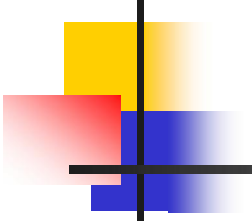


## 第8章 函 数



本章学习目标:

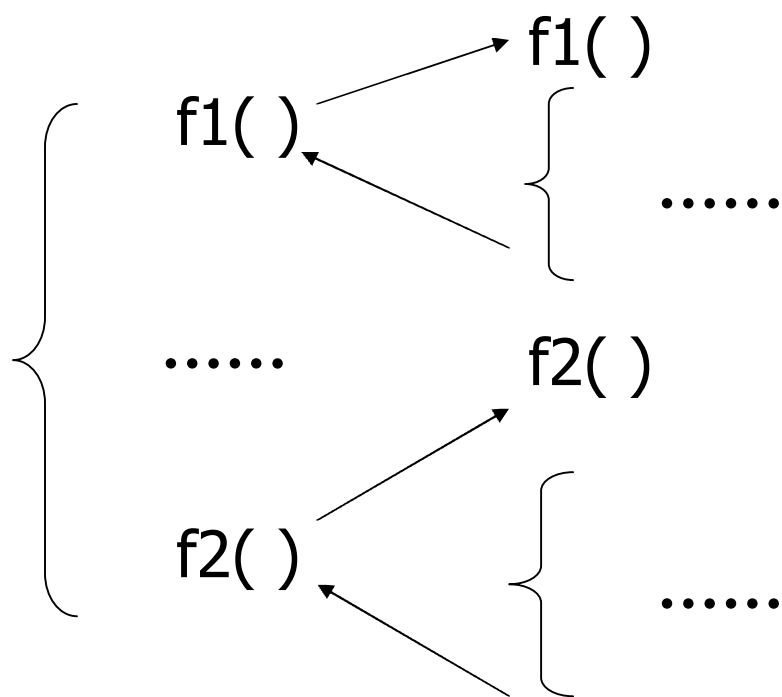
---

1. 什么是函数及函数的组成。
2. 怎样声明和定义函数。
3. 怎样向函数传递参数。
4. 怎样从函数返回一个或多个值。
5. 传值和传址有何区别。Return的作用。
6. 理解C++在函数原型中声明引用的目的，能正确使用引用。
7. 局部变量、全局变量、静态变量的区别。

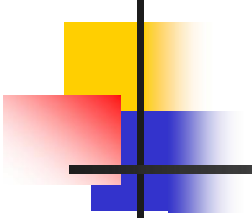
## 8.1-8.2 函数概述及定义的一般形式

把程序按功能划分为若干子程序模块，称为过程，**C**语言称之为函数。

main( )



函数调用过程：程序中若碰到一个函数调用语句（如**f1**（））时，便转到被调用函数**f1**（）执行该函数（程序段）内容。执行完后再返回原调用出口，继续执行主程序内容。



## 1) 函数的定义形式

---

类型标识符 函数名 (形式参数表列)

{ 声明部分

函数体\_\_语句序列

**return** 表达式;

//表达式值返回赋给函数名作为函数值

}

说明：形式参数表列可以为空，即“无参函数”；函数体也可以为空即“空函数”。

## 2) 函数调用

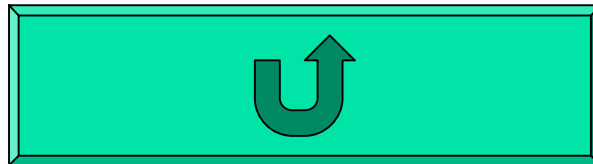
格式：函数名（实参表）

功能：语句中若遇到执行函数名，就转去调用该名的函数，用“实参”代替“形参”，然后返回函数值。

## 3) 函数声明

规律：先定义再调用的函数不再声明，调用后面定义的函数必须进行函数声明。

格式：类型 函数名（参数类型表）



## 8.3 函数参数和函数的值

### 例 8.2 调用函数时的数据传递。



```
//第8章 例题
//8.2 调用函数时的数据传递
#include <stdio.h>
int max(int x,int y) //定义有参数的函数max
{
    int z;
    z=x>y?x:y;
    return(z);
}
void main()
{
    int a,b,c;
    scanf("%d,%d",&a,&b);
    c=max(a,b);
    printf("Max is %d\n",c);
}
```

调用  
VC  
程序



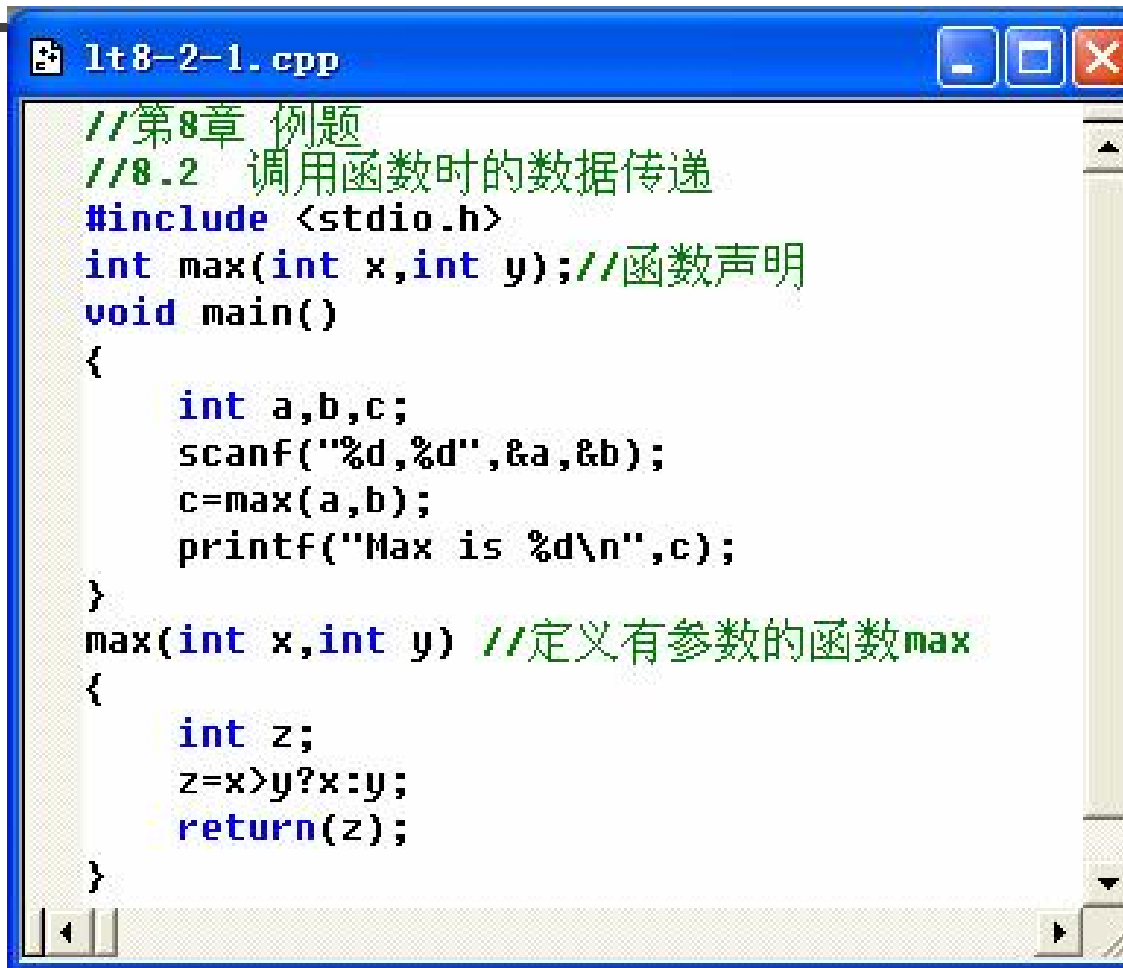
注意：函数调用时，其括号内的实际参数个数和类型应与函数建立时的形式参数相对应。

---

C语言的函数必须先建立，后调用。函数必须有返回类型，但可以是void型。

函数的建立可以分解为函数声明和函数定义两部分，函数声明在主函数前面，函数定义在后面（或其他地方）。

例 8.2 另一种形式，调用后面定义的函数，必须先作函数声明。



```
//第8章 例题
//8.2 调用函数时的数据传递
#include <stdio.h>
int max(int x,int y);//函数声明
void main()
{
    int a,b,c;
    scanf("%d,%d",&a,&b);
    c=max(a,b);
    printf("Max is %d\n",c);
}
max(int x,int y) //定义有参数的函数max
{
    int z;
    z=x>y?x:y;
    return(z);
}
```

调用VC程序

## 8.4 函数的调用

### 例 8.4

```
lt8-4.cpp
//第8章 例题
//8.4 函数调用的一般形式。
#include <stdio.h>
int f(int a,int b);//函数声明
void main()
{
    int i=2,p;
    p=f(i,++i); //函数调用,这时i值为3
    printf("%d\n",p);
}
int f(int a,int b)//函数定义
{
    //从主函数传来i值3分别给了a和b
    int c;
    if(a>b) c=1;
    else if(a==b) c=0;
    else c=-1;
    return(c); //c值为0
}
```

调用  
VC  
程序



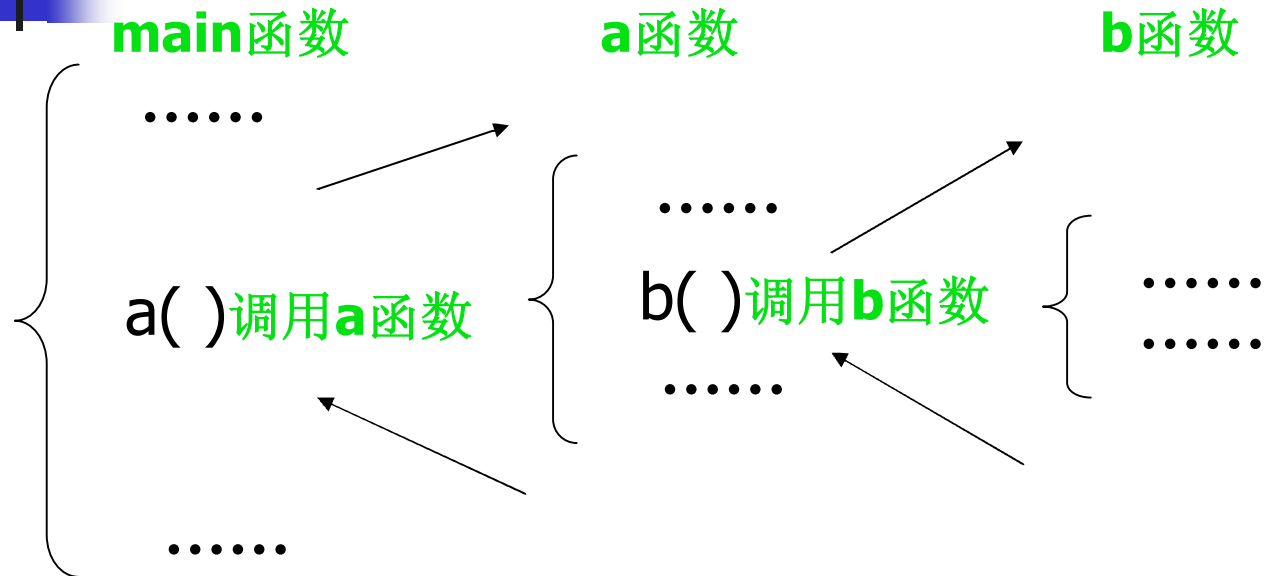
## 例 8.5 输入两个数，用函数求和。

```
lt8-5.cpp
//第8章 例题
//8.5 对被调用的函数作声明。
#include <stdio.h>
void main()
{
    float add(float x,float y); //对被调用的函数的声明
    float a,b,c;
    scanf("%f,%f",&a,&b);
    c=add(a,b);
    printf("sum is %f",c);
}
float add(float x,float y) //函数首部
{
    float z;
    z=x+y;
    return(z);
}
```

调用  
VC  
程序

## 8.5 函数的嵌套调用

概念：在调用一个函数的过程中又调用另一个函数，称为嵌套调用。见**P168-图8.5**。



说明：嵌套调用的执行过程见**P167-168 (1) - (9)**

## 8.6 递归函数

一个函数直接或间接地调用自身，这就是函数的递归调用，前者称为直接递归调用，后者称为间接递归调用。

例 8.8 递归计算n!的函数。

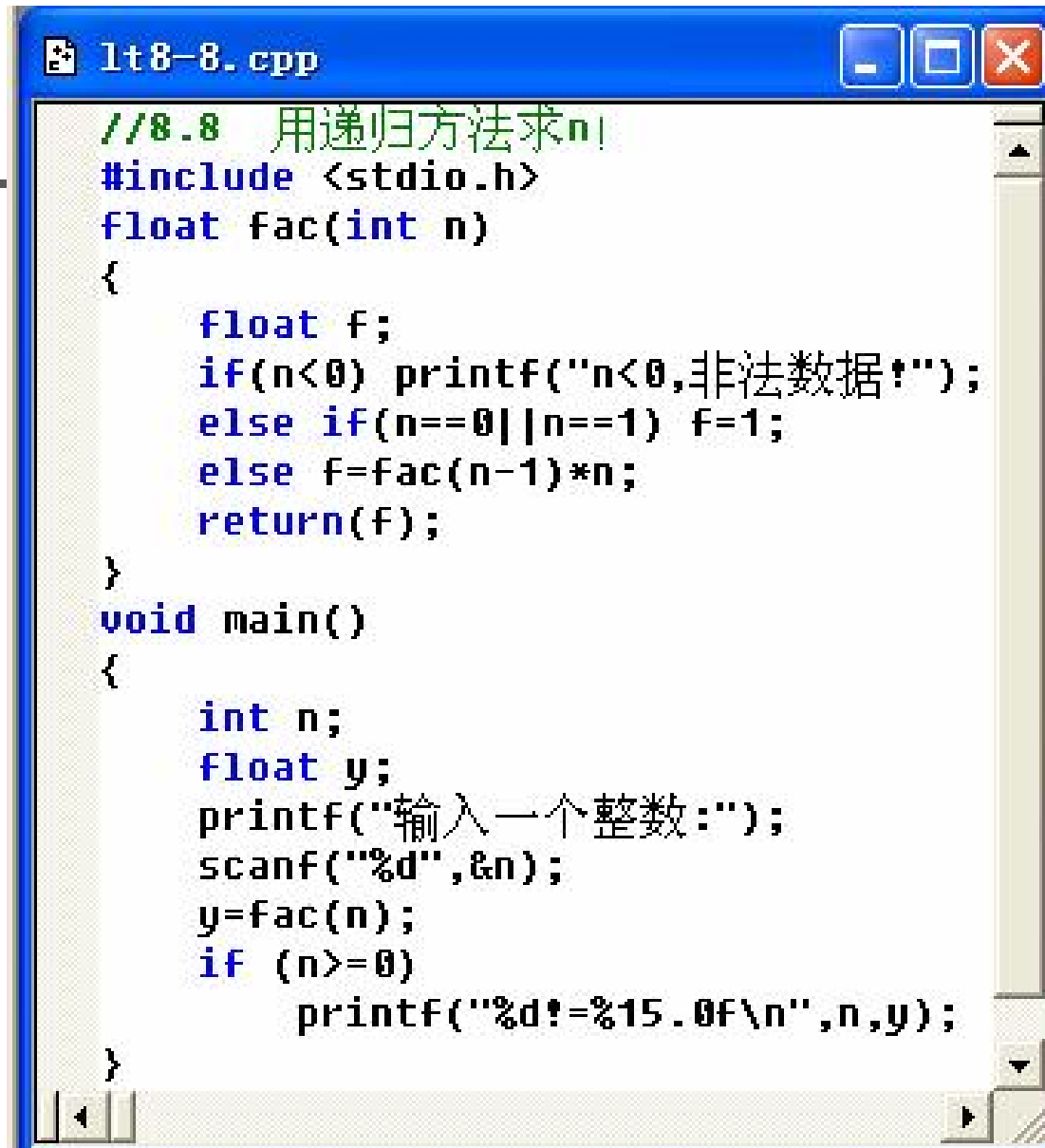
分析：

$$\text{fac}(n)=n! = \begin{cases} \text{非法} & n < 0 \\ 1 & n == 0, 1 \\ n * (n-1)! & n > 1 \end{cases}$$

如求  $3! == \text{fac}(3)$

$$3! == \text{fac}(3) \longleftarrow 3 * \text{fac}(2) \longleftarrow 3 * 2 * \text{fac}(1) \longleftarrow 3 * 2 * 1 == 6$$

## 例8.8 程序:



```
1t8-8.cpp
//8.8 用递归方法求n!
#include <stdio.h>
float fac(int n)
{
    float f;
    if(n<0) printf("n<0,非法数据!");
    else if(n==0||n==1) f=1;
    else f=fac(n-1)*n;
    return(f);
}
void main()
{
    int n;
    float y;
    printf("输入一个整数:");
    scanf("%d",&n);
    y=fac(n);
    if (n>=0)
        printf("%d!=%15.0F\n",n,y);
}
```

调用  
VC  
程序

## 8.7 数组作为函数参数

### 1、数组元素作函数实参

例 8.10 有两个数组**a**、**b**，各有**10**个元素，将它们对应地逐个相比（即**a[0]**与**b[0]**比，**a[1]**与**b[1]**比……）。如果**a**数组中的元素大于**b**数组中的相应元素的数目多于**b**数组中元素大于**a**数组中相应元素的数目（例如，**a[i]>b[i]**6次，**b[i]>a[i]**3次，其中*i*每次为不同的值），则认为**a**数组大于**b**数组，并分别统计出两个数组相应元素大于、等于、小于的次数。

```
lt8-10.cpp
//8.10 数组元素作为函数实参
#include <stdio.h>
void main()
{
    int large(int x,int y); //函数声明
    int a[10],b[10],i,n=0,m=0,k=0;
    printf("enter array a:\n");
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    printf("\n");
    printf("enter array b:\n");
    for(i=0;i<10;i++)
        scanf("%d",&b[i]);
    printf("\n");
    for(i=0;i<10;i++)
    {
        if(large(a[i],b[i])==1) n=n+1;
        else if(large(a[i],b[i])==0) m=m+1;
        else k=k+1;
    }
    printf("a[i]>b[i]%d times\na[i]=b[i]%d times \na[i]<b[i]%d times\n",n,m,k);
    if(n>k) printf("array a is larger than array b\n");
    else if(n<k) printf("array a is smaller than array b\n");
    else printf("array a is equal to array b\n");
}
large(int x,int y)
{
    int flag;
    if(x>y) flag=1;
    else if(x<y) flag=-1;
    else flag=0;
    return(flag);
}
```

调用VC程序



作业:

---

**P202 8.2, 8.3, 8.11**

## 8.7--2. 数组名可作函数参数

可以用数组名作函数参数，此时实参与形参都应用数组名。例  
8.11 有一个一维数组score,内放10个学生成绩，求平均成绩。

```
lt8-11.cpp
//8.11 有一个一维数组score,内放10个学生成绩,求平均成绩。
#include <stdio.h>
float average(float array[10])
{
    int i;
    float aver,sum=array[0];
    for(i=1;i<10;i++)
        sum=sum+array[i];
    aver=sum/10;
    return(aver);
}
void main()
{
    float score[10],aver;
    int i;
    printf("input 10 scores:\n");
    for(i=0;i<10;i++)
        scanf("%f",&score[i]);
    printf("\n");
    aver=average(score); //传递的是score的首地址
    printf("average score is %5.2f",aver);
}
```

调用  
VC  
程序



## 2. 数组名可作函数参数

```
lt8-13.cpp
//8.13 用选择法对数组中的10个整数由小到大排序
#include <stdio.h>
void sort(int array[],int n)
{
    //数组array作实参
    int i,j,k,t;
    for(i=0;i<n-1;i++)
    {
        k=i;
        for(j=i+1;j<n;j++)
            if(array[j]<array[k]) k=j;
        t=array[k];array[k]=array[i];array[i]=t;
    } //每一轮(第i轮)都将最小的数换到第i个位置
} //k表示最小的那个数的位置,其它数若比它小则与之交换
void main()
{
    int a[10],i;
    printf("enter the array\n");
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    sort(a,10); //数组a作形参
    printf("the sorted array:\n");
    for(i=0;i<10;i++)
        printf("%d ",a[i]);
    printf("\n");
}
```

例 8.13 用选择法对数组中10个整数按由小到大排序。过程分析见P181。

调用  
VC  
程序

### 3. 用多维数组名作函数参数

例 8.14 有一个 $3 \times 4$ 的矩阵（即3行4列数），求所有元素中的最大值。

```
lt8-14.cpp
//8.14
#include <stdio.h>
max_value(int array[][4])
{
    int i,j,max;
    max=array[0][0];
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            if(array[i][j]>max) max=array[i][j];
    return(max);
}
void main()
{
    int a[3][4]={{1,3,5,7},{2,4,6,8},{15,17,34,12}};
    printf("max value is %d\n",max_value(a));
}
```

调用  
VC  
程序

## 8.8 局部变量和全局变量

在文件作用域中声明的变量称为**全局变量**，该变量在整个文件中有效；在块作用域中声明的变量称为**局部变量**，该变量在此块内有效。（分析P184-185表中的变量范围）

### 例 8.16

调用VC程序

```
lt8-16.cpp
//8.16 外部变量和局部变量同名。
#include <stdio.h>
int a=3,b=5;      //a,b为外部变量
max(int a, int b) //a,b为局部变量
{
    int c;
    c=a>b?a:b;
    return(c);
}
void main()
{
    int a=8;      // a为局部变量
    printf("%d",max(a,b));
}
}
```

## 8.9 变量的存储类别

**静态储存方式：**在程序运行期间分配固定的存储空间的方式，全局变量全部存放在静态储存区。

**动态存储方式：**在程序运行期间根据需要进行动态的分配存储空间方式，如局部变量等。

**auto 变量：**动态存储变量，没有声明为auto变量的都隐含指定为auto变量。

静态局部变量：在局部变量前面加上static，其作用是在函数调用结束后，静态局部变量不会消失。但只是在声明它的函数嵌套块内有效，不是在程序中的每个函数都有效。

### 例 8.17

调用VC程序

分析：若略去static，  
程序输出将会如何？

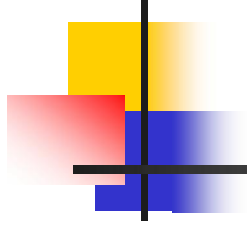
```
lt8-17.cpp
//8.17 考察静态局部变量的值
#include <stdio.h>
f(int a)
{
    auto b=0;
    static c=3;
    b=b+1;
    c=c+1;
    return(a+b+c);
}
void main()
{
    int a=2,i;
    for(i=0;i<3;i++)
        printf("%d ",f(a));
}
```

**register**变量：寄存器变量，将变量值存放在计算机寄存器中。（该情况很少用！）

用**extern**声明外部变量，例 8.20

```
lt8-20.cpp
//8.20 用extern声明外部变量，扩展程序文件中的作用域。
#include <stdio.h>
int max(int x,int y) //定义max函数
{
    int z;
    z=x>y?x:y;
    return(z);
}
void main()
{
    extern A,B; //外部变量声明
    printf("%d ",max(A,B));
}
int A=13,B=-8; //定义外部变量
```

调用  
VC  
程序

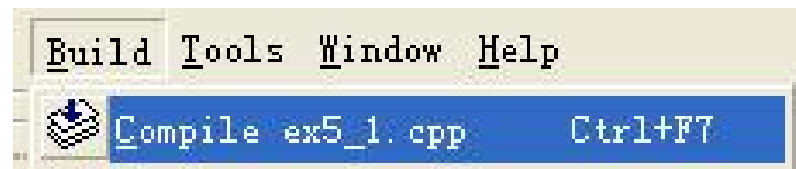


**存储类别小结，见P197-198。**

多文件组成的工程文件与运行要点。

多文件组成的工程文件，其中只有一个源文件具有主函数`main()`，而其他文件不能含有`main()`，否则程序不知道该从何处开始执行。

运行多文件组成的工程文件时，首先在**VC**状态下将所有文件打开编辑好，然后把含有主函数的文件作为**当前文件**，执行菜单**build**（连接）中的**compile**（编译）命令。



注意：若把不含主函数的文件作为当前文件进行连接编译，会出错。



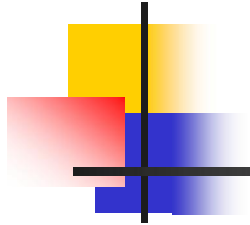
## 例 8.22 多个文件同时运行P199-200

```
lt8-22-1.cpp
//8.22 外部函数(文件1)
#include <stdio.h>
#include"lt8-22-2.cpp"
#include"lt8-22-3.cpp"
#include"lt8-22-4.cpp"
void main()
{
    extern enter_string(char str[80]);
    extern delete_string(char str[],char ch);
    extern print_string(char str[]);
    //以上3行声明在本函数中将要调用的
    //在其他文件中定义的3个函数
    char c;
    char str[80];
    enter_string(str);
    scanf("%c",&c);
    delete_string(str,c);
    print_string(str);
}

lt8-22-2.cpp
//8.22 外部函数
//(文件2)
#include<stdio.h>
enter_string(char str[80])
//定义外部函数enter_string
{gets(str);}
//读入字符串str

lt8-22-3.cpp
//(文件3)
delete_string(char str[],char ch)
{//定义外部函数delete_string
    int i,j;
    for(i=j=0;str[i]!='\0';i++)
        if(str[i]!=ch)
            str[j++]=str[i];
    str[j]='\0';
}

lt8-22-4.cpp
//(文件4)
print_string(char str[])
{//定义外部函数print_string
    printf("%s",str);
}
```



# 作业： P202-习题 8.8, 8.9

上机通过多个文件同时运行的例 8.22