载
private:
        int m_iX;
        int m_iY;
};
void Coordinate::changeX() const
{
        m_iX=10;//错误
}
//编译器理解成:
void changeX(const Coordinate *this)
{
        this->m_iX=10;//this指针是常指针不能修改
}
void Coordinate::changeX()
{
        m_iX=20;
}
//常对象调用的是常成员函数
```

20 对象指针和对象引用

```
class Coordinate
{
```

```
public:
        Coordinate(int x,int y);
        int getX();
        int getY();
        void printInfo()const;//常成员函数
private:
        int m_iX;
        int m_iY;
};
int Coordinate::getX()
{return m_iX;)
int Coordinate::getY()
{return m_iY;)
void Coordinate::printInfo()const
{
cout<<"("<<m_iX<<","<<m_iY<<")"<<endl;
}
int main()
{
Coordinate coor1(3,5);
const Coordinate &coor2=coor1;//coor2是coor1的别名,常引用
const Coordinate *pCoor=&coor1;//常指针
coor1.printInfo();
coor2.printInfo();//coor2是一个常引用只有读权限,不能调用get函数
pCoor->printInfo();
return 0;
}
```

例子走出迷宫

```
#include "MyMazeMap.H"
#include "MyMazer.H"
using namespace std;
const int SUCCESS=0;
int main()
{
        int map[8][9]={
                {WALL,WALL,WALL,WALL,WALL,WALL,WALL,ROAD,WALL},
                {WALL,WALL,WALL,WALL,ROAD,WALL,WALL,ROAD,WALL},
                {WALL,WALL,WALL,WALL,ROAD,WALL,WALL,ROAD,WALL},
                {WALL,WALL,WALL,WALL,ROAD,WALL,WALL,ROAD,WALL},
                {WALL,WALL,ROAD,ROAD,ROAD,ROAD,WALL,ROAD,WALL},
                {WALL,WALL,ROAD,WALL,WALL,ROAD,ROAD,ROAD,WALL},
                {WALL,ROAD,ROAD,WALL,WALL,WALL,WALL,WALL,WALL},
                {WALL,ROAD,WALL,WALL,WALL,WALL,WALL,WALL,WALL}
                                        };//定义一个二维数组做迷宫
```

```
        MyMazeMap maze;
        maze.setMazeMap(&map[0][0],8,9);
        maze.setMazeWall('*');
        maze.drawMap();
        MyMazer mazer;
        mazer.setPersonPosition(1,7);//起始位置
        mazer.setPersonSpeed(FAST);
        mazer.setPersonChar('T');
        mazer.start();
        return SUCCESS;
}
void MazePerson::gotoxy(int x,int y)
{
        COORD cd;
        cd.X=x;
        cd.Y=y;
        HANDLE handle=GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleCursorPosition(handle,cd);
};//动画控制
\section{继承:基类派生类}
\paragraph{例子}
\begin{minted}{c++}
class Person
{
public:
        void eat();
        string m_strName;
        int m_iAge;
};
class Worker
{
public:
        void eat();
        void work();
        string m_strName;
        int m_iAge;
        int m_iSalary;
};
//程序优化
class Worker:public Person
{
public:
        void work();
        int m_iSalary;
};
```

代码示例

```
//person.h
#include<string>
using namespace std;
class Person
{
public:
        Person();
        ~Person();
        void eat();
        string m_strName;
        int m_iAge;
};
//worker.h
#include"Person.h"
class Worker:public Person//公有继承
{
        public:
                Worker();
                ~Worker();
                void work();
                int m_iSalary;
};
//person.cpp
#include"person.h"
#include<iostream>
using namespace std;
Person::Person()
{
        cout<<"Person()"<<endl;
}
Person::~Person()
{
        cout<<"~Person()"<<endl;
}
void Person::eat()
{
        cout<<"eat()"<<endl;//表示被调用过
}
//worker.cpp
#include<iostream>
#include"Worker.h"
using namespace std;
Worker::Worker()
{
```

```
        cout<<"Worker()"<<endl;
}
Worker::~Worker()
{
        cout<<"~Worker()"<<endl;
}
Worker::Work()
{
        cout<<"Work()"<<endl;
}
//zhu.cpp
#include<iostream>
#include<cstdlib>
#include"Worker.h"
using namespace std;
int main()
{
        Worker *p=new Worker();
        p->m_strName="jim";
        p->m_iAge=10;
        p->eat();
        p->m_iSalary=1000;
        p->work();
        delete p;
        p=NULL;
        return 0;
}
```

21 继承方式

公有继承，保护继承，私有继承

公有继承

protectd说明符

```
class Person
{
public:
        void eat();
protected:
        int m_iAge;
        string m_strName;
};
int main()
```

```
{
        Person person;
        person.eat();
        person.m_iAge=6;//错误
        return 0;
}
void Person::eat()//在成员函数中可以正常访问
{
        m_strName="jim";
        m_iAge=10;
}
class Worker:public Person//继承
{
public:
        void eat();
        void work(){m_iAge=20;}
protected:
        string m_strName;
        int m_iAge;
        int m_iSalary;
};
```

private成员无法访问

保护继承私有继承

```
class Line
{
public:
        Line(int x1,int y1,int x2,int y2);
private:
        Coordinate m_coorA;
        Coordinate m_coorB;
}//has a关系
```

22 隐藏

父类子类有相同名字的成员函数时，实例化对象时只能访问到子类的该成员函数，相当于父类该成员函数被隐藏了

```
int main()
{
        Soldier.soldier;
        soldier.play();
        soldier.Person::play();
```

```
        return 0;
}
```

即使名字相同参数不同也不能形成重载只能隐藏

23 派生类：is a

```
int main()
{
        Soldier s1;
        Person p1=s1;
        Person *p2=&s1;
        s1=p1;//错误，不能说一个人也是一个士兵
        Soldier *s2=&p1;//错误
        return 0;
}
```

将基类的指针或对象或引用作为函数的参数来使他来接受传入的子类的对象打印出来的都是子类对象的值。

```
void fun1(Person *p)
{
        ......
}
void fun2(Person &p)
{
        ......
}
int main()
{
        Person p1;
        Soldier s1;
        fun1(&p1);fun2(p1);
        fun1(&s1);fun2(s1);
        return 0;
}
```

但是当我们去销毁的时候发现只执行了父类的析构函数没有子类的。这里要用到虚析构函数，当我们存在继承关系的时候我们使用父类的指针指向堆中的子类的对象并且还想用父类的指针来释放这部分内存。

24 多继承和多重继承

一个子类同时有两个父类，或者一个派生类同时有两个基类，就称之为多继承

作为最底层的子类，他如果想实例化自己的对象，他就必须要执行他继承链中的每一个类。然后才执行到自己的构造函数。析构的时候逆序进行。

```
ChildLabourer(string name, int age):Worker(name),Children(age)
```

25 虚继承

```
class Worker:virtual public Person
{
};
class Farmer:virtual public Person
{
};
class MigrantWorker:public Worker,public Farmer
{
};
```

```
#ifndef //假如没定义什么
#dedine
#endif //宏定义避免重定义
```