§4. 函数

4.13. 内联函数

形式：

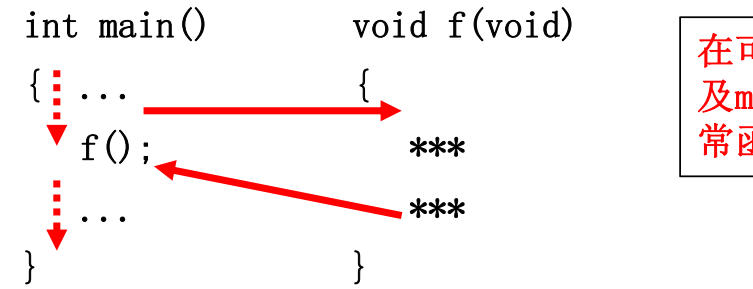
```
inline 返回类型 函数名（形式参数表）
{
   函数体
}

inline int max(int x, int y)
{
    return x>y?x:y;
}
```

§4. 函数

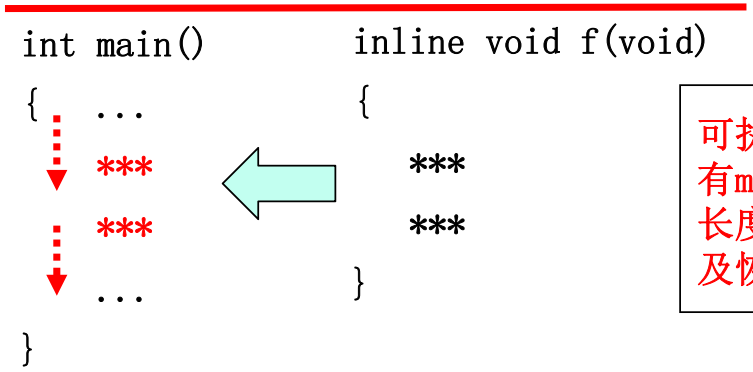
4.13. 内联函数

使用：

★ 不单独编为一段代码，而是直接插入每个调用处，调用时不按函数调用过程执行，而是直接将该函数的代码放在调用处顺序执行



在可执行代码中有f()及main()的存在，按正常函数调用方式进行



可执行代码中无f(),仅有main(),main()代码长度增加，但没有保存及恢复现场的消耗

§4. 函数

4.13. 内联函数

```
int main()
{   ...
    f();
    ...
    f();
    ...
    f();
    ...
    f();
    ...
}
```

```
inline void f(void)
{
   cout ...;
}
```

假设f()被调用了10000次

相当于

```
int main()
{   ...
    cout ...
    ...
    cout ...
    ...
    cout ...
    ...
    cout ...
    ...
}
```

```
inline void f(void)
{
   cout ...;
}
```

执行代码中f()已不存在

可执行代码中无f(),仅有main(),main()代码中包含了10000份f()的代码，长度增加，但没有保存及恢复现场的消耗

以空间的增加换取时间的加快

问：为什么不去掉f()，直接在main()中写10000次cout？

答：和前面定义符号常量一样
#define pi 3.14159
便于源程序的修改和维护

§4. 函数

4.13. 内联函数

使用：

★ 不单独编为一段代码，而是直接插入每个调用处，调用时不按函数调用过程执行，而是直接将该函数的代码放在调用处顺序执行

★ 可执行程序的代码长度增加，但执行速度加快，适用于函数体短小且调用频繁的情况（1-5行）
（保存/恢复现场的代价超过函数体自身代价的情况）

★ 不能包含分支、循环等复杂的控制语句

★ 系统编译时会自动判断是否需要真正采用内联方式
（写了inline，最终也不一定真正成为内联函数）

★ 递归函数不能内联(递归必须要保存/恢复现场)

★ 允许只在函数声明或函数定义中加inline，也可以同时加

不同编译器，三种情况可能都正确/部分正确(VS/Dev中三种都正确)

```
inline void fun();
int main()
{
   ...
}
inline void fun()
{
   ...
}
```

```
inline void fun();
int main()
{
   ...
}
~~inline~~ void fun()
{
   ...
}
```

```
~~inline~~ void fun();
int main()
{
   ...
}
inline void fun()
{
   ...
}
```

§4. 函数

4.13. 内联函数

使用：

★ inline函数及调用函数必须在同一个源程序文件中，否则编译出错

（普通函数可以放在不同源程序文件中）

```
//ex1.cpp
inline void fun();
int main()
{
   ...
}
```

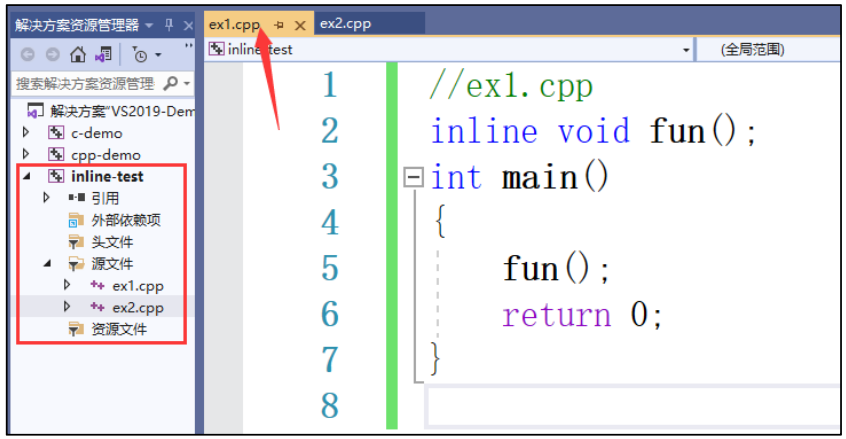
```
//ex2.cpp
inline void fun()
{
   ...
}
```

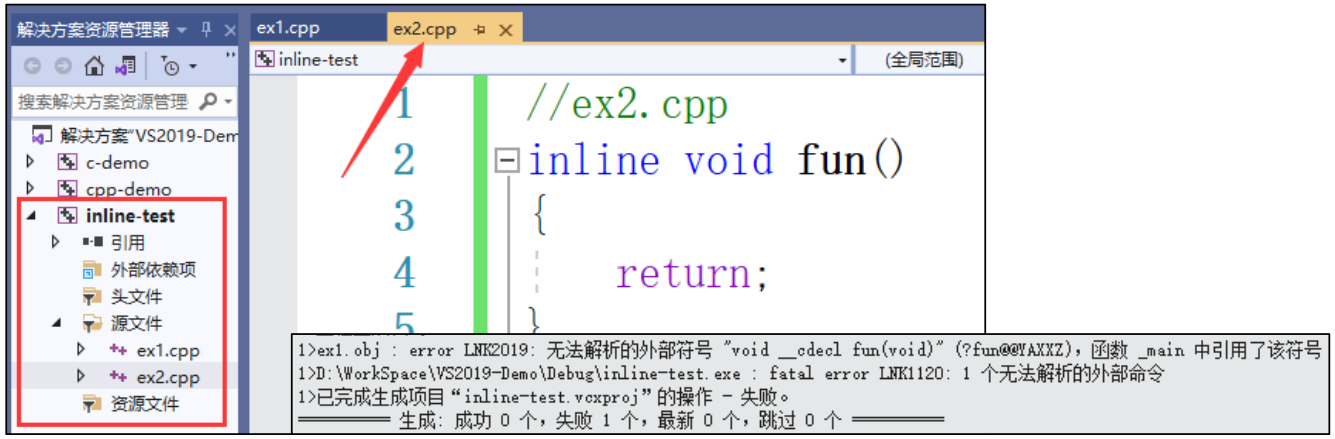
假设ex1.cpp和ex2.cpp共同构成一个可执行文件，则编译出错

```
//ex1.cpp
void fun();
int main()
{
   ...
}
```

```
//ex2.cpp
void fun()
{
   ...
}
```

假设ex1.cpp和ex2.cpp共同构成一个可执行文件，则编译正确





§4. 函数

4.14. 函数的重载

重载：同一作用域中多个函数使用相同的名称

引入：对同一类功能的实现，仅参数的个数或类型不同，希望采用相同的函数名

C不允许
C++允许

```
      int   imax(int x,   int y);
      float fmax(float x, float y);
      long  lmax(long x,  long y);
====> 希望 imax/fmax/lmax 都叫 max ?
      int   max(int x,   int y);
      float max(float x, float y);
      long  max(long x,  long y);
```

```
      int max2(int x, int y);
      int max3(int x, int y, int z);
      int max4(int x, int y, int z, int w);
====> 希望 max2/max3/max4 都叫 max ?
      int max(int x, int y);
      int max(int x, int y, int z);
      int max(int x, int y, int z, int w);
```

例：分别求两个int和double型数的最大值

```
int max(int x, int y)
{   cout << sizeof(x) << endl;
    return (x > y ? x : y);
}
double max(double x, double y)
{   cout << sizeof(x) << endl;
    return (x > y ? x : y);
}
int main()
{   cout << max(10,   15)   << endl;
    cout << max(10.2, 15.3) << endl;
}
```

?

例：分别求两个/三个int数的最大值

```
int max(int x, int y)
{   cout << 2 << ' ';
    return (x > y ? x : y);
}
int max(int x, int y, int z)
{   cout << 3 << ' ';
    int t = (x > y ? x : y);
    return (t > z ? t : z);
}
int main()
{   cout << max(10, 17)    << endl;
    cout << max(23, 15, 8) << endl;
}
```

?

§4. 函数

4.14. 函数的重载

重载：同一作用域中多个函数使用相同的名称

引入：对同一类功能的实现，仅参数的个数或类型不同，希望采用相同的函数名

重载函数调用时的匹配查找顺序：

（1）寻找参数个数、类型完全一致的定义(严格匹配)

（2）通过系统定义的转换寻找匹配函数

（3）通过用户定义的转换寻找匹配函数

★ 若某一步匹配成功，则不再进行下一顺序的匹配

★ 若某一步中发现两个以上的匹配则出错

例：分别求两个int和double型数的最大值

```
#include <iostream>
using namespace std;
int max(int x, int y)
{   cout << sizeof(x) << ' ';
    return (x > y ? x : y);
}
double max(double x, double y)
{   cout << sizeof(x) << ' ';
    return (x > y ? x : y);
}
int main()
{   cout << max(10,   15)     << endl;    //int, int
    cout << max(10.2, 15.3)   << endl;    //double, double
    cout << max(10,   int(15.3)) << endl; //int,double
    cout << max(5+4i, 15.3)   << endl;    //复数，double
    return 0;
}
```

1

2

哪句语句编译会错?
其它正确语句的输出是什么？

复数形式目前编译会错，如何定义复数以及定义复数向double的转换，具体见荣誉课相关内容

严格匹配1
严格匹配2
系统转换1
需自定义转换

自行将max的参数换成U/L/F等不同组合，看是否报错，按什么类型做系统转换

§4. 函数

4.14. 函数的重载

使用：

★ 要求同名函数的参数个数、参数类型不能完全相同

`void fun(int x, int y);`	正确
`void fun(int x, int y, int z);`	参数个数不同,类型同
`void fun(int x, int y);`	正确
`void fun(long x, long y);`	参数个数同,类型不同
`void fun(int x, int y);`	正确
`void fun(long x, long y, long z);`	个数类型均不同
`void fun(int x, int y);`	错误
`void fun(int x, int y);`	个数类型均相同

★ 返回类型及参数名不做检查(仅这两个不同认为错)

`int  max(int x, int y);`	错误,仅返回类型不同
`long max(int x, int y);`	参数类型个数完全相同
`int  max(int x, int y);`	错误,仅参数名不同
`int  max(int p, int q);`	参数类型个数完全相同

★ 若参数类型是由typedef定义的不同名称的相同类型，则会产生二义性

`typedef  INTEGER  int;`	相当于给int起个别名叫INTERGER，具体见第7章
`int  fun(int a);`	错误
`INTEGER  fun(INTEGER a);`	

★ 尽量使同名函数完成相同或相似的功能，否则可能导致概念混淆

§4. 函数

4.15. 函数模板

函数重载的不足：对于参数个数相同，类型不同，
而实现过程完全相同的函数，
仍要分别给出各个函数的实现

问题:两段一样的代码
能否合并为一段？

```
int max(int x, int y)
{
    return x>y?x:y;
}
double max(double x, double y)
{
    return x>y?x:y;
}
```

函数模板：建立一个通用函数，其返回类型及参数类型不具体指定，用一个虚拟类型来代替，该通用函数称为函数模板，调用时再根据不同的实参类型来取代模板中的虚拟类型，从而实现不同的功能

一段代码,两个功能
1、两个int型求max
2、两个double型求max

问题1：如果传入两个unsigned int型数据，
T的类型被实例化为什么？如何证明？
问题2：如果x,y的类型不同，自行摸索转换规律

```
#include <iostream>
using namespace std;
template <typename T>
T max(T x, T y)
{   cout << sizeof(x) << ' ';
    return x>y?x:y;
}
int main()
{   int    a=10, b=15;
    double f1=12.34, f2=23.45;
    cout << max(a, b) << endl;
    cout << max(f1, f2) << endl;
    return 0;
}
```

```
4 15
8 23.45
```

§4. 函数

4.15. 函数模板

函数重载的不足：对于参数个数相同，类型不同，而实现过程完全相同的函数，仍要分别给出各个函数的实现

函数模板：建立一个通用函数，其返回类型及参数类型不具体指定，用一个虚拟类型来代替，该通用函数称为函数模板，
调用时再根据不同的实参类型来取代模板中的虚拟类型，从而实现不同的功能

使用：

★ 仅适用于参数个数相同、类型不同，实现过程完全相同的情况

★ typename可用class替代

★ 类型定义允许多个

```
template <typename T1, typename t2>
template <class T1, class t2>
```

```
#include <iostream>
using namespace std;
template <typename T1, typename T2>
char max(T1 x, T2 y)
{   cout << sizeof(x) << ' ';
    cout << sizeof(y) << ' ';
    return x>y ? 'A' : 'a';
}
int main()
{   int    a = 10, b = 15;
    double f1 = 12.34, f2 = 23.45;
    cout << max(a, f1) << endl;
    cout << max(f2, b) << endl;
    return 0;
}
```

?

问：max(a,f1)时，T1/T2类型分别是?
max(f2,b)时，T1/T2类型分别是?

§4. 函数

4.16. 有默认参数的函数

引入：假设已经定义了某个函数，并进行了大量的应用后来随着要求的增加，需要扩充函数的功能并且增加相应的参数来满足扩充的功能

例：定义 circle(int x, int y)用于画圆心在(x,y)处半径为10的圆，并已被调用1000次

```
int main()
{
   ...
   circle(...);
   ...
   circle(...);
   ...
   circle(...);
   ...
   circle(...);
   ...
}  //有1000次调用
void circle(int x, int y)
{
    //具体实现过程
}
```

经过不断的测试，程序已稳定运行

例：增加要求，要求半径可变，前面已调用的1000次中900次维持半径为10不变，100次改为不同值，又新增调用1000次

```
int main()
{
   ...
   circle(.旧.);
   ...
   circle(.旧.);
   ...
   circle(.新.);
   ...
   circle(.新.);
   ...
}
```

```
void circle(int x,int y,int r)
{
    //具体实现过程
}
```

首先：修改circle的定义及实现

其次：修改旧的1000次调用语句，从两参改为三参

最后：新增1000次三参调用

§4. 函数

4.16. 有默认参数的函数

例：一个程序要求的不断演变

1、定义 circle(int x, int y)用于画圆心在(x,y)处半径为10的圆，并已被调用1000次

2、增加要求，要求半径可变，前面已调用的1000次中900次维持半径为10不变，100次改为不同值，又新增调用1000次

3、增加要求，要求指定不同的颜色，前面已调用过的2000次中1800次保持白色，200次改为其它颜色，又新增调用1000次

4、新增要求，要求指定空心还是实心，前面已调用过的3000次中2700次都是空心，300次改为实心，又新增调用1000次

工程思维的基本概念：

1、使程序稳定运行所需要的测试工作工作量很大

2、一旦修改了程序，原来稳定运行的部分也可能出现各种问题，需要重新测试

3、新功能的增加是必须的

问题：能否在功能增加的同时使程序的改动尽可能少？

§4. 函数

4.16. 有默认参数的函数

引入：假设已经定义了某个函数，并进行了大量的应用后来随着要求的增加，需要扩充函数的功能并且增加相应的参数来满足扩充的功能

含义：对函数的某一形参，大部分情况下都对应同一个实参值时，可以采用默认参数(默认值建议为常量)

形式：

返回类型 函数名(无默认参数形参,有默认参数形参)
{
    函数体
}

```
void circle(int x, int y, int r=10)
{
    ...
}
```

调用：circle(0,0); ⇔ circle(0,0,10);
circle(5,8,12);

§4. 函数

4.16. 有默认参数的函数

使用：

★ 便于函数功能的扩充，减少代码维护，修改的数量

针对刚才的例子：

=> 1、两个参数的原始程序完成，调用1000次

```
void circle(int x, int y)
```

=> 2、加半径参数，不变900处，改100处，增1000处

```
void circle(int x, int y, int r=10)
```

=> 3、加颜色参数，不变1800处，改200处，增1000处

```
void circle(int x, int y, int r=10, int color=WHITE)
```

=> 4、加填充参数，不变2700处，改300处，增1000处

```
void circle(int x, int y, int r=10, int color=WHITE, int filled=NO)
```

结论：

1、有默认参数的函数，能有效地减少了修改次数，减少了工作量

2、最好的方法，是在初始设计函数时，就考虑到更多可能的因素
（包括客户暂时未想到的问题）

§4. 函数

4.16.有默认参数的函数

使用：

★ 便于函数功能的扩充，减少代码维护，修改的数量

★ 允许有多个默认参数，但必须是连续的最后几个

```
void circle(int y, int x=0, int r=5)(对)
void circle(int x=0, int y, int r=5)(错)
```

★ 若有多个默认参数，调用时，前面使用缺省值，后面不使用缺省值，则前面也要加上

```
void circle(int x,int y,int r=5,int c=WHITE)
circle(10, 15);
circle(10, 15, 10);
circle(10, 15, 12, BLUE);
circle(10, 15, 5, BLUE);
```

虽然是缺省，也要加

§4. 函数

4.16. 有默认参数的函数

使用：

★ 若函数定义在调用函数之后，则声明时必须给出默认值，定义时不再给出

```
void circle(int x, int y, int r=10);
int main()
{   ...
}
void circle(int x, int y, int r)
{   ...
}
```

正确

```
void circle(int x, int y, int r=10);
int main()
{   ...
}
void circle(int x, int y, int r=10)
{   ...
}
```

错误，即使相同

```
void circle(int x, int y, int r);
int main()
{   ...
}
void circle(int x, int y, int r=10)
{   ...
}
```

错误

```
void circle(int x, int y, int r=10);
int main()
{   ...
}
void circle(int x, int y, int r=5)
{   ...
}
```

错误

★ 重载与带默认参数的函数一起使用时，可能会产生二义性

```
int fun(int a, int b=10);
int fun(int a);
```

若调用为：fun(10，20)　正确

fun(50)　二义性