知识点1【内联函数 inline】（了解）

宏函数（带参数的宏）的缺点：

内联函数条件：

知识点2【函数的默认（缺省）参数】

注意：

1、函数的默认参数从左向右，如果一个参数设置了默认参数，那么这个参数之后的参数都必须设置默认参数

2、如果函数声明和函数定义分开写，函数声明和函数定义不能同时设置默认参数

知识点3【占位参数】（了解）

知识点4【函数重载】c++的多态的特性（重要）

函数重载：同一个函数名在不同场景下可以具有不同的含义。

函数重载意义：方便的使用函数名。

函数重载的条件： 同一个作用域 参数个数不同 参数类型不同 参数顺序不同（重要）

注意：

1、函数的返回值类型 不能作为 函数重载的依据。

2、函数重载和默认参数一起使用，需要额外注意二义性问题的产生

知识点5【c++和c混合编程】c库的

知识点6【C语言中的结构体】

1、知识点的引入

2、封装

把变量（属性）和函数（操作）合成一个整体，封装在一个类中

对变量和函数进行访问控制（公有、私有、保护）

知识点7【类的初识】

知识点1【内联函数 inline】（了解）

宏函数（带参数的宏）的缺点：

第一个在c中也会出现，宏看起来像一个函数调用，但是会有隐藏一些难以发现的错误。

第二个问题是c++特有的，预处理器不允许访问类的成员，也就是说预处理器宏不能用作类的成员函数

内联函数：内联函数为了继承宏函数的效率，没有函数调用时开销，然后又可以像普通函数那样，可以进行参数，返回值类型的安全检查，又可以作为成员函数

内联函数：是一个真正的函数。函数的替换 发生在编译阶段

```
inline int my_mul(int x,int y)
{
 return x*y;
}
void test01()
{
 cout<<"my_mul = "<<my_mul(10+10,20+20)<<endl;
}
```

任何在类内部定义的函数自动成为内联函数。

```
class Person{
public:
      Person(){ cout << "构造函数!" << endl; }
      void PrintPerson(){ cout << "输出 Person!" << endl; }
}
```

内联函数条件：

1、不能存在任何形式的循环语句

2、不能存在过多的条件判断语句

3、函数体不能过于庞大 不能对函数进行取址操作

内联仅仅只是给编译器一个建议，编译器不一定会接受这种建议，如果你没有将函数声明为内联函数，那么编译器也可能将此函数做内联编译。一个好的编译器将会内联小的、简单的函数。

知识点2【函数的默认（缺省）参数】

c++在声明函数原型的时可为一个或者多个参数指定默认(缺省)的参数值，当函数调用的时候如果没有传递该参数值，编译器会自动用默认值代替。

```
//函数的默认参数 指定x的默认值为10 y为20
int my_add(int x=10,int y=20)
{
 return x+y;
}
void test02()
{
 //如果函数传参 那么各自的默认参数将无效
 cout<<"my_add = "<<my_add(100,200)<<endl;//300

 //如果某个参数未被传参 将启用默认值x=100 y使用默认值20
 cout<<"my_add = "<<my_add(100)<<endl;//120

 //x=10 y=20
 cout<<"my_add = "<<my_add()<<endl;//30
}
```

注意：

1、函数的默认参数从左向右，如果一个参数设置了默认参数，那么这个参数之后的参数都必须设置默认参数

```
//函数的默认参数从左向右
int func01(int x,int y=20,int z=30)
{
 return x+y+z;
}
void test03()
{
 cout<<func01(100,200)<<endl;//330
 cout<<func01(100)<<endl;//150
 //cout<<func01()<<endl;//err x没有设置默认参数 必须传参
}
```

2、如果函数声明和函数定义分开写，函数声明和函数定义不能同时设置默认参数

建议：函数声明出设置缺省参数

fun.cpp

```
int func02(int x,int y,int z)
{
 return x+y+z;
}
```

main.cpp

```
//分文件 函数定义处的默认参数 是无效的
//建议:分文件是 在声明 给默认参数
extern int func02(int x,int y=25,int z=35);
//extern int func02(int x,int y,int z);//err
void test04()
{
 cout<<func02(100,200)<<endl;//335
 cout<<func02(100)<<endl;//160
}
```

知识点3【占位参数】（了解）

函数的参数只有类型名 没有形参名，这个参数就是占位参数

由于有类型名 所以 函数调用的时候 必须给占位参数传参。

由于没有形参名 所以 函数内部 是无法使用占位参数。

```
void func03(int x,int y,int)
{
 cout<<"x = "<<x<<", y = "<<y<<endl;
 return;
}
void test05()
{
 //func03(10,30,"hehe");//err "hehe"和int类型不符
 func03(10,30,40);
}
```

知识点4【函数重载】c++的多态的特性（重要）

函数重载：同一个函数名在不同场景下可以具有不同的含义。

函数重载意义：方便的使用函数名。

函数重载的条件： **同一个作用域 参数个数不同 参数类型不同 参数顺序不同（重要）**

```
void myFunc(int a)
{
 cout<<"int的myFunc"<<endl;
}
void myFunc(int a,int b)
{
 cout<<"int,int 的myFunc"<<endl;
}
void myFunc(int a,double b)
 {
  cout<<"int , double的myFunc"<<endl;
 }
 void myFunc(double a,int b)
 {
  cout<<"double,int的myFunc"<<endl;
 }

 void test06()
 {
  myFunc(10);//int
  myFunc(10,20);//int int
  myFunc(10,20.2);//int double
  myFunc(10.1,20);//double int

 }
```

注意：

1、函数的返回值类型 不能作为 函数重载的依据。

```
void myFunc(double a,int b)
{
    cout<<"double,int的myFunc"<<endl;
}
int myFunc(double a,int b)//不能重载
{
    cout<<"double,int的myFunc 返回值类型int"<<endl;
}
```

2、函数重载和默认参数一起使用，需要额外注意二义性问题的产生

```
void myFunc02(int a)
{
 cout<<"int的myFunc02"<<endl;
}
void myFunc02(int a,int b=10)//默认参数
{
 cout<<"int,int 的myFunc02"<<endl;
}
void test07()
{
 //myFunc02(int a) 和 myFunc02(int a,int b=10)都能识别
 myFunc02(10);//二义性产生
}
```

3、函数重载的原理（了解）

3.15.2.2 函数重载实现原理

编译器为了实现函数重载，也是默认为我们做了一些幕后的工作，编译器用不同的参数类型来修饰不同的函数名，比如 void func(); 编译器可能会将函数名修饰成 *func，当编译器碰到 void func(int x),编译器可能将函数名修饰为 funcint,当编译器碰到 void func(int x,char c),编译器可能会将函数名修饰为 funcintchar* 我这里使用”可能”这个字眼是因为编译器如何修饰重载的函数名称并没有一个统一的标准，所以不同的编译器可能会产生不同的内部名。

```
void func(){}
void func(int x){}
void func(int x,char y){}
```

以上三个函数在 linux 下生成的编译之后的函数名为:

```
_Z4funcv //v 代表 void,无参数
_Z4funci //i 代表参数为 int 类型
_Z4funcic //i 代表第一个参数为 int 类型，第二个参数为 char 类型
```

知识点5【c++和c混合编程】c库的

fun.c

```
#include<stdio.h>
int my_add(int x,int y)
{
 return x+y;
}
int my_sub(int x,int y)
{
```

```
 return x-y;
}

```

fun.h

```
#ifndef FUN_H
#define FUN_H
#if __cplusplus
extern "C"{
#endif

 extern int my_add(int x,int y);
 extern int my_sub(int x,int y);


#if __cplusplus
}
#endif

#endif // FUN_H

```

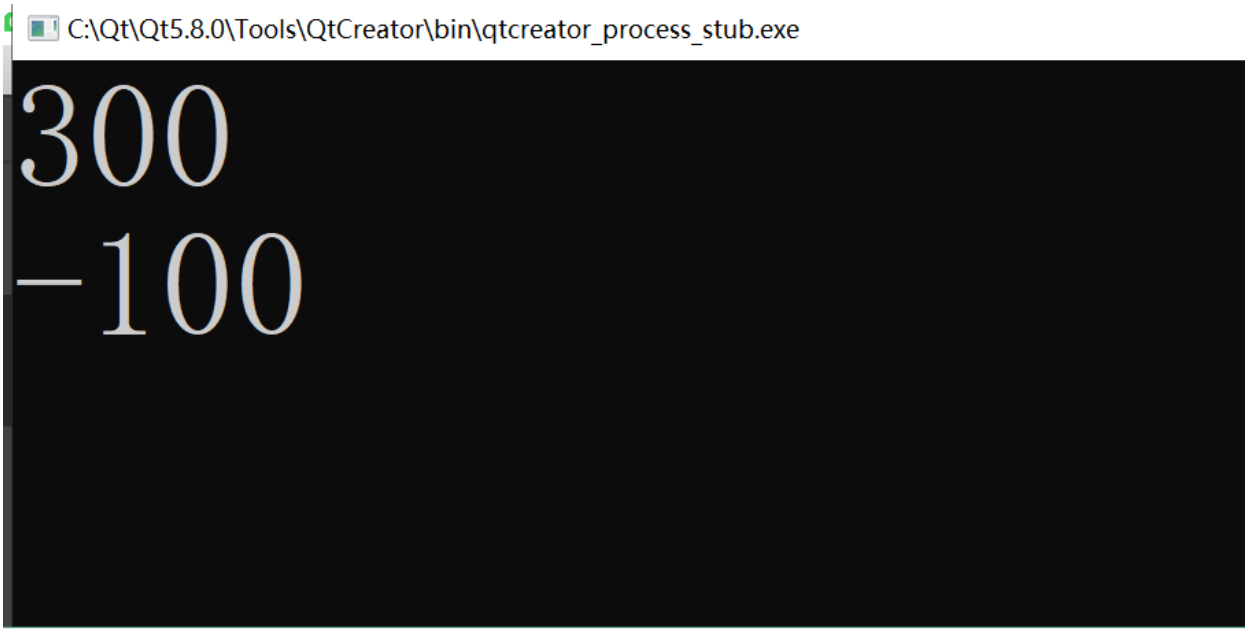
main.cpp

```
#include <iostream>
#include"fun.h"
using namespace std;

int main(int argc, char *argv[])
{
 cout<<my_add(100,200)<<endl;
 cout<<my_sub(100,200)<<endl;
 return 0;
}

```

qtcreater直接运行：



ubuntu:

main.cpp fun.c fun.h

混合编译步骤：

gcc -c fun.c -o fun.o

g++ main.cpp fun.o -o main

知识点6【C语言中的结构体】

1、知识点的引入

```
//c语言的思想:数据 方法 分开
//人
typedef struct
{
 char name[32];
 int age;
}Person;

//动物
typedef struct
{
 char name[32];
 int age;
 int type;
```

```
15 }Dog;
16
17 void PersonEat(Person *p)
18 {
19  cout<<p->name<<"正在吃饭"<<endl;
20 }
21 void DogEat(Dog *d)
22 {
23  cout<<d->name<<"正在吃狗粮, 汪汪"<<endl;
24 }
25
26 void test01()
27 {
28  Person person = {"老王",43};
29  Dog dog={"旺财",6};
30
31  PersonEat(&person);
32  DogEat(&dog);
33
34  // 出现一个问题（数据 方法独立 容易造成 方法 调用错误数据）
35  DogEat((Dog *)&person);
36 }
```

运行结果：

2、封装

把变量（属性）和函数（操作）合成一个整体，封装在一个类中

对变量和函数进行访问控制（公有、私有、保护）

3. 在类的内部(作用域范围内)，没有访问权限之分，所有成员可以相互访问
4. 在类的外部(作用域范围外)，访问权限才有意义：public，private，protected
5. 在类的外部，只有 public 修饰的成员才能被访问，在没有涉及继承与派生时， private 和 protected 是同等级的，外部不允许访问

访问属性	属性	对象内部	对象外部
public	公有	可访问	可访问
protected	保护	可访问	不可访问
private	私有	可访问	不可访问

知识点7【类的初识】

```
class 类名{//抽象的概念 系统不会为其分配空间
 private://私有 类的外部 不可直接访问
 protected://保护 类的外部 不可直接访问
 数据
 public://公有 类的外部 可以直接访问
 方法

 //在类的内部 没有权限之分 都可以相互访问
};
```

```
class Person//抽象的概念
{//类的内部
private:
 int m_money;//私有数据
protected:
 int m_age;
public:
 void dese()
 {
  m_money = 100;
  m_age = 18;
  cout<<"我有房 有车 又年轻"<<m_age<<"岁又有钱"<<m_money<<"万美金 我就爱嘚瑟"<<endl;

 }
};
void test01()
```

```
{
 //用类 去 实例化 一个对象（就是用Person定义一个变量）
 Person lucy;
 //cout<<"兄弟你的钱:"<<lucy.m_money<<endl;//err 内的外部不可访问
 //cout<<"兄弟你的年龄:"<<lucy.m_age<<endl;//err 内的外部不可访问

 lucy.dese();//ok 公有的类的外部可用访问
 //private protected虽然是私有、保护的 类外不可访问 但是用户可以借助 public公有的方法
 //间接的访问私有、保护的数据
}
```

class默认是私有的　数据私有 方法公有 用户就可以借助 公有方法 间接的操作 私有数据

知识点8【Person类的设计】

```
#include <iostream>
#include<cstring>
using namespace std;
class Person//抽象的
{
private:
 char m_name[32];//定义类的时候 不要给成员初始化
 int m_age;
public:
 //初始化函数
 void initPerson(char *name, int age)
 {
 if(name != NULL)
 strcpy(m_name,name);

 if(age>0 && age <100)
 m_age = age;
 }
 //对m_name进行写操作
 void setName(char *name)
 {
 if(name != NULL)
 strcpy(m_name,name);
 }
 //对m_name进行读操作
 char* getName(void)
```

```
27  {
28  return m_name;
29  }
30
31  //对m_age进行写操作
32  void setAge(int age)
33  {
34  if(age >0 && age <100)
35  m_age = age;
36  else
37  cout<<"age无效"<<endl;
38  }
39  //对m_age进行读操作
40  int getAge(void)
41  {
42  return m_age;
43  }
44
45  void showPerson(void)
46  {
47  cout<<"姓名: "<<m_name<<", 年龄: "<<m_age<<endl;
48  }
49
50 };
51
52 int main(int argc, char *argv[])
53 {
54  //通过类 实例化 一个对象（用Person定义一个变量）
55  Person lucy;//Person是类名称 抽象的 lucy是一个对象 实例（实际存在）
56  //对象 只能借助 公有方法 操作私有数据
57  //初始化
58  lucy.initPerson("lucy",18);
59  //获取姓名
60  cout<<"姓名:"<<lucy.getName()<<endl;
61  //获取年龄
62  cout<<"年龄:"<<lucy.getAge()<<endl;
63  //更改 年龄
64  lucy.setAge(200);
65  lucy.setAge(28);
66
```

```
67   //遍历lucy的信息
68   lucy.showPerson();
69
70   return 0;
71  }
72
```

运行结果：

知识点9【立方体类的设计】

```
1  #include <iostream>
2
3  using namespace std;
4  class Cub
5  {
6  private://确定数据
7   int m_l;
8   int m_w;
9   int m_h;
10 public:
11  //m_l m_w m_h有写操作
12  void setL(int l)
13  {
14  m_l = l;
15  }
16  void setW(int w)
17  {
18  m_w = w;
19  }
20  void setH(int h)
```

```
21  {
22  m_h=h;
23  }
24
25  //m_l m_w m_h有读操作
26  int getL(void)
27  {
28  return m_l;
29  }
30  int getW(void)
31  {
32  return m_w;
33  }
34  int getH(void)
35  {
36  return m_h;
37  }
38
39  //计算面积
40  int getS(void)
41  {
42  return 2*(m_l*m_w+m_l*m_h+m_w*m_h);
43  }
44
45  //计算体积
46  int getV(void)
47  {
48  return m_l*m_w*m_h;
49  }
50
51  //成员函数判断
52  bool myCompareCub2(Cub &ob)//类的内部 没有权限之分
53  {
54  if(m_l == ob.m_l && m_w == ob.m_w && m_h == ob.m_h)
55  return true;
56
57  return false;
58  }
59
60 };
61
```

```
62 //全局函数
63 bool myCompareCub1(Cub &c1, Cub &c2)
64 {
65  //类的外部 必须使用getL获取长度
66  if((c1.getL() == c2.getL()) && (c1.getW() == c2.getW()) && (c1.getH() =
= c2.getH()))
67  {
68  return true;
69  }
70
71  return false;
72
73 }
74 int main(int argc, char *argv[])
75 {
76
77  Cub cub1;//实例化一个对象
78  cub1.setL(10);
79  cub1.setW(10);
80  cub1.setH(10);
81
82  //计算面积
83  cout<<"cub1的面积:"<<cub1.getS()<<endl;
84  //计算体积
85  cout<<"cub1的体积:"<<cub1.getV()<<endl;
86
87  Cub cub2;//实例化一个对象
88  cub2.setL(10);
89  cub2.setW(20);
90  cub2.setH(10);
91
92  //全局函数
93  if(myCompareCub1(cub1,cub2) == true)
94  {
95  cout<<"相等"<<endl;
96  }
97  else
98  {
99  cout<<"不相等"<<endl;
100  }
101
```

```
 Cub cub3;
 cub3.setL(10);
 cub3.setW(10);
 cub3.setH(10);
 //成员函数
 if( cub3.myCompareCub2(cub1) == true)
 {
 cout<<"相等"<<endl;
 }
 else
 {
 cout<<"不相等"<<endl;
 }

 return 0;
}

```

知识点10【设计一个点和圆的类】

```
#include <iostream>

using namespace std;
//设计点的类
class Point
{
private:
 int m_x;
 int m_y;
public:
 void setX(int x=0)
 {
 m_x = x;
 }
 void setY(int y=0)
 {
 m_y = y;
 }

 int getX(void)
```

```
21  {
22  return m_x;
23  }
24  int getY(void)
25  {
26  return m_y;
27  }
28 };
29
30 //设计圆的类
31 class circle
32 {
33 private:
34  Point m_p;//圆心
35  int m_r;//半径
36 public:
37  void setR(int r)
38  {
39  m_r = r;
40  }
41  int getR(void)
42  {
43  return m_r;
44  }
45
46  void setPoint(int x,int y)
47  {
48  m_p.setX(x);
49  m_p.setY(y);
50  }
51  Point getPoint(void)
52  {
53  return m_p;//得到圆心
54  }
55
56  //判断 点 在 圆的位置
57  void isPointOnCircle(Point &ob)
58  {
59  //点到圆心的距离
60  int distance = (m_p.getX()- ob.getX())*(m_p.getX()- ob.getX())+\
```

```
61   (m_p.getY()-ob.getY())*(m_p.getY()-ob.getY());
62   if(distance < m_r*m_r)
63   {
64   cout<<"点在圆内"<<endl;
65   }
66   else if(distance == m_r*m_r)
67   {
68   cout<<"点在圆上"<<endl;
69   }
70   else
71   {
72   cout<<"点在圆外"<<endl;
73   }
74
75   }
76 };
77 int main(int argc, char *argv[])
78 {
79   //实例化一个点对象
80   Point point;
81   point.setX(20);
82   point.setY(20);
83
84   //实例化一个圆的对象
85   circle cir;
86   cir.setPoint(5,5);//设置圆心
87   cir.setR(5);//设置半径
88
89   //判断点point 与 圆的关系
90   cir.isPointOnCircle(point);
91   return 0;
92 }
93
```

运行结果：


C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
点在圆外