

程序设计基础

(C Programming)

北京航空航天大学计算机学院

软件工程研究所

晏海华

程序设计基础 (C Programming)

第一讲：C语言基础(一)



本章目标

- 了解C语言的历史及现状
- 初步了解C程序的结构
- 知道在某个操作系统环境下(Windows)一个C程序的编写过程；
- 掌握变量、常量、简单数据类型、类型转换、表达式及部分C语言运算符；
- 知道运算符优先级及结合律
- 掌握选择、循环等控制结构
- 掌握程序测试和调试方法



程序设计与程序设计语言

- **程序设计(Programming):** 为计算机解决问题所需的分析、设计、编写及调试程序过程。（The process of planning, writing, testing, and correcting the steps required for a computer to solve a problem or perform an operation.）
- **程序设计语言(Programming Language):** 用来表达程序的计算机能够执行的人工语言，是程序设计过程中用到的一种工具。如，Fortran, C, C++, Java, C# 等。

可以这样理解两者之间的关系：
程序设计语言是我们解决问题过程（程序设计）中用到的工具之一（可能还有其它工具，如分析与设计工具。



为什么要学C语言程序设计

- C语言是一种简洁精练的语言，所涉及的概念比较少，语法和结构也简单，主要特点：
 - 表达能力强，支持结构化程序设计；
 - 语言简洁；
 - 代码效率高：C编写的程序仅比用汇编语言编写的程序相差20%；
 - 可移植性好；
 - 特别适合编写操作系统、编译程序、数据库系统、嵌入式软件及图形/图象处理等对性能要求高的软件；
- C语言仍是目前广泛使用的编程语言，其应用领域遍及系统软件、应用软件、数值计算、嵌入式软件等
- C语言是目前广泛流行的面向对象语言C++、C#及Java的基础

C语言历史

■ C语言的产生与UNIX操作系统是密不可分的：

- UNIX由Bell Lab的K. Thompson和D. M. Ritchie最先在1969年开发的O. S.（它的前身是MIT和AE开发的Multics）。
- 1970年，V1，V2版在PDP-7机上用汇编语言实现
- 1971年V3 PDP11/23；1972年V4 PDP11/45
- 1972年，D. M. Ritchie开发出新语言C。（C ← B ← BCPL ← CPL单数据型语言）
- 1973年，Ritchie和Thompson用C改写了UNIX核心（90%）即V5

C语言历史（续）

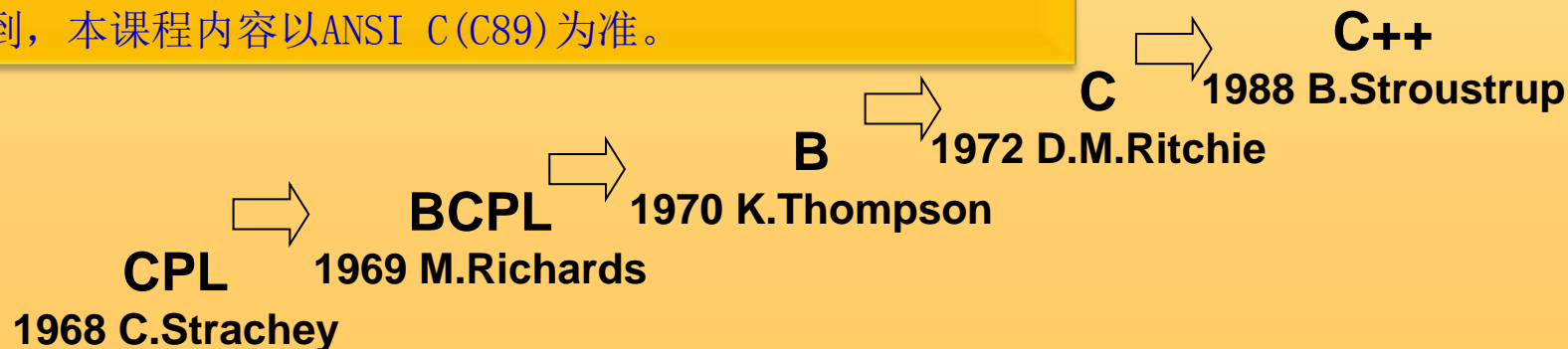
■ C语言的发展经历三个阶段：

- 第一阶段：78年以前，C称为UNIX C，即C被看成UNIX的一部分。
- 第二阶段：78年D. Ritchie的《C程序设计语言》出版到88年ANSI C（标准C）标准出现。此C又称为K&R C。
- 第三阶段：88年ANSI C标准（89年3月批准，又称C89）。

1999年：ANSI和ISO联合提出了ISO/IEC9899:1999(又称C99)

2011年：ISO提出了ISO/IEC9899:2011(又称C11)

由于C99和C11支持的编译器较少，同时C99和C11新增特性初学者使用不到，本课程内容以ANSI C(C89)为准。



一个简单的C程序：在屏幕上显示一行正文

[例1-1]

```
/* file: hello.c */
```

```
#include <stdio.h>
```

```
int main( )
```

```
{
```

```
    printf("hello, world\n");
```

```
    return 0;
```

```
}
```

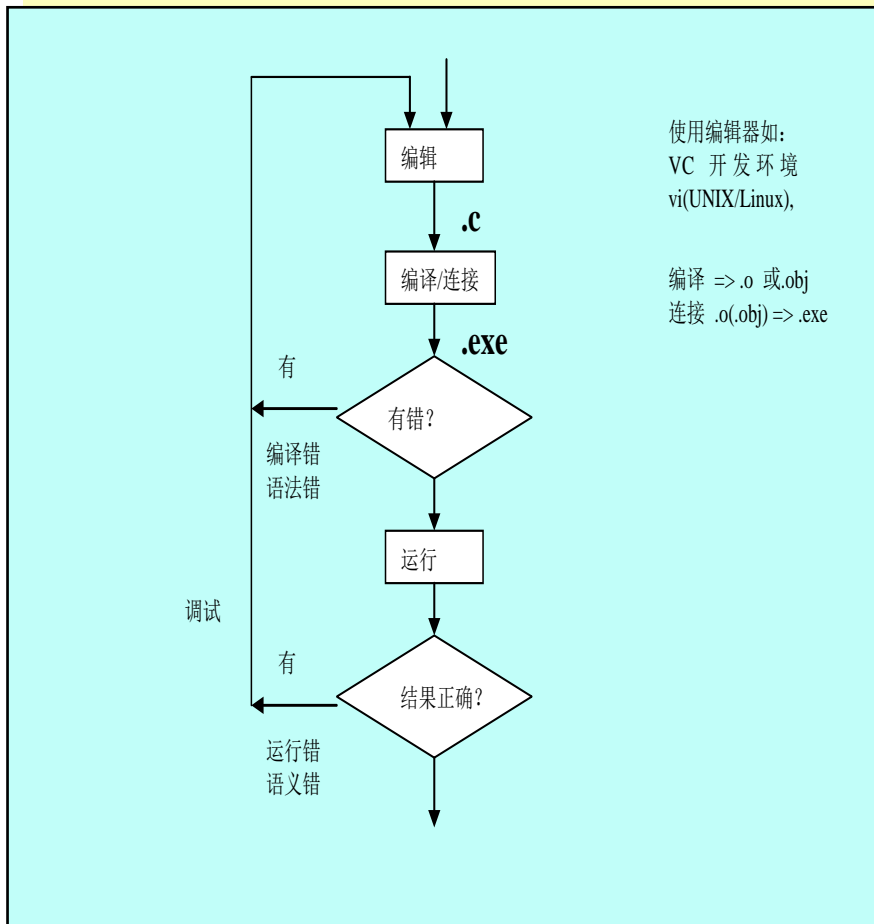
/* ... */ 为注释(*comment*), 不可嵌套

#include为一条预处理指令, 当程序中用到输入/输出函数时, 应在文件开始处加上该指令。

- main为一主函数(*function*)名。
- 由{ }括起来的部分为函数体。
- 函数名为一标识符(*identifier*)。
- int 为main函数的类型, 表明其返回一个整数值

- printf为一条输出语句(*statement*), 在C语言中分号(;)为语句的结束符。
- printf为标准I/O库中标准输出函数。
- “...”为一字符串常量(*constant*)。
- \n为C语言转义字符, 表示回车。
- return 0语句将函数返回值设为0。

C程序的创建过程（编辑、编译及运行）



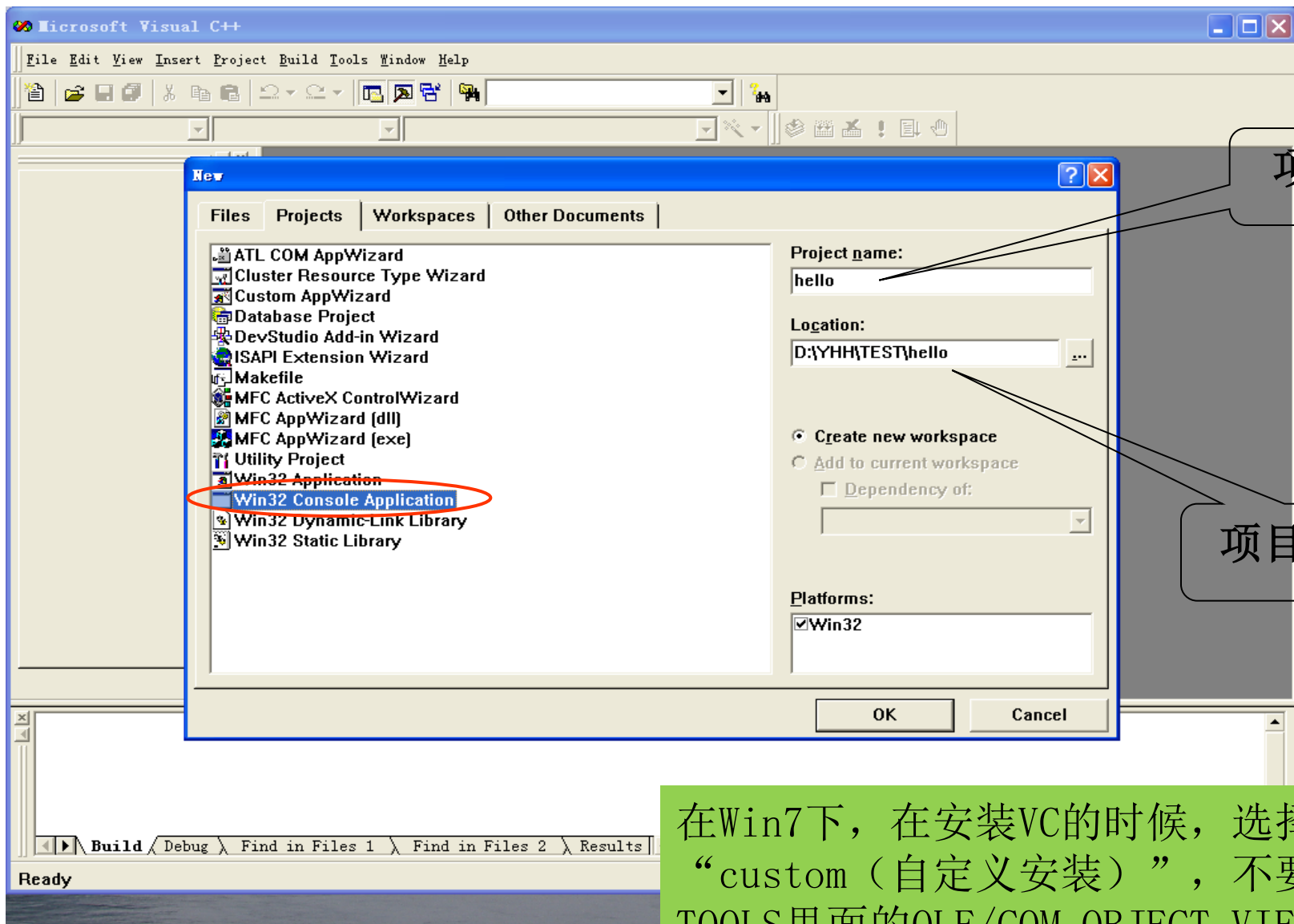
1. 使用编辑器（如VC IDE，或vi）创建一个包含源代码的.c文件；
2. 使用编译器（如VD IDE, 或gcc）对该源文件进行**编译**，若没有语法错误，将生成一个执行文件（.exe）
3. **运行**该执行文件，提供相应输入数据，若没有语义错误，将得到正确结果。（在Windows下可通过双击该执行文件来运行该文件，也可通过VC IDE来直接运行该文件）
4. 若在编译或运行阶段程序出错，则可根据错误信息（如编译信息或运行结果）找到源程序中**错误**(bug)并**修改**(fix)后，重新编译运行程序。（该过程又称**调试**，**debuging**）

语法(编译)错误—syntax(compiler)
error

语义(运行)错误—semantics(runtime)
error

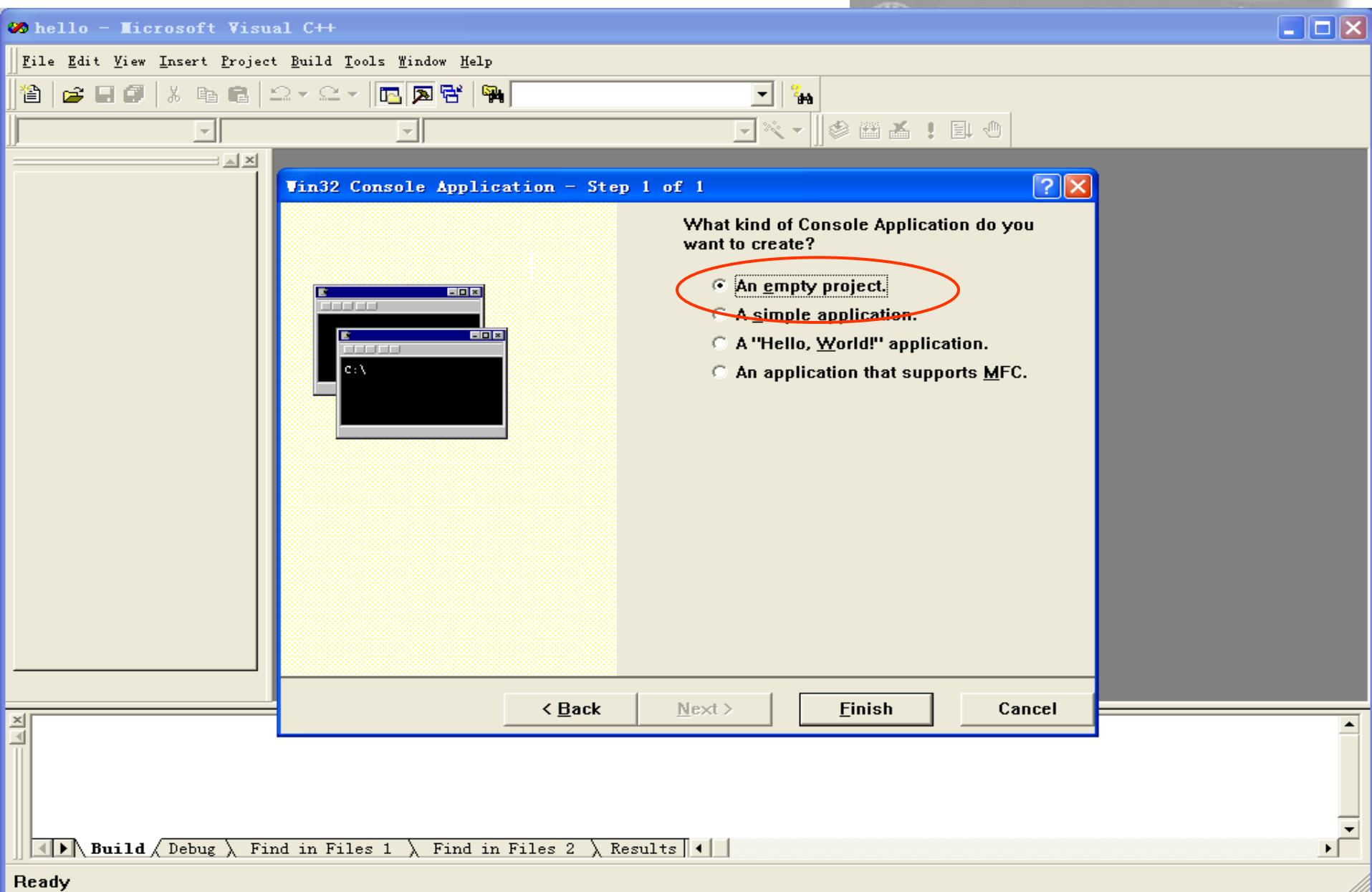
学会调试程序的方法是一个好程序员必备的素质！

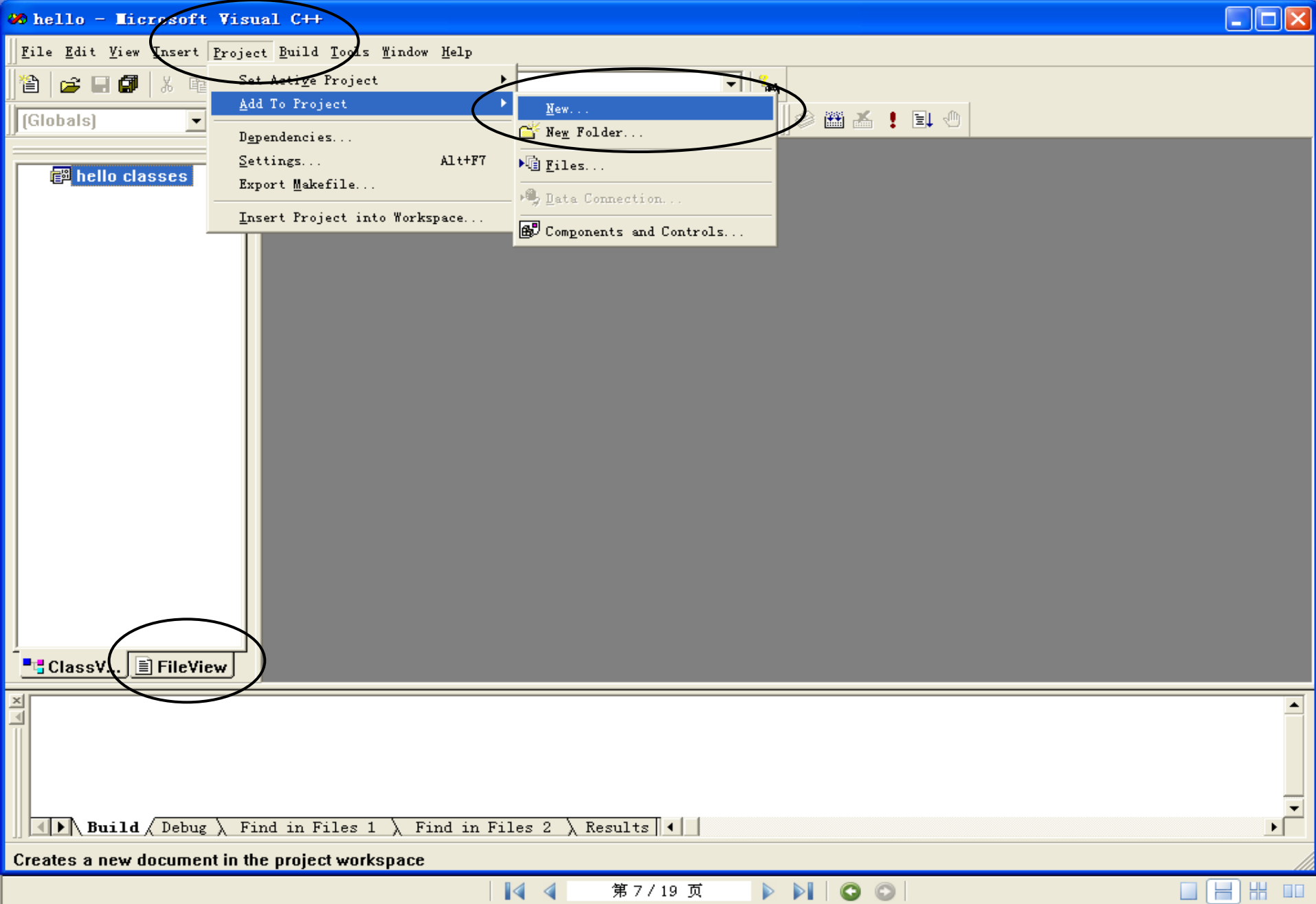
在Windows下使用VC6.0编写及运行C程序

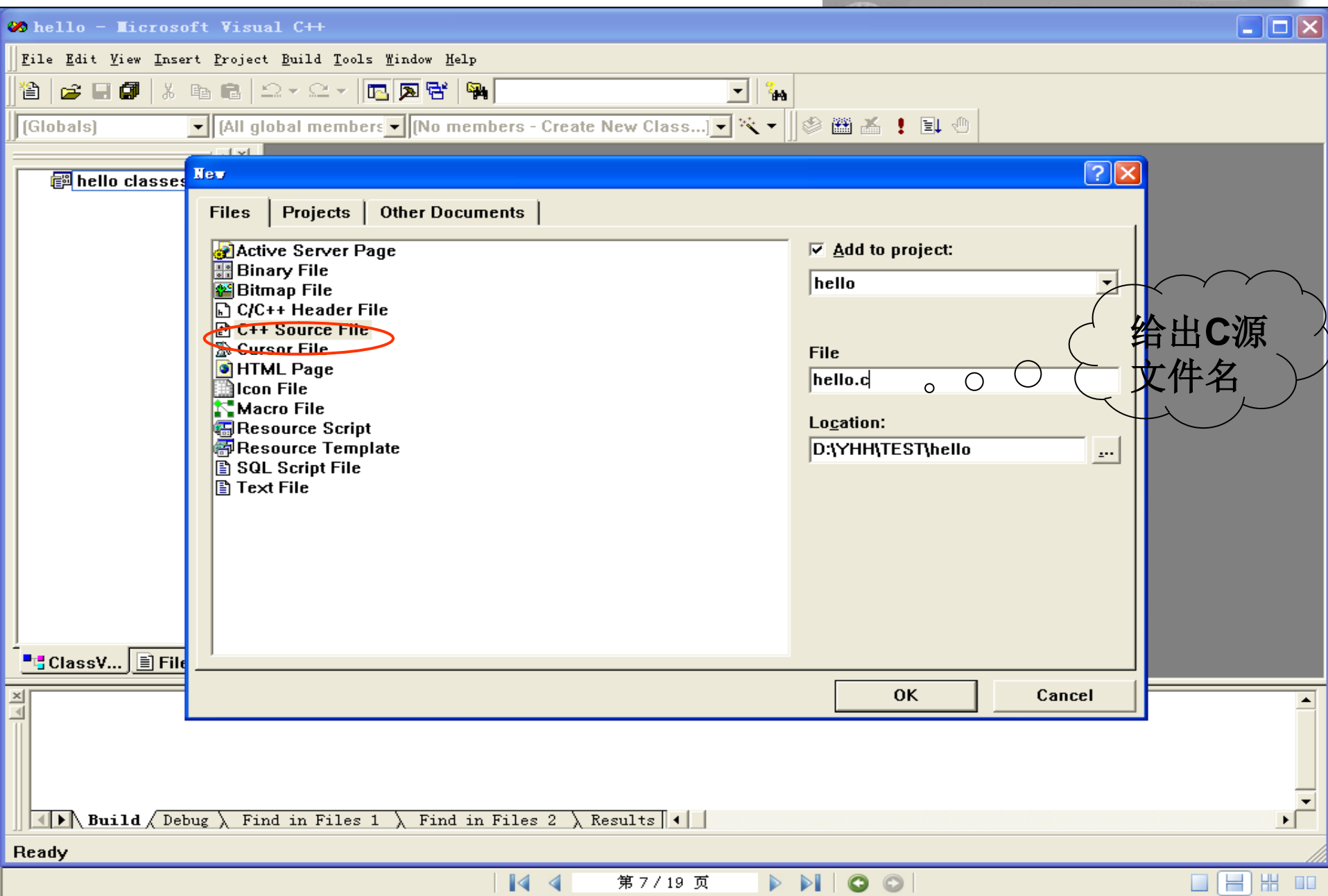


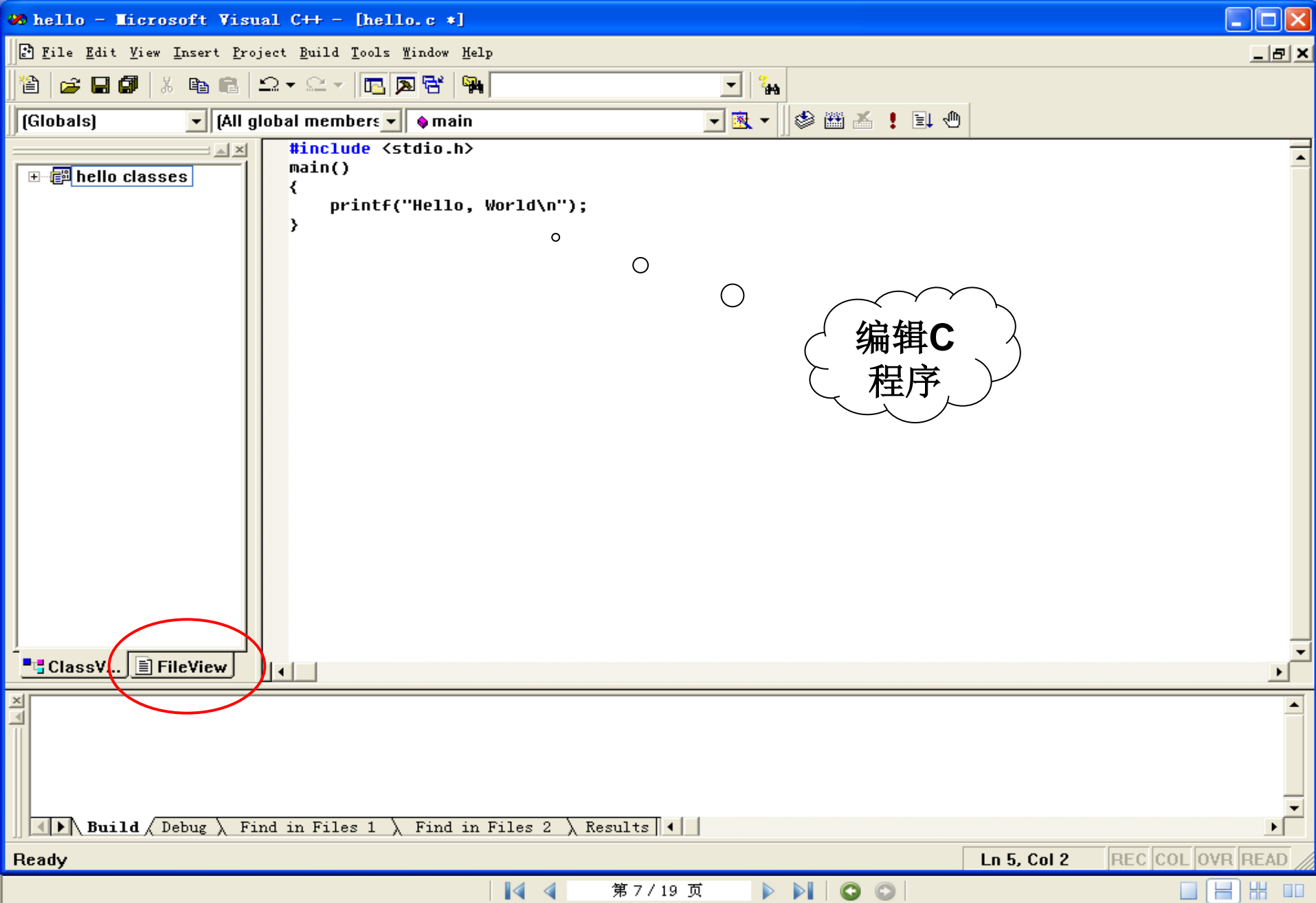
C程序设计

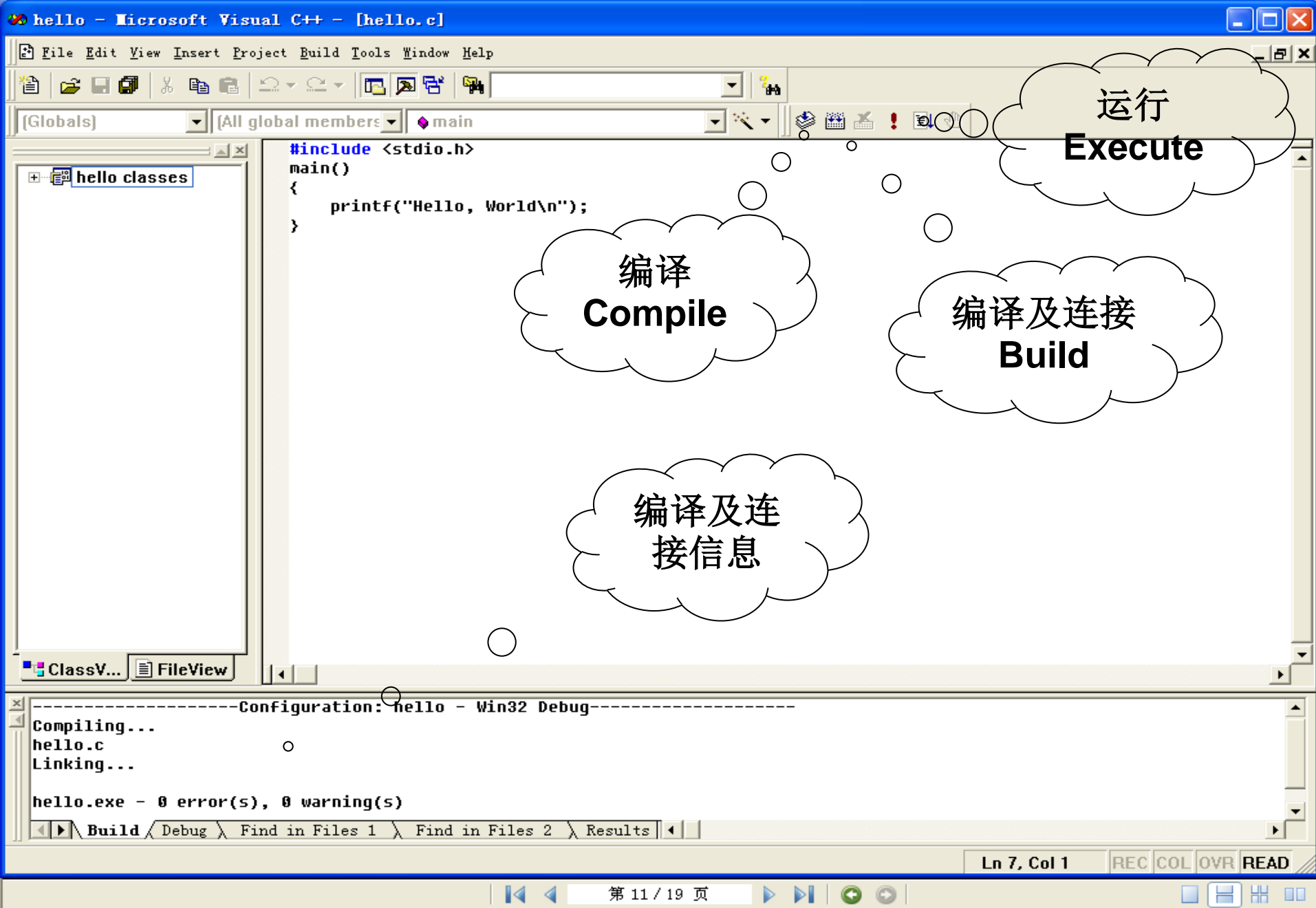
在Win7下，在安装VC的时候，选择“custom（自定义安装）”，不要选择TOOLS里面的OLE/COM OBJECT VIEWER工具，即可。

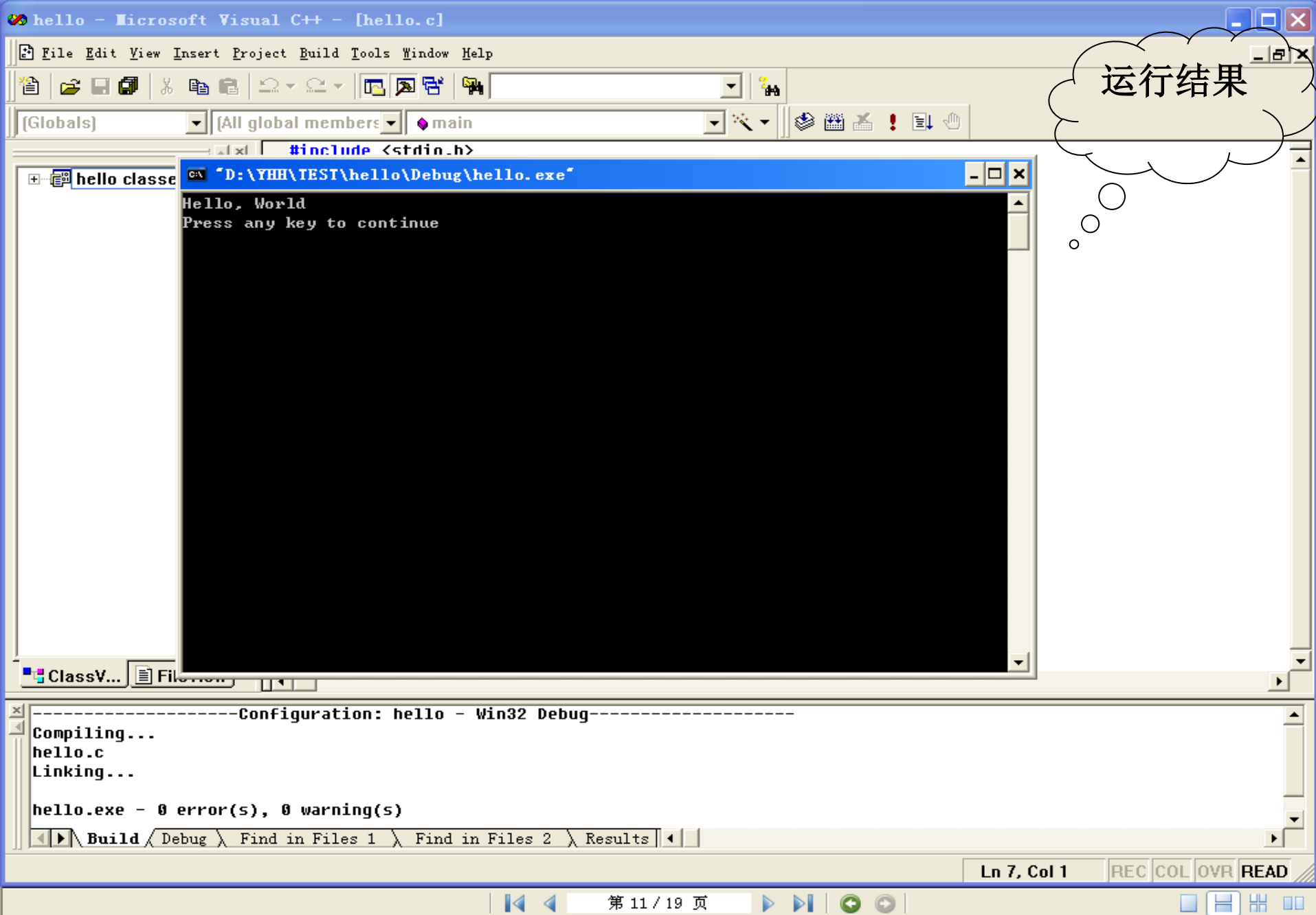














Visual Studio 2010中文版

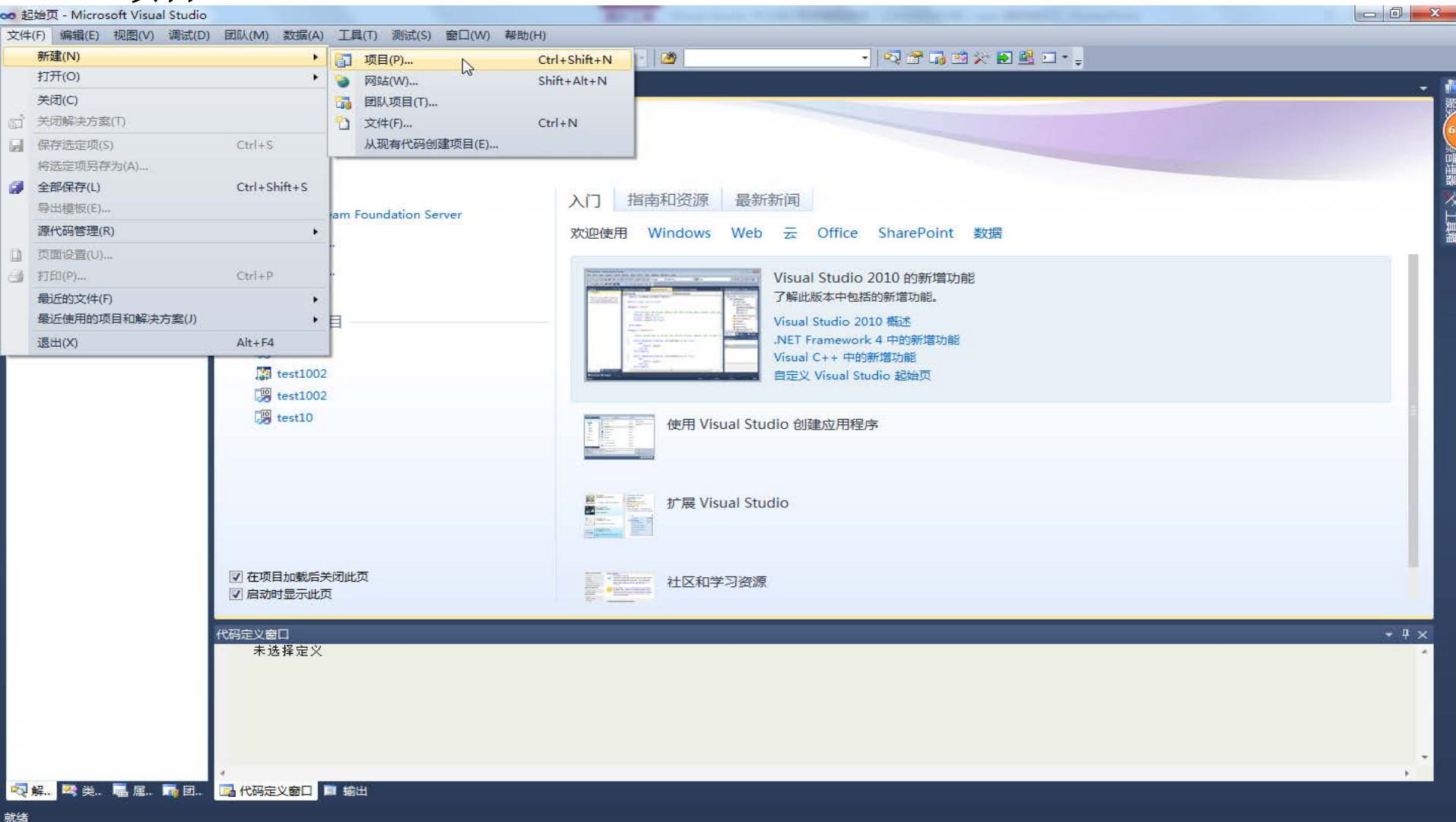
C程序编程指南

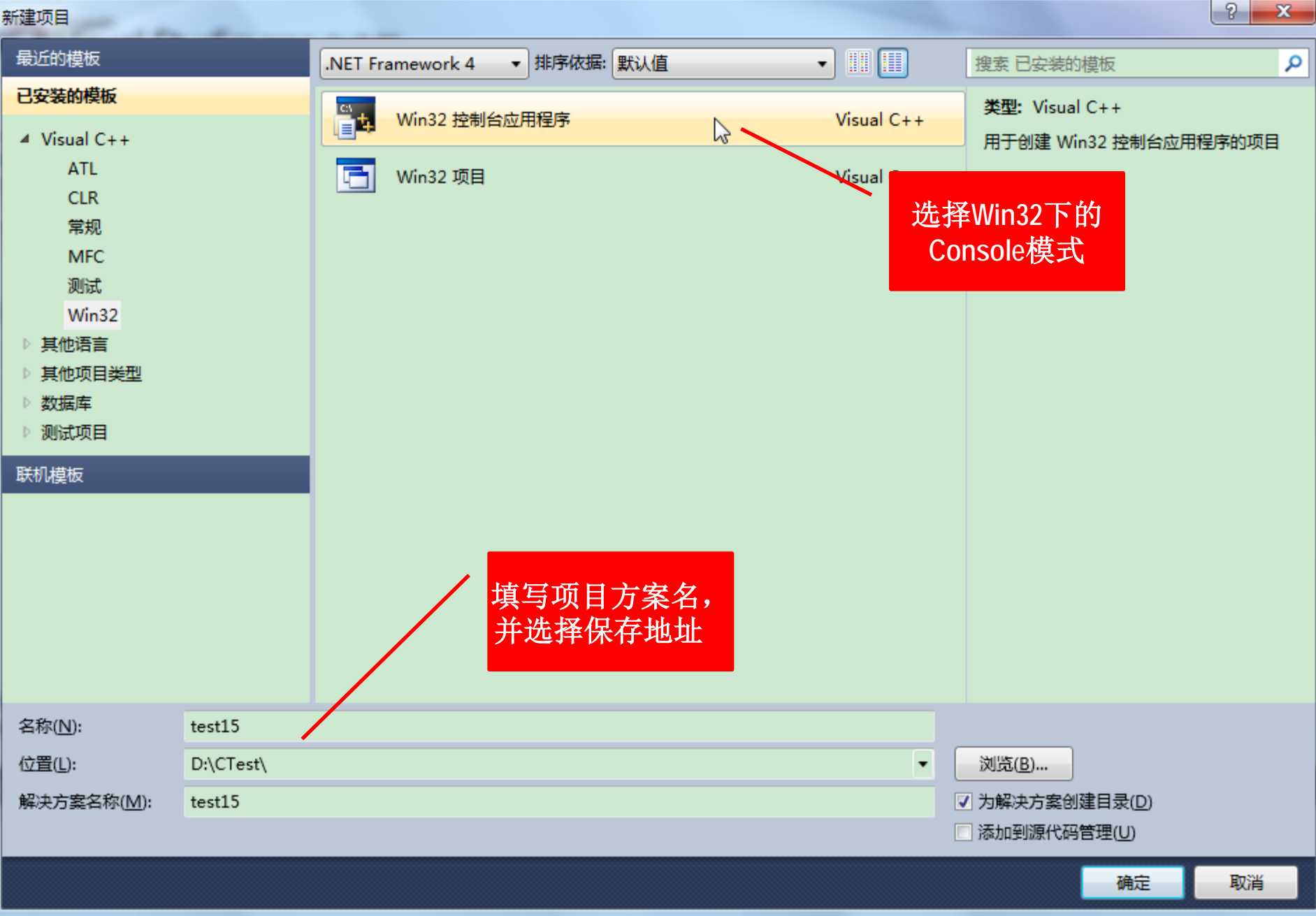


首次运行Visual Studio 2010时，请选择：

- Visual C ++ 开发设置

创建C程序项目：选择主菜单“文件”->“新建”->“项目”





新建项目

最近的模板

已安装的模板

- Visual C++
 - ATL
 - CLR
 - 常规
 - MFC
 - 测试
 - Win32
- 其他语言
- 其他项目类型
- 数据库
- 测试项目

联机模板

.NET Framework 4 排序依据: 默认值

搜索 已安装的模板

- Win32 控制台应用程序 Visual C++
- Win32 项目 Visual

类型: Visual C++
用于创建 Win32 控制台应用程序的项目

选择Win32下的 Console模式

填写项目方案名, 并选择保存地址

名称(N): test15

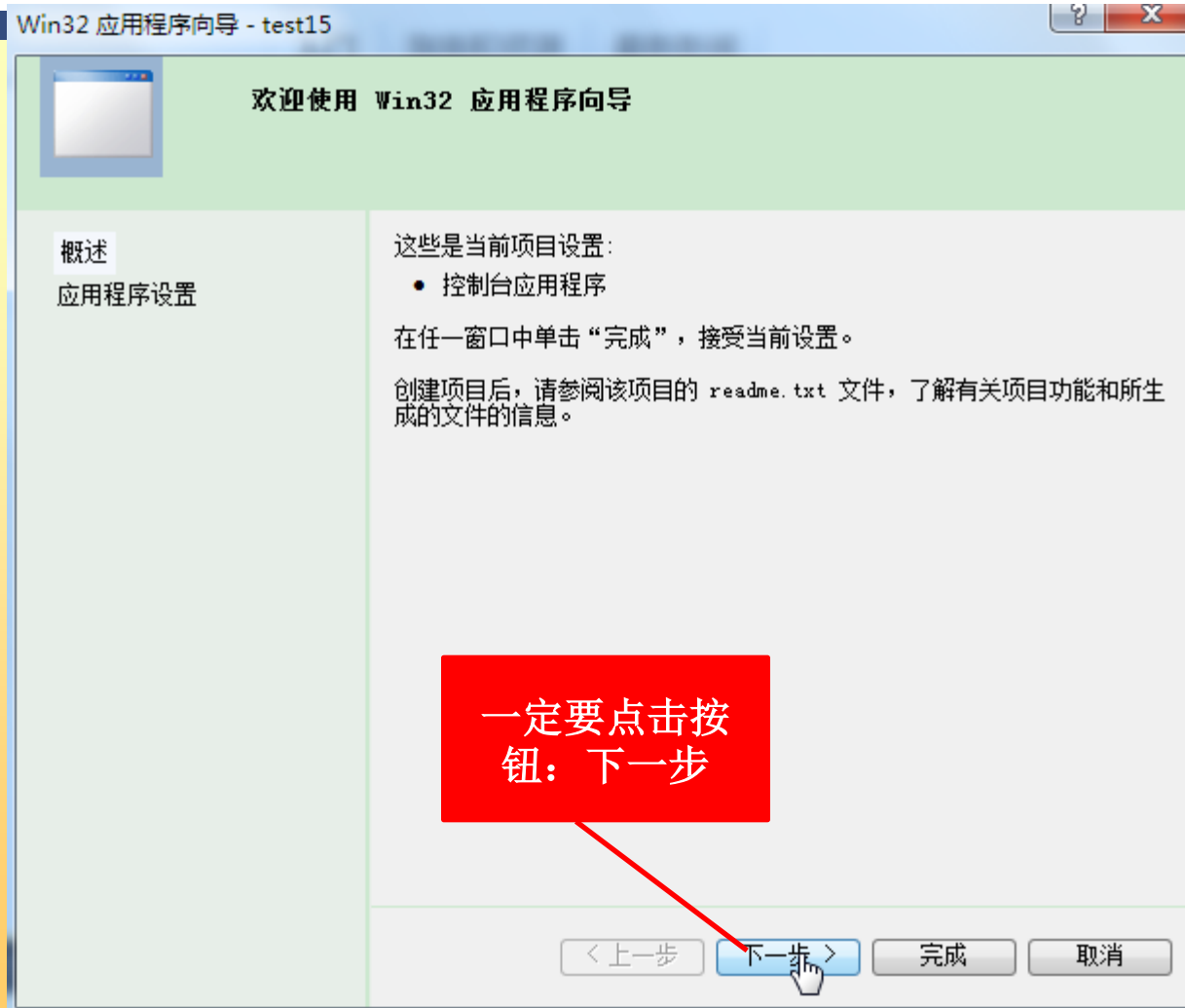
位置(L): D:\CTest\

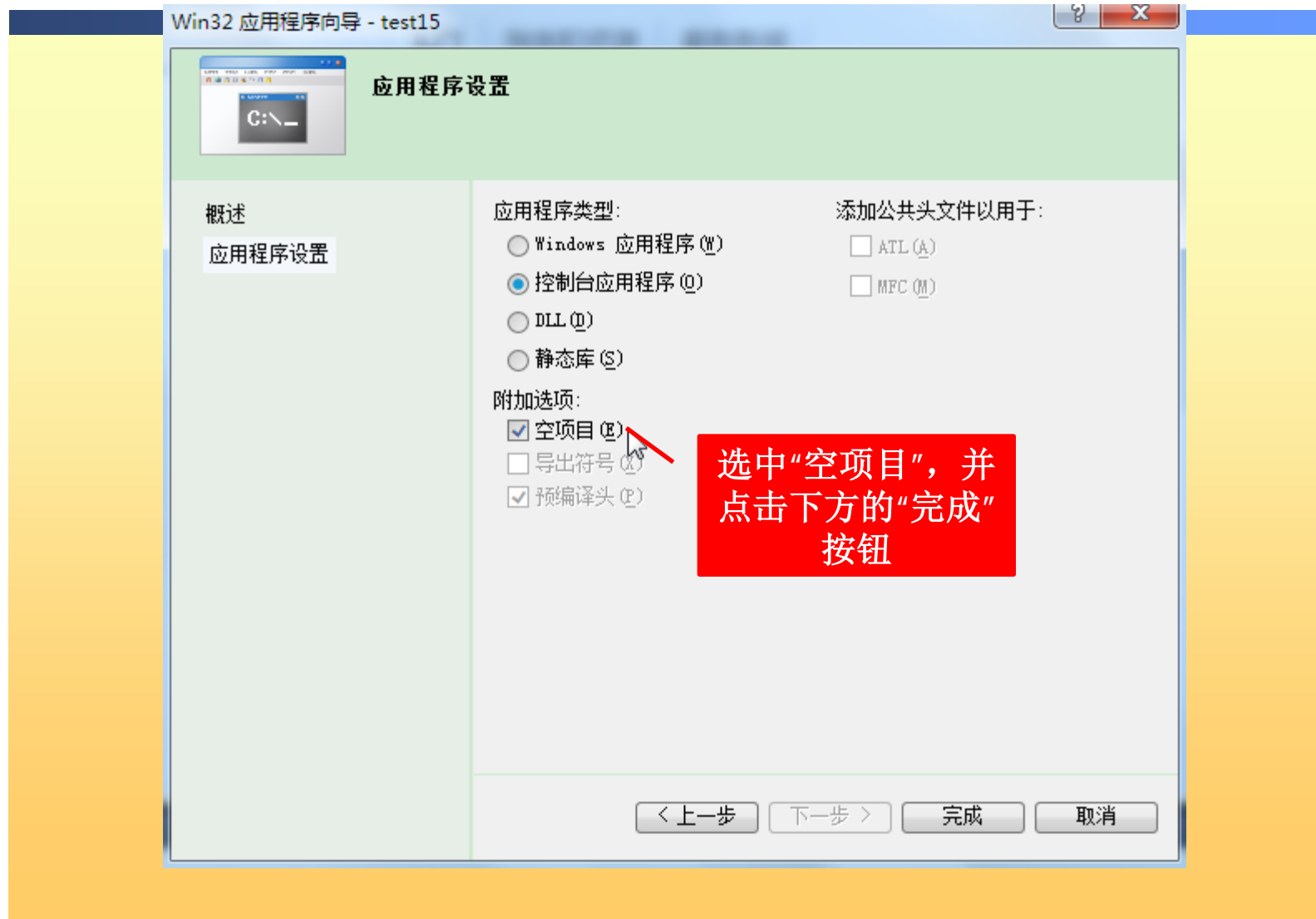
解决方案名称(M): test15

浏览(B)...

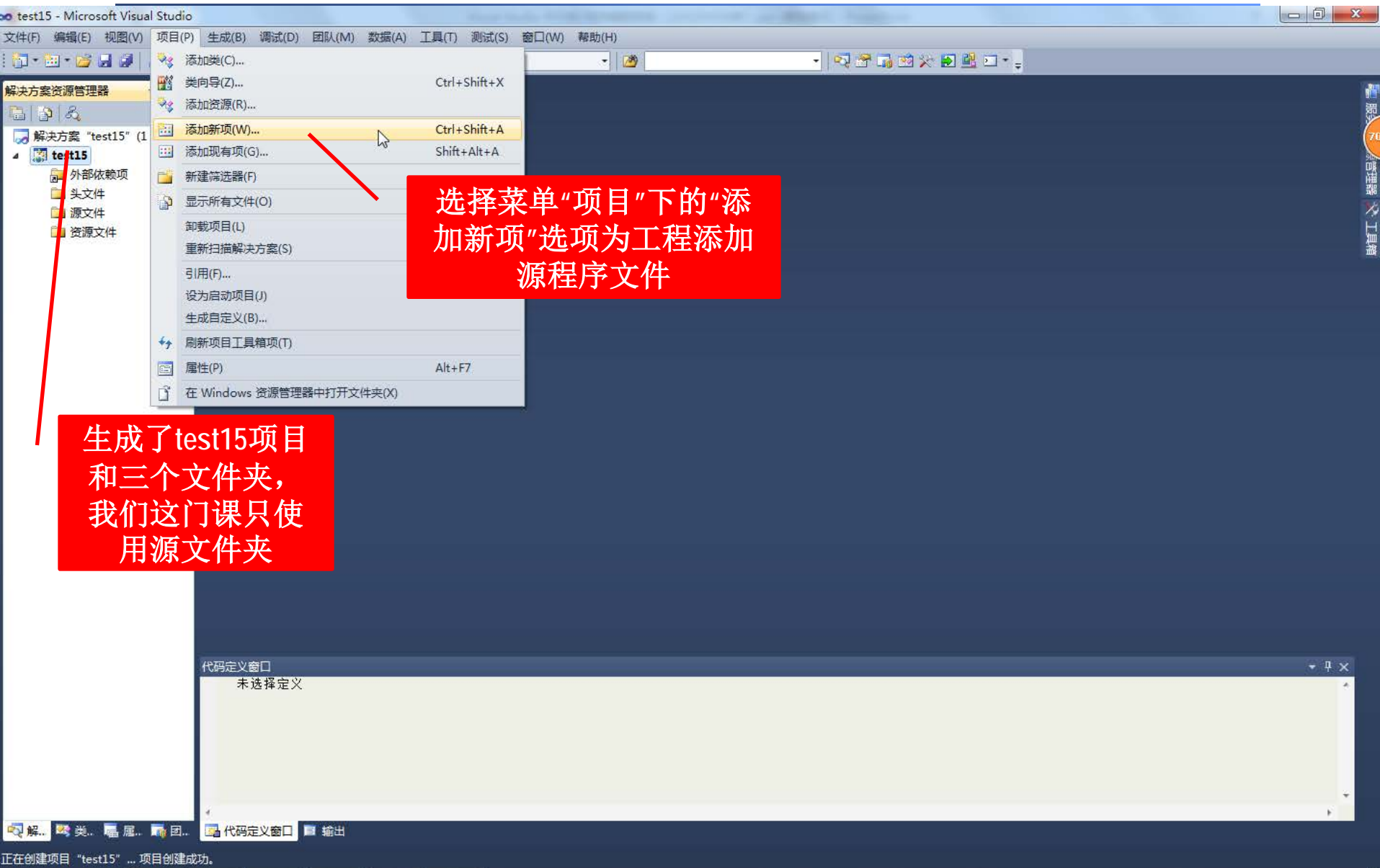
- ☒ 为解决方案创建目录(D)
- ☐ 添加到源代码管理(U)

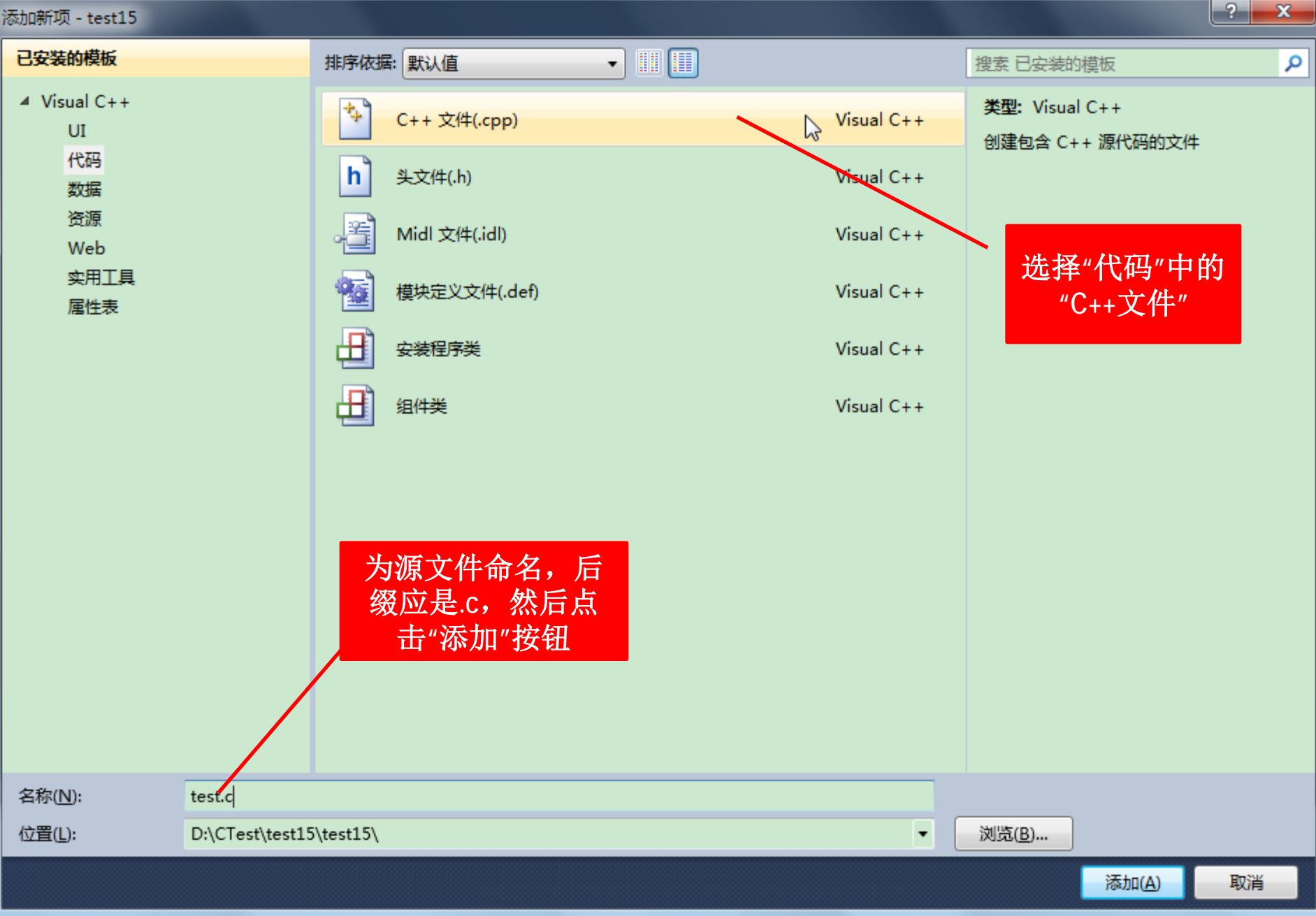
确定 取消

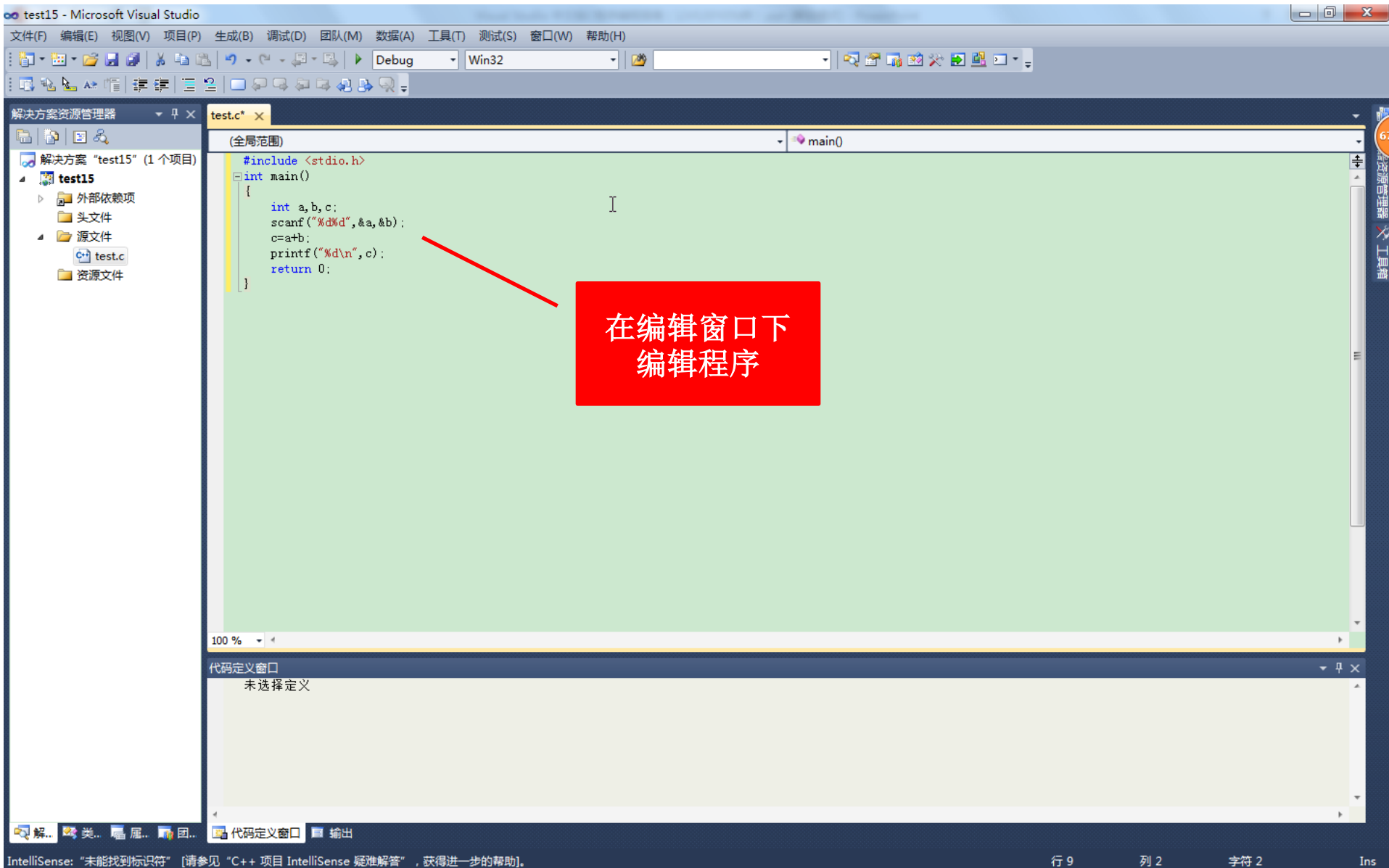




生成一空的项目，往项目中添加程序文件：







The screenshot shows the Microsoft Visual Studio interface. The 'Build' menu is open, highlighting the 'Build' option (Ctrl+F7). A red arrow points from the 'Build' menu item to a red text box. The Output window at the bottom shows the build process, with a red arrow pointing from the 'Build' menu item to the 'Build' output line. The status bar at the bottom indicates '生成成功' (Build Succeeded).

生成(B) 调试(D) 团队(M) 数据(A) 工具(T) 测试(S) 窗口(W) 帮助(H)

生成解决方案(B) F7
重新生成解决方案(R) Ctrl+Alt+F7
清理解决方案(C)

生成 test15 (U)
重新生成 test15 (E)
清理 test15(N)
仅用于项目(J)
按配置优化(P)
批生成(T)...
配置管理器(O)...

编译(M) Ctrl+F7

点击“生成”菜单下的“编译”进行编译，并点击“生成(工程名)”进行链接创建可执行程序

输出

显示输出来源(S): 生成

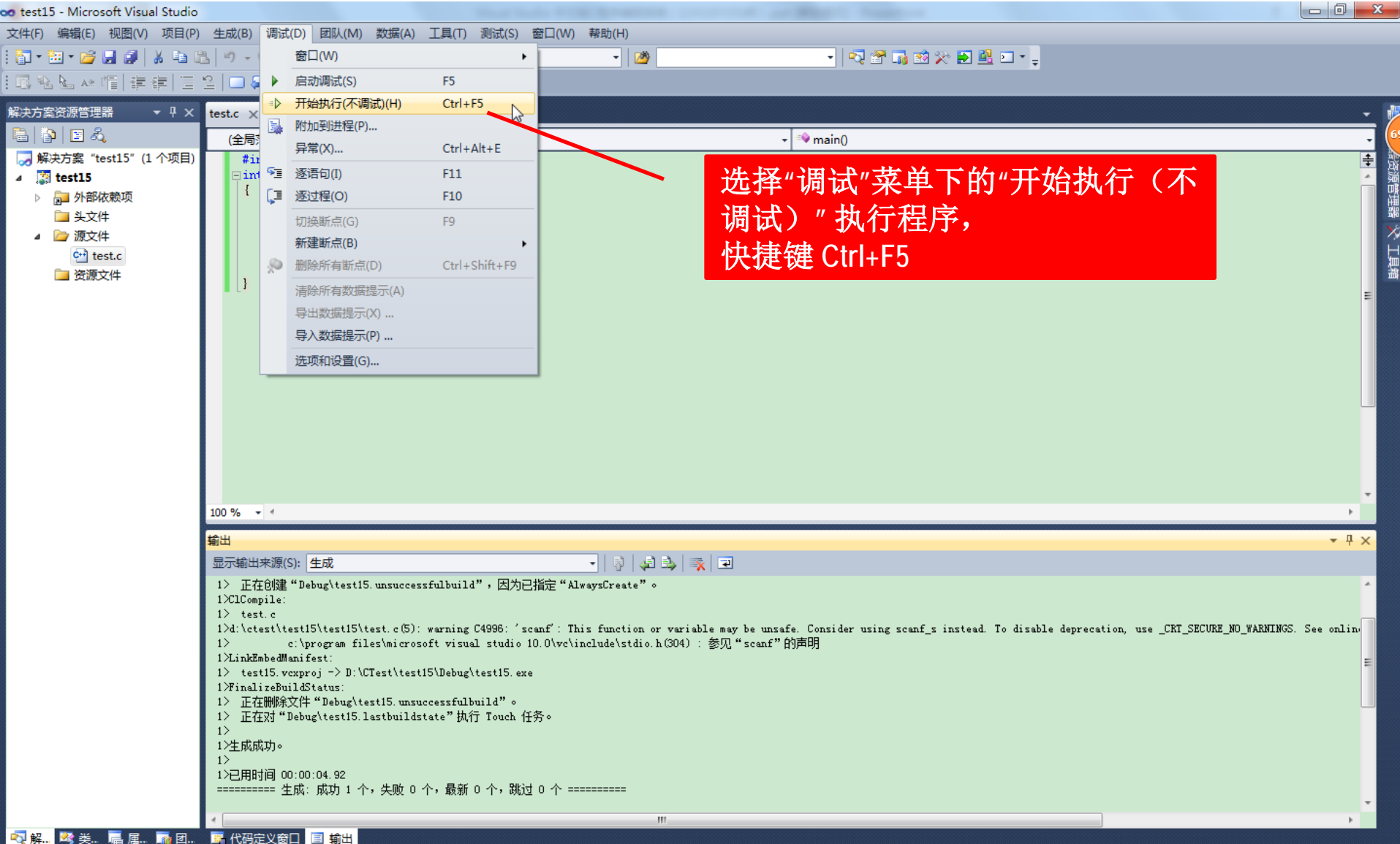
```
1> 正在创建 "Debug\test15.unsuccessfulbuild"，因为已指定 "AlwaysCreate"。
1> ClCompile:
1> d:\ctest\test15\test15\test.c(5): warning C4996: 'scanf': This function or variable may be unsafe. Consider using scanf_s instead. To disable this warning, use the /SCN4996 compiler option. See online documentation for details.
1> c:\program files\microsoft visual studio 10.0\vc\include\stdio.h(304): 参见 "scanf" 的声明
1> LinkEmbedManifest:
1> test15.vcxproj -> D:\CTest\test15\Debug\test15.exe
1> FinalizeBuildStatus:
1> 正在删除文件 "Debug\test15.unsuccessfulbuild"。
1> 正在对 "Debug\test15.lastbuildstate" 执行 Touch 任务。
1>
1> 生成成功。
1>
1> 已用时间 00:00:04.92
===== 生成: 成功 1 个, 失败 0 个, 最新 0 个, 跳过 0 个 =====
```

编译信息

生成成功

行 20 列 71 字符 54 Ins

晏海华



```
#include <stdio.h>
int main()
{
    int a,b,c;
    scanf ("%d%d",&a,&b);
    c=a+b;
    printf ("%d\n",c);
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
2 3
5
请按任意键继续. . .
```

执行结果显示窗口：
控制台窗口



另一个简单的C程序：整数求和

[例1-2]

```
#include <stdio.h>
```

```
int main( )
```

```
{    /* c1_2.c */
```

```
    int a, b, c, sum;
```

```
    a = 1;
```

```
    b = 2;
```

```
    scanf("%d", &c); /*注意不能省略&*/
```

```
    sum = a + b + c;
```

```
    printf("Sum = %d\n", sum);
```

```
    return 0;
```

```
}
```

- **int** 为整数(*integer*)数据类型(*type*)说明符，其为一个关键字(*keyword*)。a,b,c,sum为变量(*variable*)，其为标识符。
- **a = 1;** 为赋值(*assignment*)语句，其中1为数字常量，=为赋值运算符。
- **scanf**为标准输入函数，在此从键盘上读入一个整数存入变量c中，%d表示读入一个整数。
- 在此，**printf**将变量sum内容显示到屏幕上。
- **+**为算术运算符(*operator*)。
- **return**返回一个值给调用者。通常程序正常结束返回一个0值。

另一个简单的C程序*： 计算公式 $x = \sqrt{a^2 + b^2}$

[例1-3]

```
#include <stdio.h>
```

```
#include <math.h>
```

/* 使用数学函数时应加上该预处理指令*/

```
int main()
```

```
{
```

```
    double a, b, x;
```

/* **double**为双精度浮点类型，即一种实数类型 */

```
    scanf("%lf %lf", &a, &b);
```

/* 两个%lf表示从键盘输入2个double类型的实数，
分别存放到变量a和b中*/

```
    x = sqrt(a*a + b*b);
```

/* *为乘法运算符，sqrt为计算平方根数学函数*/

```
    printf("x = %f\n", x);
```

/* %f表示按浮点数（实数）形式输出一个浮点数*/

```
    return 0;
```

```
}
```



简单的C程序结构

```
预处理指令      /* 如 #include <stdio.h> */  
int main( )      /* 函数头 */  
{               /* 函数体开始 */  
    变量说明部分 /* 如 int a, b, c, sum; */  
    执行语句部分 /* 如 sum = a + b + c; */  
    return 0;     /* 函数返回一个值 */  
}               /* 函数体结束 */
```

C程序结构

- 一个C程序由一系列外部说明和函数组成；
- 一个函数则由局部变量说明及语句序列组成；
- 一个C程序可由一个或多个函数组成，但其中必有一个（也只能有一个）命名为main（主函数），其它函数可由用户任取名字。程序运行时必需从main开始，但main函数在程序中的前后位置没有关系；
- 组成C程序的各个函数可在一个源文件上，也可以分放在多个文件上（函数不能跨文件），每个源文件可单独编译。C源文件必须以.c作后缀（.h为C程序的头文件）；



标识符

- 在C语言中**标识符** (*identifier*) 定义为：“由字母（或_）开头的字母（_）数字串”。标识符在C语言中可作为**变量名**、**常量名**、**函数名**、**参数** (*parameter*)名、**类型名**、**枚举** (*enumeration*)名和**标号** (*label*)等。

注意：C语言对字母大小写敏感, 因此,

name, Name, NAME 是**3个不同的标识符**。



标识符（续）

■ 标识符满足以下规则:

- 所有标识符必须由一个字母 (a~z, A~Z) 或下划线 () 开头;
- 标识符的其它部分可以用字母、下划线或数字 (0~9) 组成;
- 大小写字母表示不同意义, 即代表不同的标识符;
- 标识符通常只有前32个字符有效 (依赖系统实现);
- 用户定义的标识符不能是C语言的关键字。



判断下面哪些是非法的标识符？

Beijing

C_Programming

a8673

~~8a673~~

_1024

~~sin(25)~~

num

num_of_stdudent

~~a[1]~~

关键字

关键字 (keyword): 已经被系统使用的标识符, 具有预先定义好的特殊意义, 不能作为变量名、函数名及标号等使用。

int

float

char

short

long

unsigned

double

struct

union

void

enum

typedef

sizeof

const

signed

auto

static

extern

register

goto

return

break

continue

if

else

while

for

do

switch

case

default

define, undef, include, ifdef, ifndef, endif, 及line, 虽不是关键字, 但是最好把它们看作关键字, 它们主要用于C预处理程序中。



基本数据类型

数据类型		一般 32 位操作系统 (如 UNIX, W32)	16 位操作系统 (如 DOS)
int	整型	32	16
short(int)	短整型	16	16
long(int)	长整型	32	32
unsigned(int)	无符号整型	32	16
float	单精度浮点型	32	32
double	双精度浮点型	64	64
char	字符型	8	8

注意：在C语言中，没有Bool（布尔）类型，它用非0值表示真（True），用0表示假（False）。

说明：

1. 在C语言中，数据是有范围的；如对于32位int其数值范围为 $-2,147,483,648 \sim 2,147,483,647$ ($-2^{32-1} \sim 2^{32-1}-1$)

2. 在不同的系统平台，同一数据类型（如int）范围可能不同。

一般来说，不同的系统（如Windows, Linux），相同数据类型长度可能有差异。但有个原则是：短整型(short)不能长于普通整型(int)；长整型(long)不能短于普通整型(int)。

变量与变量说明

- 在C中，所有变量必须先说明（定义）后使用；
- 说明方式：

[存贮类] 类型 变量（列）表；

例：

```
int lower, upper, step;  
char c, line[100];  
extern double x;  
double PI = 3.1415926;  
char msg[ ] = "Warning" ;
```

[例1-2]

```
#include <stdio.h>  
int main( )  
{   /* c1_2.c */  
    int a, b, c, sum;  
    a = 1; b = 2;  
    scanf("%d", &c);  
    sum = a + b + c;  
    printf("Sum = %d\n", sum);  
    return 0;  
}
```

变量属性

变量的类型

变量的名字

double salary = 1500.23;

变量的存储位置（地址）
例如地址：0x00222000

变量的初值

常量(*constant*)

常量：就是在程序中不可以被程序改变的数值。

- 整型常量（十进制、八进制、十六进制、long整型常量），
如：

十进制	八进制	十六进制	long整型常量
1275	0127	0x19a, 0xABD	123l, 89L

- 浮点常量（缺省为double类型）

如：3.14159, -8E-3, -125e+4

常量（续）

标准ASCII字符集有如下特点：

1. 其值范围为0~127（是一个小范围的正整数）；
2. 相邻字符之间值相差为1（如' a' 和' b' 之间），因此'a'+3就得到字符d的编码值
3. 相应大小写字母间相差一个固定值（可用' a' -' A' 表示）

- 字符常量，用一对单引号括起来字符称为字符常量，如：

'A', 'b', '?' ...

一个字符常量的值是该字符在机器字符集（通常是ASCII字符集，但某些IBM大型机用EBCDIC码）中的编码值，它是一个整数值。在ASCII字符集中 'A'的值为65， 'b'的值为98， ' ? ' 的值为63。

在C语言中，还有一类特殊的字符常量，称为**转义字符常量**，如：

'\0', '\n', '\t', ...

'\ddd'(位模式，ddd为八进制数，其值为ddd)

字符常量可像其它整数一样参与数值运算，主要用于同其它字符作比较，如：

```
if( c >= 'A' && c <= 'Z')
```

```
    c = c + 'a' - 'A';
```

换行符	\n	水平制表符	\t	反斜线符	\\
退格符	\b	垂直制表符	\v	换页符	\f
响铃符	\a	回车符	\r	问号	\?
单引号	\'	双引号	\"		

ASCII编码表

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
0	NUL	32	(space)	64	@	96	,
1		33	!	65	A	97	a
2		34	"	66	B	98	b
3		35	#	67	C	99	c
4		36	\$	68	D	100	d
5		37	%	69	E	101	e
6		38	&	70	F	102	f
7		39	,	71	G	103	g
8		40	(72	H	104	h
9		41)	73	I	105	i
10		42	*	74	J	106	j
11		43	+	75	K	107	k
12		44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14		46	.	78	N	110	n
15		47	/	79	O	111	o
16		48	0	80	P	112	p
17		49	1	81	Q	113	q
18		50	2	82	R	114	r
19		51	3	83	X	115	s
20		52	4	84	T	116	t
21		53	5	85	U	117	u
22		54	6	86	V	118	v
23		55	7	87	W	119	w
24		56	8	88	X	120	x
25		57	9	89	Y	121	y
26		58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28		60	<	92	/	124	
29		61	=	93]	125	}
30		62	>	94	^	126	~
31		63	?	95	_	127	DEL

常量（续）

■ **字符串常量**，用一对双引号括起来的字符串称为字符串常量，如：

“The C Programming Language”

注意：所有字符串均以 ‘\0’ 结束（编码值为0的字符），因此，“x”和 ‘x’ 不同，末尾的 ‘\0’ 由编译程序自动添加。

字符串中可以有转义字符：

```
printf("Welcome \nto \nBeijing!\n");
```

输出：

```
Welcome  
to  
Beijing!
```



例：计算圆的面积及周长

[例1-4]

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    double radius, area, perimeter;
```

```
    scanf("%lf", &radius);
```

```
    area = 3.14159 * radius * radius;
```

```
    perimeter = 2 * radius * 3.14159;
```

```
    printf("Radius = %6.2f Area = %6.2f Perimeter = %6.2f",  
           radius, area, perimeter);
```

```
    return 0;
```

```
}
```

双精度浮点
数据

读入一个双
精度浮点数

常量



常量定义

- 所有常量可以用#define来定义，即可以给一个常量命名（符号常量）。如：

```
#define PI 3.14156
```

- 使用常量定义的好处：

- 可提高程序的可读性
- 程序的可移植性更好，可维护性更好

```
#include <stdio.h>
#define PI 3.14159
int main()
{
    double radius, area, perimeter;
    scanf("%lf", &radius);
    area = PI * radius * radius;
    perimeter = 2 * radius * PI;
    printf("Radius = %6.2 Area = %6.2\n",
        radius, area, perimeter);
    return 0;
}
```

标准输入/输出

- 标准输入（stdin）：通常指键盘
- 标准输出（stdout）：通常指屏幕显示器

在Windows等图形界面的操作系统中，输入、输出信息一般被显示在一个窗口中，在VC的Console编程模式下，显示标准输入、输出信息的窗口为控制台窗口。

标准输入函数：scanf

scanf (格式控制串, 其他参数1, ..., 其他参数n) ;

格式控制串是一字符串, 其中的转换控制字符 (以%开始) 决定了其他参数(输入数据)的个数和输入格式; 其它字符要必须原样输入。

转换控制字符	输入形式	对应参数类型
%d	十进制整数形式	int
%c	一个字符	char
%s	一个字符串 (以空格分隔)	字符数组或字符指针
%f	小数形式	float
%lf	小数形式	double

标准输出函数：printf

printf (格式控制串, 其他参数1, ..., 其他参数n) ;

格式控制串是一字符串, 其中的转换控制字符 (以%开始) 决定了其他参数的~~个数~~和~~输出格式~~; 其它字符原样输出。

转换控制字符	实现的转换	对应参数类型
%d	按十进制整数形式输出	int
%c	输出一个字符	char
%s	输出一个字符串 (以'\0'结束)	字符数组或字符指针
%f	按带小数点形式输出	浮点型
%e	按科学计数法形式输出	浮点型

在控制字符前还可以加数字, 如:

%4d : 输出最小域宽为4个字符的整数。


%6.2f : 输出最小域宽为6个字符的浮点数, 并且小数点占两位。



标准输入/输出：常见问题

■ 如何输出多个数据？

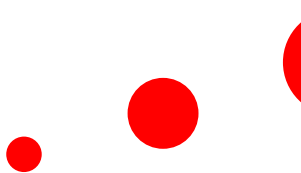
```
int x=2;  
int y=3;  
printf ( "%d + %d = %d\n ", x, y, x+y );
```



输出结果: **2+3=5**

■ 如何输入多个数据？

```
int n;  
float r;  
scanf( "%d%f", &n, &r );
```



输入多个数据时，用空格或回车换行分隔，但在格式字符串中不用给出空格或回车换行符。

标准输入/输出：常见问题

必须原样输入
非转换字符

```
float r;
```

```
scanf( "r=%f",&r );
```

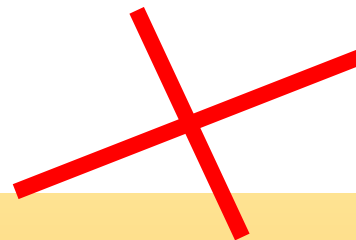
一个输入例子：

r=3.14

(3.14将被读入到变量 r 中)

```
double r;
```

```
scanf( "%lf",&r );
```





标准输入/输出：常见问题(续)

- 输入格式控制串中含有空白字符（如空格、\n或\t）

例：

```
int a,b;
```

```
scanf("%d %d\n",&a,&b);
```

What happened?

回车换行符后无法读入数据！！
建议最后不要有\n。



标准输入/输出：常见问题(续)

- 输出域宽和实际输出长度

输出格式: **%6.2f**

输出数据: **3.14159**

屏幕显示

		3	.	1	4
--	--	---	---	---	---

域宽为六个字符，输出数据若小于输出域宽，则右对齐输出，其它部分填充格；若输出数据长度多于给定域宽，则按实际数据输出。

类型转换(type conversion)



摄氏温度转华氏温度的公式:

```
int f, c;  
f=1.8*c+32;
```

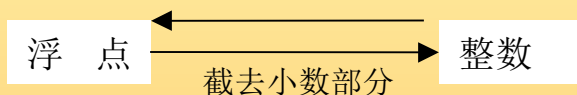
■ C语言类型转换通常是自动的--- 隐式(自动)类型转换

1) 字符与整数

可以用整数的地方就可以用字符。

而整数转换成字符时，超出8位就将高位丢掉。

2) 浮点数与整数



3) 无符号整数

普通整数 (int) 和无符号整数 (unsigned) 混合使用，则普通整数转换成无符号整数。

类型转换（续）

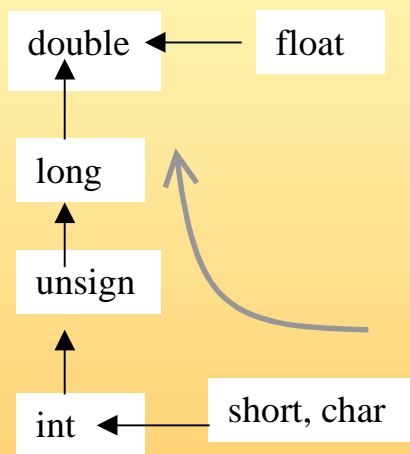


摄氏温度转华氏温度的公式：

```
int f, c;  
f=1.8*c+32;
```

4) 算术转换

如果一个运算符，有不同类型的运算对象，那么“较低”类型会自动转换成“较高”类型。



如， $n+1.5$ 结果将为double类型

5) 赋值号右边表达式的类型会自动转换为赋值号左边变量类型。

类型转换（续）

■ 强制类型转换（cast）--- 显式类型转换

（<类型名>）<表达式>

如：

```
int n;  
n = 5;  
x = sqrt((double) n );
```

```
int int_part;  
double x, decimal_part;  
int_part = (int)x;    /* 获取x的整数部分*/  
decimal_part = x - (int)x; /*获取x的小数部分*/
```



表达式与运算符

- 在C语言中，通常由**运算符**(operators)及运算对象(operands)构成的式子称为**表达式** (expression)，如： $x + y * z$ 。此外，单个变量或函数调用也是一个表达式。
- 在C语言中，一个表达式后跟一个分号可构成一条语句：

〈表达式〉; 称为表达式语句

如：

$x++;$

$n = 5;$

$a = b = c = 0;$



算术运算符

- 算术运算符: $+, -, *, /, \%$

- 求余运算符 (%) 只能用

- 在算术运算时, 注意整除

```
double f;  
f = 3 / 2;
```

f 值为 1.0 而不是 1.5,
若要 f 为 1.5, 则应写为:

$f = (\text{double})3 / 2;$

(强制类型转换)

或:

$f = 3.0/2;$

温度转换

摄氏温度转华氏温度的公式为:

$$F = (9/5) * C + 32$$

编程实现时要注意整除问题。

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int fahr,celsius;
```

```
    scanf("%d", &celsius);
```

```
    fahr = (9.0/5) * celsius + 32;
```

```
    /* fahr = (double)9/5*celsius + 32; */
```

```
    printf("%d\n", fahr);
```

```
    return 0;
```

```
}
```

若 N 是一个整数, 则:

$N \% 10$: 为其个位数

$(N/10) \% 10$: 为其十位数



赋值运算符

- 赋值运算符: $=, +=, -=, /=, \%=, *=, >>=, <<=$
 $, \&=, ^=, |=$

赋值表达式可表示为:

$(e1) \text{ op} = (e2)$

其中op为一个双目运算符(binary operators) (如+, -, *, /, %, ...), 其等价于:

$(e1) = (e1) \text{ op } (e2)$

如, $x += n$; 即 $x = x + n$;

注意:

$y *= n+1$; 等价于 $y = y * (n+1)$; 而不是 $y = y * n + 1$;



增(减)量运算符

- 增(减)量运算符: `++`, `--`
- 依据运算符与运算对象间的位置, 分为前置和后置运算
 - 前置运算是先进行增减量, 再取其值。
 - 后置运算是先取其值, 再进行增减量运算。

如:

```
n = 5;  
x = ++n;
```

结果:

x: 6, n: 6

```
n = 5;  
x = n++;
```

x: 5, n: 6

像`++`, `--`这类只要求一个运算对象的运算符又称为单目运算符或 (unary operators)。`++`, `--`运算符在有些书中称为自增 (减)

如果`++`或`--`运算符单独使用 (即不出现在含有其它运算符的表达式中), 则前置和后置运算相同, 如:

```
++i;
```

```
i++;
```

均为给变量`i`加1.



增(减)量运算符（续）

[例1_5]给出下列程序的输出结果

```
#include <stdio.h>
main( )
{
    int a, b, c;
    a = b = c = 0;
    a = ++b + ++c;
    printf("%d %d %d\n",a, b, c);          /* 2 1 1 */
    a = b++ + c++;
    printf("%d %d %d\n",a, b, c);          /* 2 2 2 */
    a = ++b + c++;
    printf("%d %d %d\n",a, b, c);          /* 5 3 3 */
    a = b-- + --c;
    printf("%d %d %d\n",a, b, c);          /* 5 2 2 */
}
```

运算符优先级及结合律

优先级	运算符	结合律	
	初等量运算符		初等量运算符
1	() [] . -> 单目运算符		
2	- ! ~ ++ -- & * (类型名) sizeof 双目运算符	右结合	单目运算符
3	* / %		算术运算符
4	+ -		
5	<< >>		移位运算符
6	< <= >= >		关系运算符
7	== !=		
8	&		按位运算符
9	^		
10			
11	&&		逻辑运算符
12	 三目运算符		
13	? :	右结合	
	赋值运算符		
14	*= /= %= += -= <<= >>= &= ^= = 逗号运算符	右结合	
15	,		

应该记住常用运算符的优先级. 由于运行符优先级而造成程序出错也是程序员常犯的错误之一。



运算符优先级及结合律（续）

- 可以使用（）运算符来改变表达式中运算符的计算次序。如：

$(x+y) / 12$

`if((x = n) > 0) ...`

`while((c = getchar()) != EOF) ...`

- 风格建议：在由多个运算符组成的表达式中应使用()来明确计算次序，这样更加清晰，可读性好，不易出错。



问题1.1

- 问题：“从键盘读入一个学生成绩，判断该学生成绩是否及格”

问题1.1： 解决步骤

■ 解决问题1.1的步骤可描述为：

读入学生成绩

如果学生成绩大于或等于60

输出信息 “Pass”

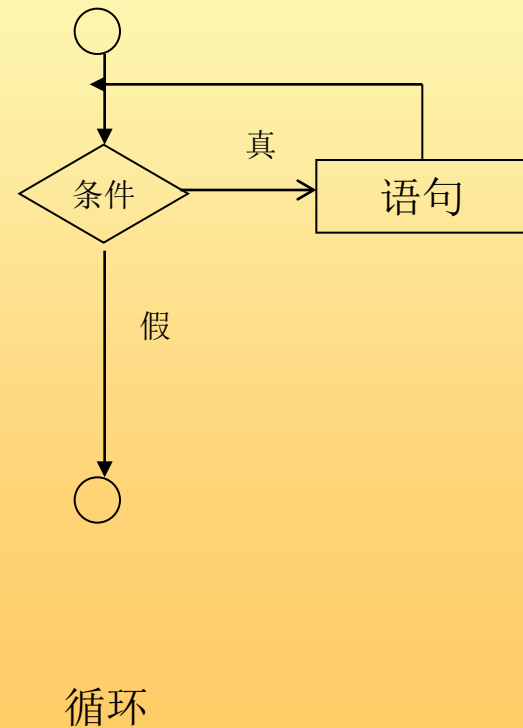
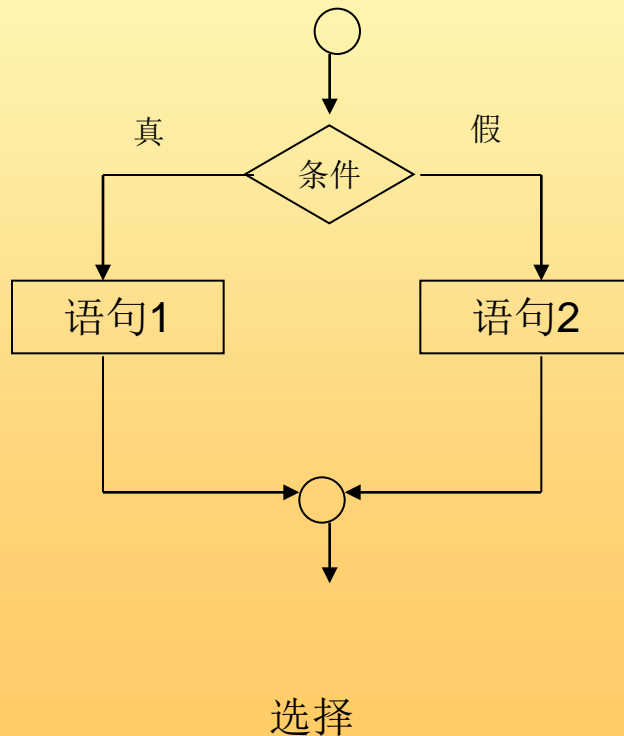
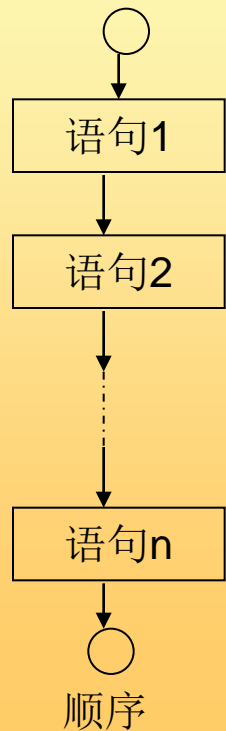
否则

输出信息 “Fail”

控制语句

控制语句结构

- 计算机语言通常提供三种控制方式来控制算法的执行：顺序（Sequence）、选择（Selection）和循环（Loop）。



条件表达式：关系运算符及逻辑运算符

- 关系运算符： $>$, $<$, $>=$, $<=$, $==$, $!=$
- 逻辑运算符： $\&\&$, $\|\|$, $!$

$\&\&$	T	F
T	T	F
F	F	F

$\ \ $	T	F
T	T	T
F	T	F

例如：

A $\&\&$ B 若A为0（假），则B

A $\|\|$ B 若A为非0（真），则

$(5 >= 3) \|\| (x == 5)$

$(2 == 3) \&\& (x >= 7)$

注意：在C语言中，没有Bool（布尔）类型，它用非0值表示真（True），用0表示假（False）。因此，在C中，任何一个表达式都可作为条件。



条件表达式（续）

一些常见的条件表达式例子：

- 判断整型变量n的值为一个0到10之间的值：

```
( 0 <= n && n <= 10)
```

- 判断字符变量c是字母：

```
( 'a' <= c && c <= 'z' ) |  
&& c <= 'Z' )
```

注意：

不能写为 “**0 <= n <= 10**”

不能得到预期结果

（该表达式是一个合法的逻辑表达式，分析一下其值为多少？）

- 判断某年是平年还是闰年（闰年为能被4整除但不能被100整除，或能被400整除）：

```
( ( y % 4 == 0 ) && ( y % 100 != 0 ) ) ||  
( y % 400 == 0 )
```



语句

- C语言语句分成简单语句和构造语句两类。

- 简单语句

- 1) 表达式(expression)语句

赋值表达式语句, 如: `x = y *= z+3;`

其它表达式语句, 如: `++x; --y;`

函数调用语句等, 如: `printf(“Hello, World!\n”);`

- 2) 转移语句

`goto` 标号;

`break;` 间断语句

`continue;` 继续语句

`return;` `return (表达式);` 返回语句

- 3) 空语句

;



语句（续）

■ 构造语句

1) 复合语句（又称块语句）（compound or block statement）

```
{  
    [ <局部数据说明> ]  
    <语句> *  
}
```

注意：}后没有分号（；），这与单个语句不同。

2) 选择（条件）语句

if语句，if_else语句

3) 循环语句

for语句

while语句

do_while语句

4) 开关语句

switch语句（包括case语句）



问题1.1：代码实现与测试

- 根据该问题解题步骤描述，很容易将问题“判断学生成绩是否及格”的解决转换为相应的程序。

读入 学生成绩
如果 成绩 ≥ 60
 输出 “及格”
否则
 输出 “不及格”

解题步骤

例1-4

/*判断学生成绩是否及格*/

#include <stdio.h>

int main()

{

int score;

scanf("%d", &score);

if(score ≥ 60)

printf("Pass\n");

else

printf("Fail\n");

return 0;

}

程序

if选择(条件)
语句

程序设计的常用方法为首先给出问题的解决步骤描述(算法)，然后将其编程序实现。



问题1.1： 代码实现与测试

如何判断程序解决了相应问题（能正确运行）？可用下面输入数据来测试（检验）：

75（输出应为“**Pass**”）

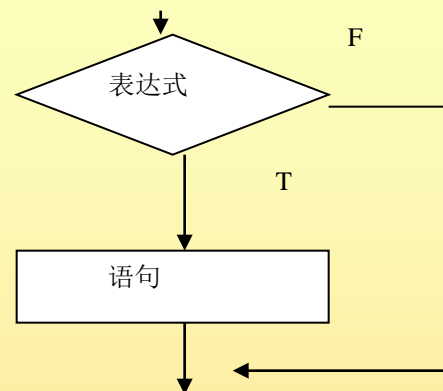
30（输出应为“**Fail**”）

60（输出应为“**Pass**”）-- 特殊数据

选择结构：if语句

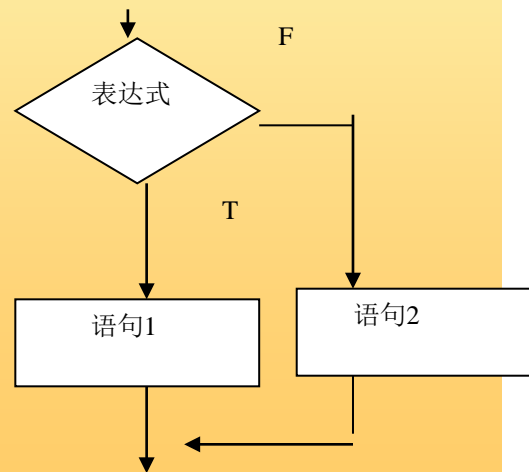
if(表达式)
语句

条件体语句为多个语句情况：
if(表达式) {
语句1;
...
语句n;
}



if(表达式)
语句1
else
语句2

if(表达式) {
语句1;
...
}
else {
语句1;
...
}






选择结构：if语句（续）

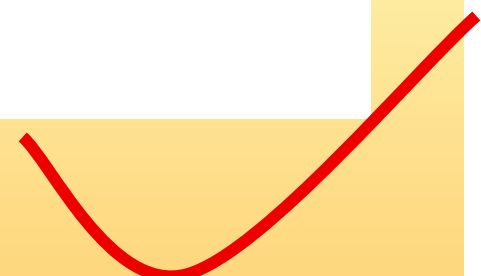
例：

```
if(a>b)
    max=a;
else
    max=b;
```

```
if(a>b)
    max=a;
    min=b;
```



```
if(a>b)
{
    max=a;
    min=b;
}
```





选择结构：if语句（续）

注意：在if嵌套中，省略else会产生二义性。如：

```
if( n > 0)
    if(a > b)
        z = a;
    else
        z = b;
```

即else与前面最近的不带else的if相对应。若要使上面的else与第一个if相匹配，可使用{}。如：

```
if(n > 0) {
    if(a > b)
        z = a;
}
else
    z = b;
```



条件运算符（?:）与条件表达式

- 条件运算符（三目运算符）： **?:**
- 条件表达式：

〈表达式1〉 ? 〈表达式2〉 : 〈表达式3〉

先计算**表达式1**，若其值为非零（为真），则整个表达式结果为**表达式2**的值，否则（为假）就为**表达式3**的值。

例：计算a和b的最大值

```
if( a > b)
```

```
    z = a;
```

```
else
```

```
    z = b;
```

等价于：

```
z = ( a > b ) ? a : b;
```

问题1.2

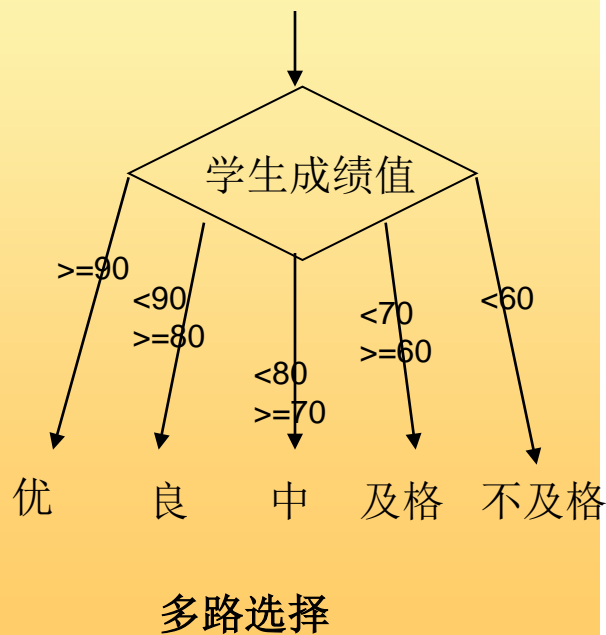
- 问题：“判断某学生成绩对应的（优、良、中、及格和不及格）五级评分成绩”



问题1.2：解题步骤

■ 解决该问题的步骤如下：

读入学生成绩值
如果该成绩值大于或等于90
 输出信息“优”
否则
 如果该成绩值大于或等于80
 输出信息“良”
 否则
 如果该成绩值大于或等于70
 输出信息“中”
 否则
 如果该成绩值大于或等于60
 输出信息“及格”
 否则
 输出信息“不及格”



问题1.2：代码实现与测试

读入学生成绩值

如果该成绩值大于或等于90

输出信息“优”

否则

如果该成绩值大于或等于80

输出信息“良”

否则

如果该成绩值大于或等于70

输出信息“中”

否则

如果该成绩值大于或等于60

输出信息“及格”

否则

输出信息“不及格”

例1-5

/*判断学生成绩对应的五级评分*/

```
#include <stdio.h>
```

```
int main( )
```

```
{
```

```
    int score;
```

```
    scanf("%d", &score);
```

```
    if( score >= 90)
```

```
        printf("A\n");
```

```
    else
```

```
        if( score >= 80)
```

```
            printf("B\n");
```

```
        else
```

```
            if(score >= 70)
```

```
                printf("C\n");
```

```
            else
```

```
                if(score >= 60)
```

```
                    printf("D\n");
```

```
                    printf("F\n");
```

```
    return 0;
```

```
}
```

嵌套if结构

测试数据为：

95 90

85 80

75 70

65 60

55

哪些数据是特殊（边界）数据？



多路选择：if_else if

例1-5a

/*判断学生成绩对应的五级评分*/

```
#include <stdio.h>
```

```
int main( )
```

```
{
```

```
    int score;
```

```
    scanf("%d", &score);
```

```
    if( score >= 90)
```

```
        printf("A\n");
```

```
    else if( score >= 80)
```

```
        printf("B\n");
```

```
    else if(score >= 70)
```

```
        printf("C\n");
```

```
    else if(score >= 60)
```

```
        printf("D\n");
```

```
    else
```

```
        printf("F\n");
```

```
    return 0;
```

```
}
```

一种更好的风格！

程序设计实践：编译（语法）错误及定位

■ 程序员常犯如下三类语法错误

- 变量名拼写错误（使用未定义变量名）；
- 遗漏语句结束符（";"）
- 遗漏复合语句结束符（"}"）

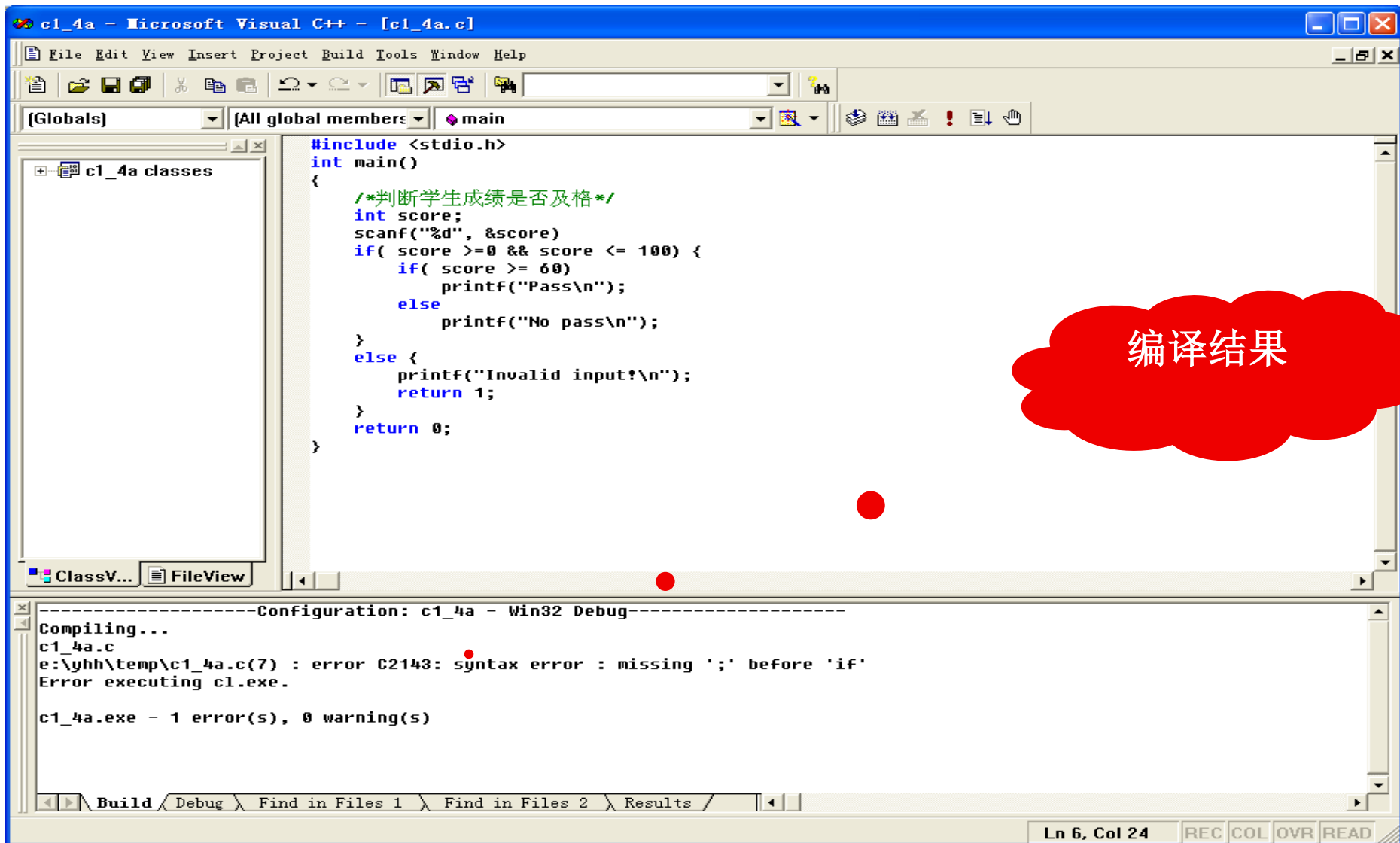
在VC下，编译错误提示信息为：
“**error C2065: 'xxx' : undeclared identifier**”

在VC下，编译错误提示信息为：
“**error C2146: syntax error : missing ';' before identifier 'xxx'**”

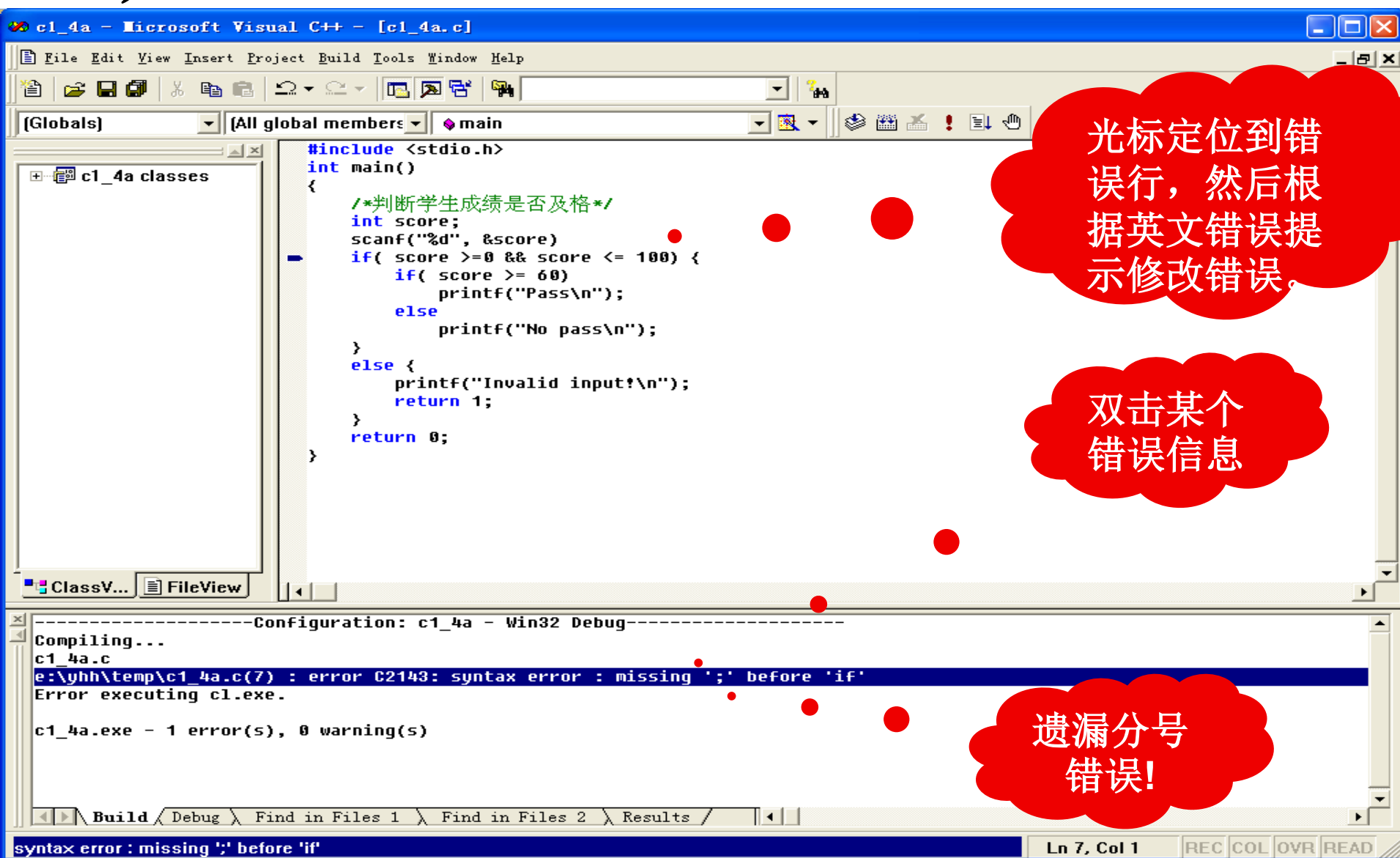
■ 下面例1_4来说明如何利用编译信息判断语法错误

如，遗漏最后函数结束符，在VC下，编译错误提示信息为：
“**fatal error C1004: unexpected end of file found**”

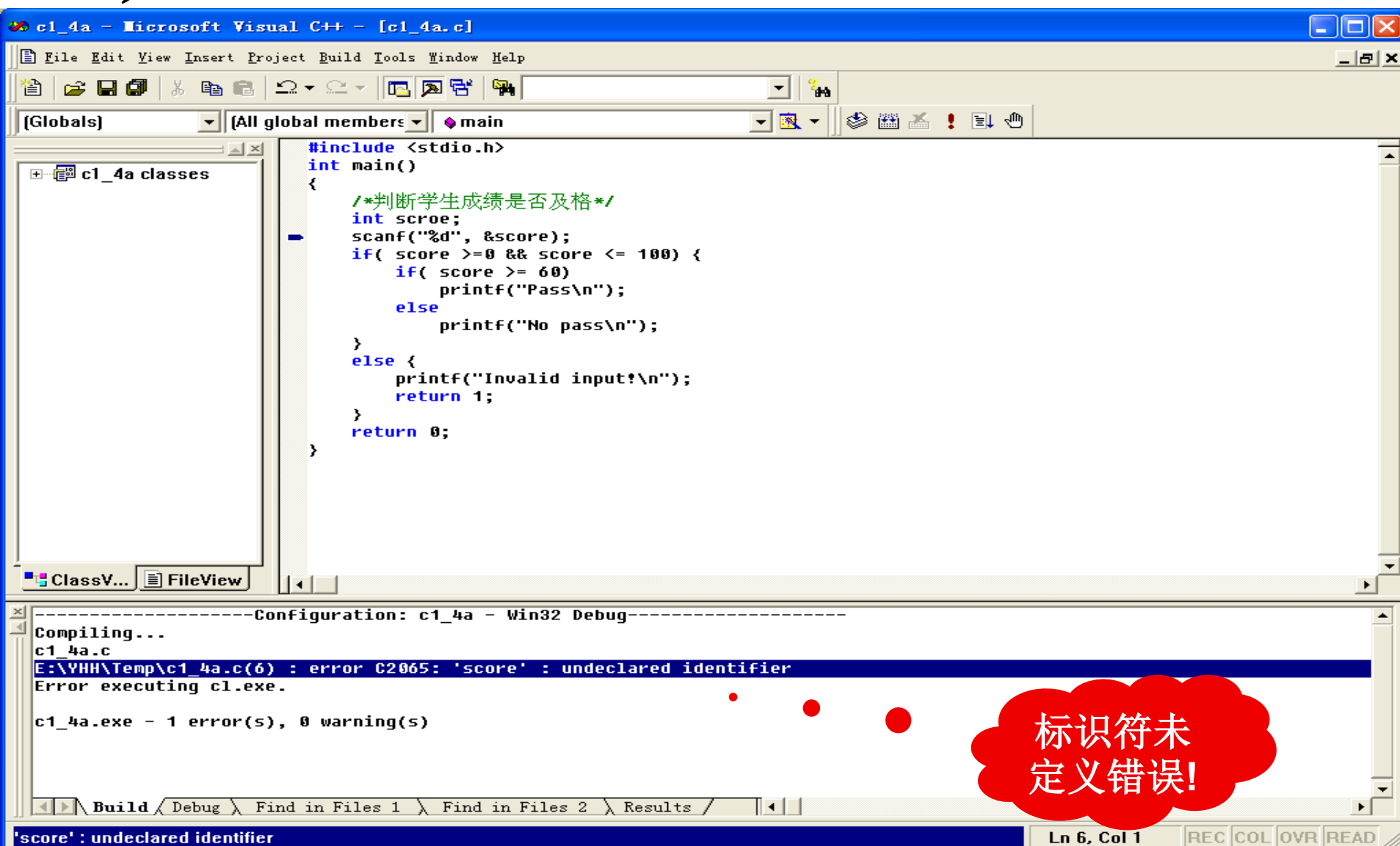
程序设计实践：编译（语法）错误及定位（续）



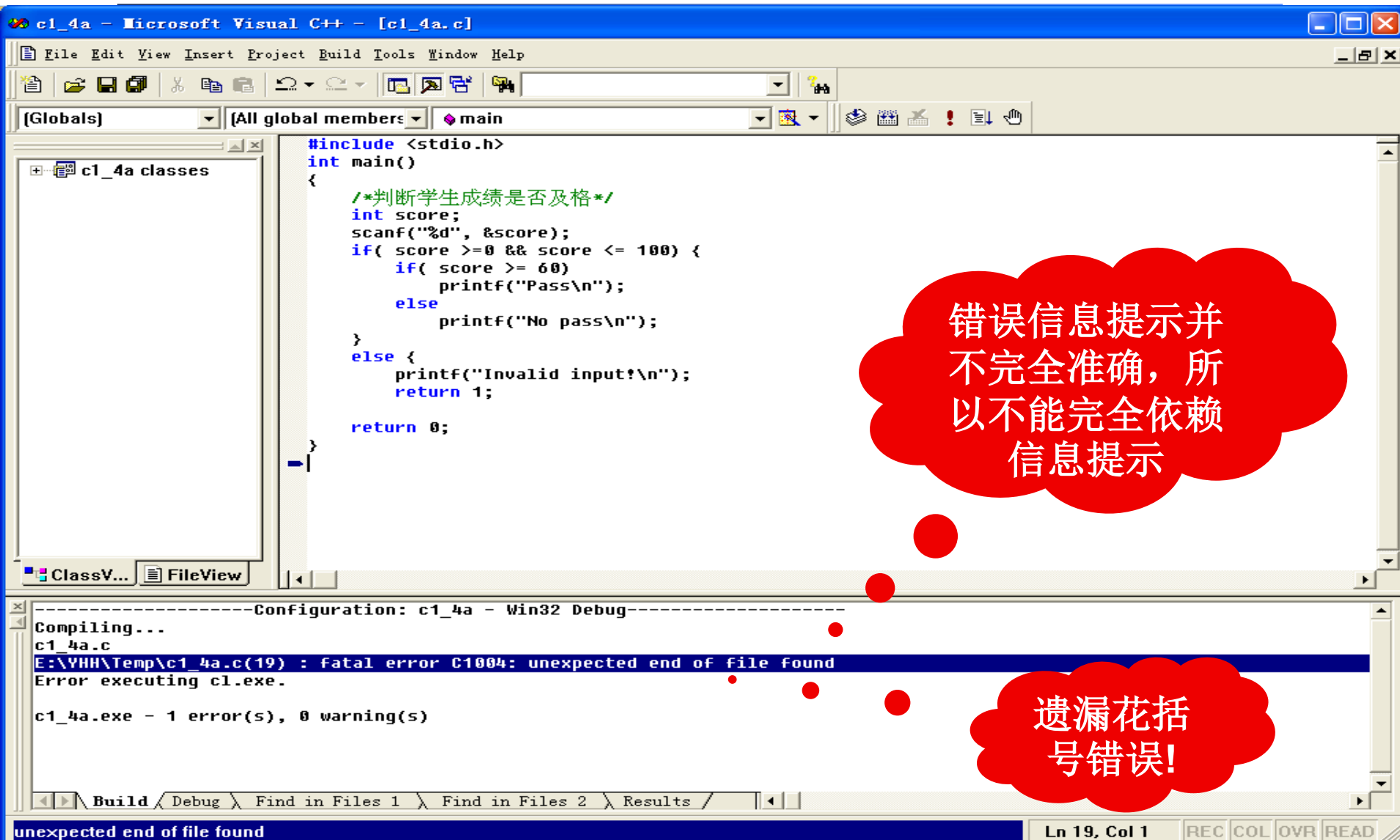
程序设计实践：编译（语法）错误及定位（续）



程序设计实践：编译（语法）错误及定位（续）



程序设计实践：编译（语法）错误及定位（续）





问题1.3

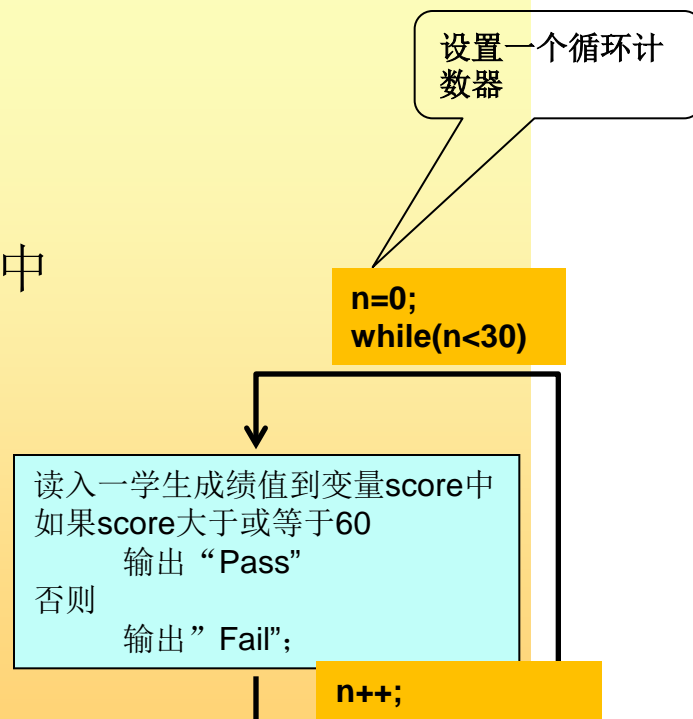
- 问题：“某班有30名学生，输入每个学生成绩并判断其是否及格”。



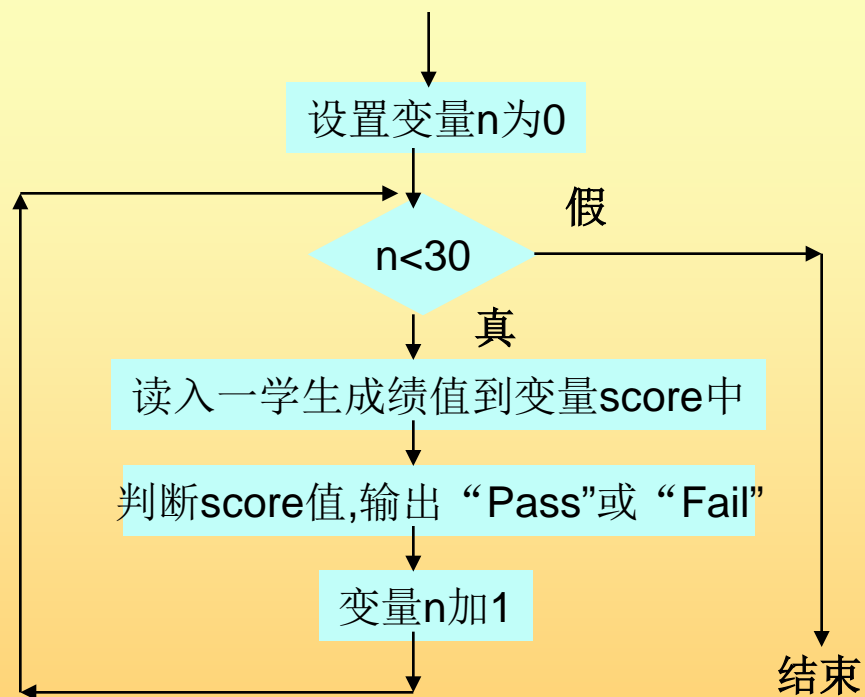
问题1.3：解决步骤

步骤：

1. 设置变量n为0
2. 当n小于30时
 - 2.1 读入一学生成绩值到变量score中
 - 2.2 如果score大于或等于60
输出“Pass”
否则
输出“Fail”；
 - 2.3 变量n加1, 转步骤2



问题1.3： 解决步骤(续)



程序流程图

问题1.3：代码实现

步骤：

1. 设置变量n为0
2. 当n小于30时
 - 2.1 读入一学生成绩值到变量score中
 - 2.2 如果score大于或等于60
 - 输出 “Pass”
 - 否则
 - 输出 “Fail”;
 - 2.3 变量n加1, 转步骤2

例1-6

/*判断某班学生成绩是否及格*/

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int n, score;
```

```
    n = 0;
```

```
    while(n < 30) {
```

```
        scanf("%d", &score);
```

```
        printf("score = %d\n", n);
```

```
        if(score < 60)
```

```
            printf("Fail\n");
```

```
        else
```

```
            printf("Pass\n");
```

```
        n++;
```

```
    }
```

```
    return 0;
```

```
}
```

while循环语句

测试数据考虑：

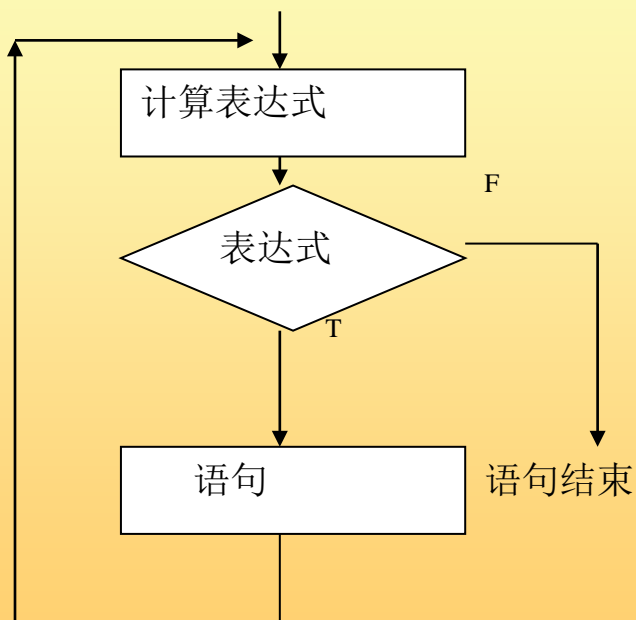
依次输入30个成绩值，这些值中应包括100,60,0等特殊（边界）值。

循环结构：while语句

while（表达式） 语句

循环体语句为多个语句情况：

```
while(表达式) {  
    语句1;  
    ...  
    语句n;  
}
```





循环结构：while语句(续)

例1-7：从键盘读入字符并输出

```
#include <stdio.h>

int main()
{
    int c;
    while(( c = getchar())!= EOF)
        putchar( c );
    return 0;
}
```

步骤分析：

读入一个字符;
while 不是输入结束符
输出字符;
读入

标准字符输入函数
int c;
c = getchar();
或
scanf("%c", &c);

标准字符输出函数
putchar(int c);
如,putchar('A');
或
printf("%c", c);

2) 为何程序运行后
hello <enter>
hello

3) 为何用int c, 而不用
char c;



循环结构：for语句

只执行一次通常用来初始化循环变量。
通常用来改变循环变量。
通常根据循环变量是否满足某个条件来终止循环。

for (表达式1; 表达式2; 表达式3)

语句

其等价于：

表达式1;

while (表达式2) {

语句;

表达式3;

}

循环体语句为多个语句情况：

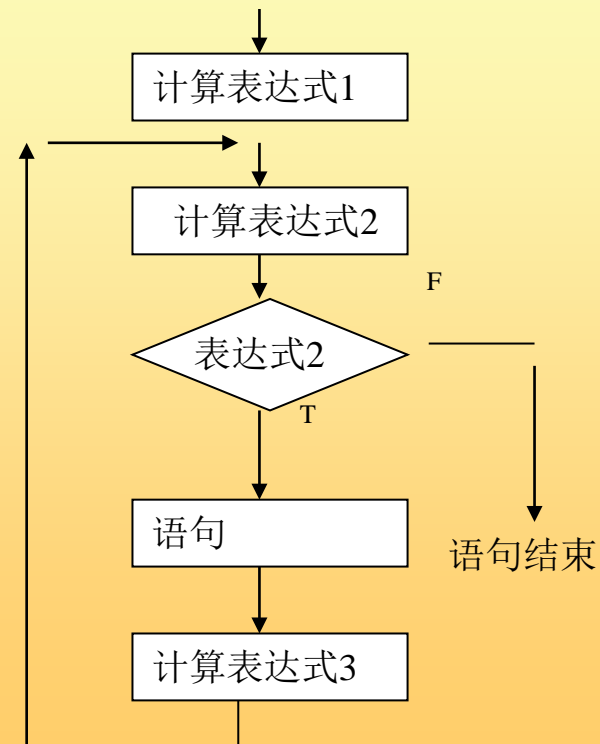
for(表达式1;表达式2;表达式3) {

语句1;

...

语句n;

}





循环结构：for语句(续)

for语句特别适合于循环次数固定或有固定步长的循环，如下面是一个循环十次的循环：

```
for( i=0; i<10; i++)  
    ...
```

当然还有各种变化应用，在今后的例子中将会看到。关键是要理解for循环头中三部分表达式的含义。



循环结构：for语句(续)

例1-6a

/*判断某班学生成绩是否及格*/

```
#include <stdio.h>
```

```
int main( )
```

```
{
```

```
    int n, score;
```

```
    for (n=0; n < 30; n++) {
```

```
        scanf( "%d" , &score);
```

```
        if(score >= 60)
```

```
            printf( "Pass\n" );
```

```
        else
```

```
            printf( "Fail\n" );
```

```
    }
```

```
    return 0;
```

```
}
```

例1-6

/*判断某班学生成绩是否及格*/

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int n, score;
```

```
    n = 0;
```

```
    while(n < 30) {
```

```
        scanf("%d", &score);
```

```
        if( score >= 60)
```

```
            printf("Pass\n");
```

```
        else
```

```
            printf("Fail\n");
```

```
            n++;
```

```
    }
```

```
    return 0;
```

```
}
```

循环结构：for语句(续)

注意：从语法上讲，表达式中任一个都允许省略，但分号不能省。

如：

```
for( ; (c = getchar( )) != EOF; )  
    语句s;
```

```
for( ; ; )  
    语句s;
```

当表达式2不出现时，则认为条件为真，循环永远下去（无穷循环），退出它只能用break或return语句。

因此，

```
while(A)  
    B;
```

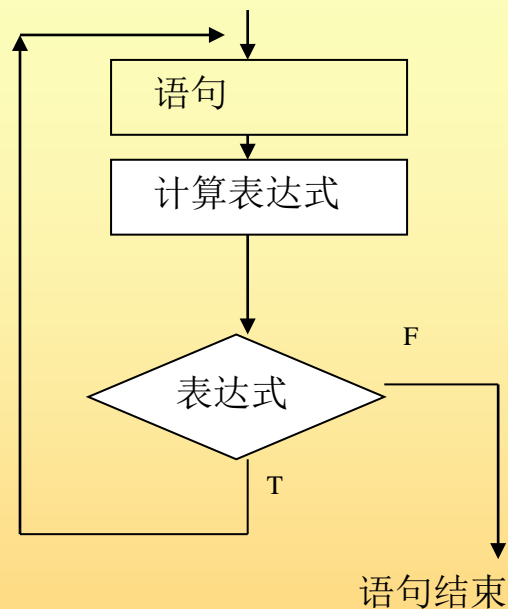
等价于

```
for(; A; )  
    B;
```

循环结构：do_while语句

```
do  
    语句  
while (表达式);
```

作用类似于while，但do_while是否终止，其判定条件是在循环体之后，即它总是先执行一次循环体，再判断表达式的值是否为真。



循环体语句为多个语句情况：

```
do {  
    语句1;  
    ...  
    语句n;  
} while(表达式);
```



循环结构：do_while语句(续)

总的说来，while(for)循环是在顶部测试循环终止条件，而do_while是在每次通过循环体之后，在底部进行测试，所以循环体至少要进行一次。

注意：do_while与Pascal语言中的Repeat...until不同---其布尔表达式为False时继续循环，若为True时，终止循环。



循环结构：do_while语句(续)*

例1-8：求立方根，已知 x_1 ，使用公式 $x_{n+1} = (2x_n + x/x_n^2)/3$ 计算 x 的立方根， $x_1=x$ ，在满足 $|x_{n+1}-x_n|<10^{-6}$ 时停止计算。

```
#define DELTA 1e-6
int main()
{
    double x, x1, x2, d;
    scanf( "%lf" , &x);
    x2 = x;
    do {
        x1 = x2;
        x2 = (2.0 * x1 + x / ( x1 * x1)) / 3.0;
        d = x2 -x1;
    } while ( d > DELTA || d < -DELTA);
    printf( " %.6f\n" , x2);
    return 0;
}
```



循环结构

三种循环主要的用途：

- `for`通常用于固定步长的循环
- `while`通常用于循环条件在头部判断的循环（即没有循环初始化操作，也没有循环控制状态有规律的修改）
- `do_while`通常用于至少循环一次的循环（即先执行后判断）

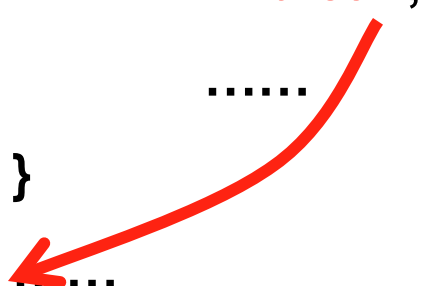
break和continue语句*

- **break**: 迫使程序从包含它的最内层循环体或switch语句中跳出（循环只能跳出一层）。
- **continue**: 迫使包含它的最内层循环体立即执行下一次循环（不管现在程序执行到何处）。

```

.....
while(.....)
{
    .....
    if(...)
        break;
    .....
}
.....

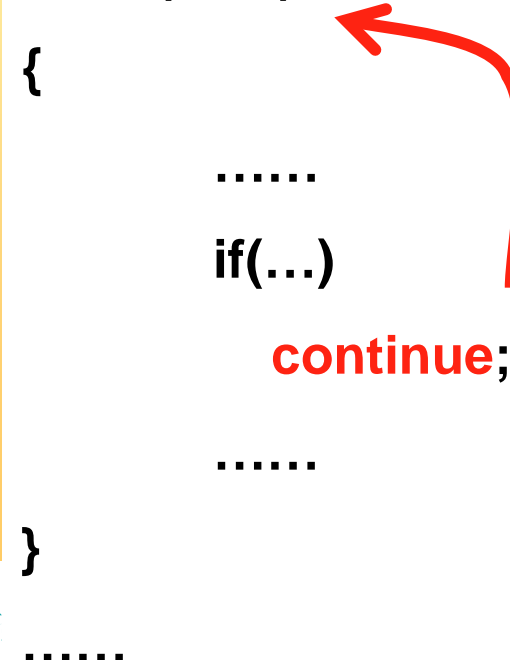
```



```

.....
while(.....)
{
    .....
    if(...)
        continue;
    .....
}
.....

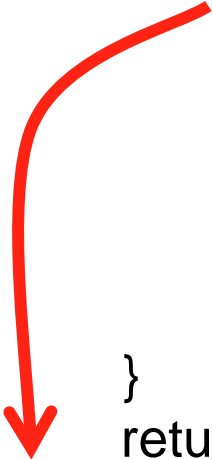
```





break和continue语句(续) *

```
/*判断某班学生成绩对应的五级评分*/
#include <stdio.h>
int main( )
{
    int n, score;
    for (n=0; n < 30; n++) {
        scanf("%d", &score);
        if(score >100 || sorce < 0){
            printf("Input error!\n");
            break;
        }
        if(score >= 60)
            printf("Pass\n");
        else
            printf("Fail\n");
    }
    return 0;
}
```





break和continue语句(续)*

例：统计非数字字符数。

```
int main( )
{
    int count;
    char c;
    count = 0;
    while(( c = getchar(
        if( c >= '0' &&
            continue;
        count++;
    }
    printf("non digital character: %d\n", count);
    return 0;
}
```

不用**continue**语句同样可以实现该程序：

```
int main( )
{
    int count;
    char c;
    count = 0;
    while(( c = getchar( )) != EOF)
        if( c < '0' || c > '9')
            count++;
    printf("non digital character: %d\n", count);
    return 0;
}
```



程序设计实践：测试（Testing）

- 如何验证一个程序解决了问题（即实现了所要的功能），可分析每个输入数据的范围，然后对每个数据从下面几个方面来考虑测试数据：
 - 正常数据（范围内，如问题2.1中的75），以确定程序做了该做的事；
 - 边界数据（范围边界，如问题2.4中的60, 0, 100），以确定程序在数据边界上是否没有错误；
 - 非法数据（范围外，如问题2.4中的-1, 105），以确定程序没有错误处理；

如何发现边界数据？

提示：当软件行为将发现变化的拐点数据就是边界数据。对于整数输入数据来说，如果某值加1或减1后，软件行为将发生变化，其就是边界数据。如在判断学生成绩的例子中，60, 0, 100就是边界数据。

如何发现非法数据？

提示：对于一个软件来说，不合法的、无效的或无意义的输入数据就是非法数据。如在判断学生成绩的例子中，-1, 105就是非法数据。



程序设计实践：测试（Testing）

问题2.3：“某班有30名学生，输入每个学生成绩并判断其是否及格”。

■ 测试设计：

- 正常数据：80 45 90 56
- 边界数据：60 0 100
- 非法数据：-10 200



程序设计实践：调试程序(Debug)

■ 调试(Debug)：定位并解决问题

■ 调试方法：

- 简单：使用打印语句
- 高级：使用编程环境

■ 调试方式：

- 设置/删除断点 (Insert)
- 查看变量 (Watch)
- 单步执行 (Step Over / Step Into)
- 执行到光标处 (Run to Cursor)

调试策略

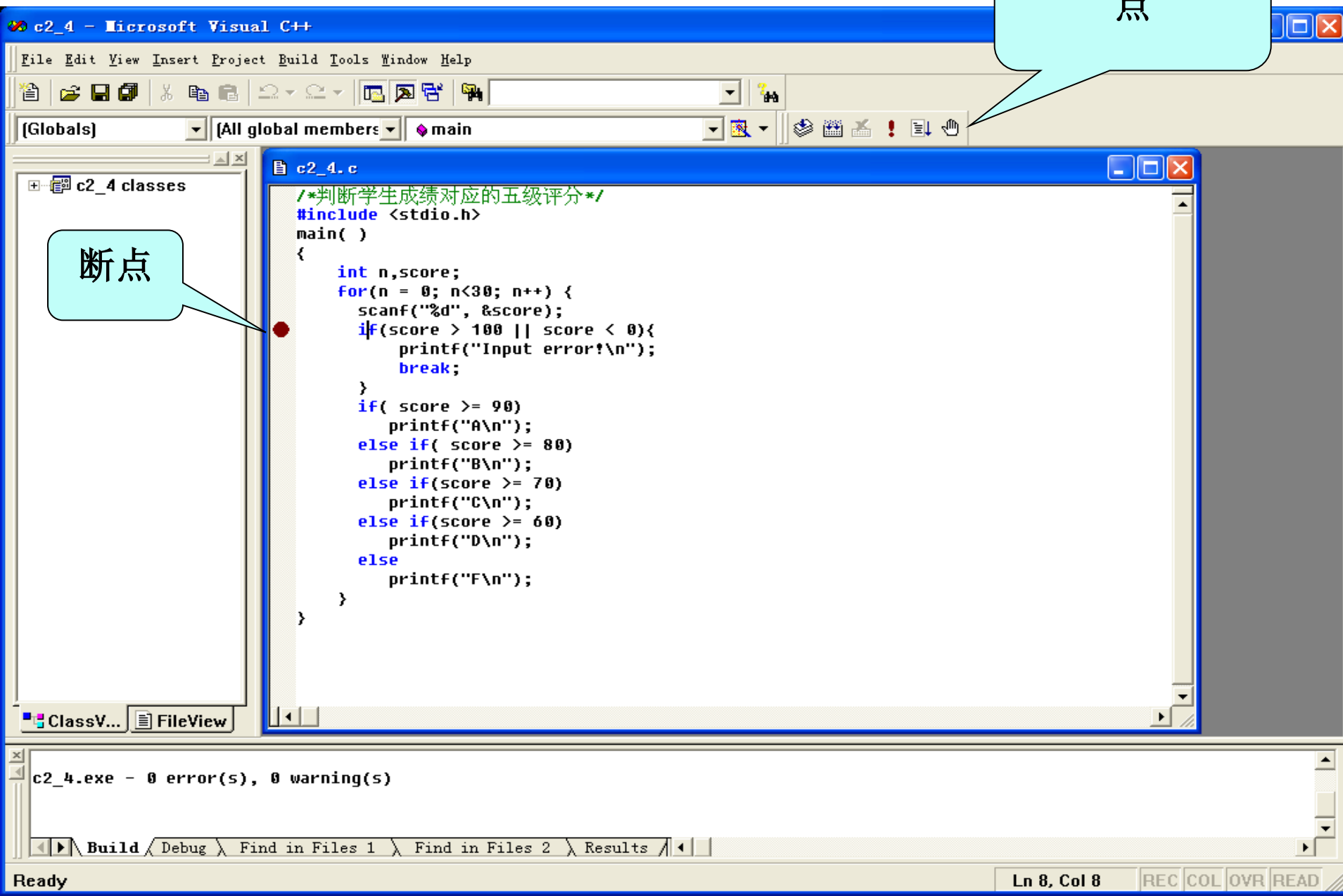
调试的前提是程序有一个反例（即有一个使程序运行出错的输入，而且该出错能始终再现）。

1. 首先在怀疑程序可能出错点前设置断点；（如输入语句、if语句或循环语句前）
2. 单步执行一次或多次；
3. 查看程序执行是否正确（主要查看变量的值是否正确，或执行流程是否是你所期望的）；（如输入值是否被正确读入，或if语句是否执行到你所期望的语句）。
4. 排除完一个出错点后，继续在下一个怀疑点前重复前面步骤1-3，直到程序执行完全正确后，清除掉所有断点。

调试程序(Debug)(续): 设置断点

设置/删除断点

断点



调试程序(Debug)(续): 运行并查看变量

开始调试 (运行)

程序执行停在断点处

查看其它变量内容

单步执行(Step Over)
执行到光标处(Run to Cursor)

当前变量及内容

The screenshot shows the Microsoft Visual C++ 6.0 IDE with a C program being debugged. The program is paused at a breakpoint. The code in the editor is as follows:

```
/*  
#include <stdio.h>  
main  
{  
    int score;  
    for(n = 0; n < 30; n++) {  
        scanf("%d", &score);  
        if(score > 100 || score < 0){  
            printf("Input error!\n");  
            break;  
        }  
        if( score >= 90)  
            printf("A\n");  
        else if( score >= 80)  
            printf("B\n");  
        else if(score >= 70)  
            printf("C\n");  
        else if(score >= 60)  
            printf("D\n");  
        else  
            printf("F\n");  
    }  
}
```

The 'Debug' toolbar is visible, showing the 'Run to Cursor' button (a yellow arrow) and the 'Step Over' button (a blue arrow with a magnifying glass). The 'Watch' window at the bottom right shows the current state of variables:

Name	Value
n	0

The 'Auto' window at the bottom left shows the current state of variables:

Name	Value
&score	0x0012ff78
score	95

The status bar at the bottom indicates 'Ready' and 'Ln 8, Col 1'.



调试程序(Debug)(续): 一个范例

```
#include <stdio.h>
main( )
{
    int n,score;
    for(n = 0; n<3; n++) {
        scanf("%d", &score);
        if( score >= 90)
            printf("A\n");
        else if( score >= 80)
            printf("B\n");
        else if(score >= 70)
            printf("C\n");
        else if(score > 60)
            printf("D\n");
        else
            printf("F\n");
    }
}
```

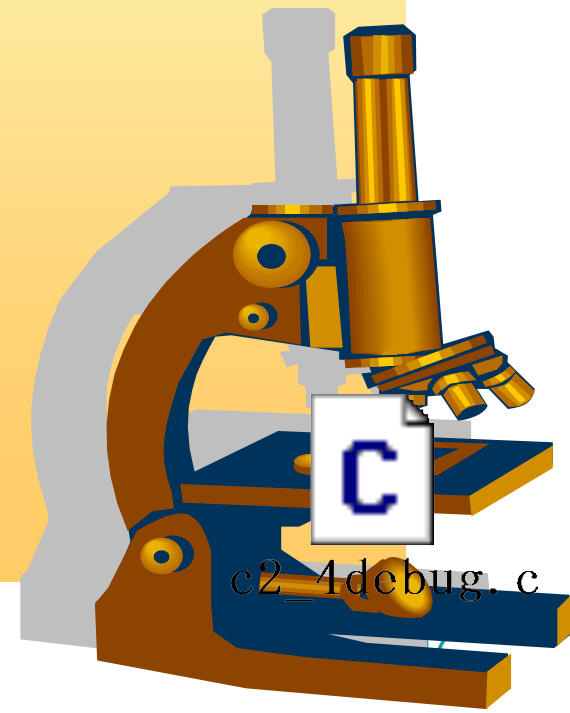
输入: 92 60 30

输出:

A

F

F





程序设计实践： 程序设计风格(Style or Convention)(一)

■ 为什么要强调程序设计风格：

可以改善软件的可读性，帮助程序员理解代码 。

```
#include <stdio.h>
int main( )
{
    int n, score;
    n = 0;
    while(n < 30) {
        scanf("%d", &score);
        printf("%d : ", n);
        if( score >= 60)
            printf("Pass\n");
        else
            printf("Fail\n");
        n++;
    }
    return 0;
}
```

好风格

```
#include <stdio.h>
int main( ){int n, score;
    n = 0;
    while(n < 30) {    scanf("%d", &score);
        printf("%d : ", n);if( score >= 60)
            printf("Pass\n"); else
        printf("Fail\n");
        n++;}        return 0;
}
```

不好风格

程序设计风格(Style or Convention)(一)(续)

■ 变量名与常量名:

- 变量名应简短且富于描述(自说明)。变量名的选用应该易于记忆, 即, 能够指出其用途, 如前面例子中用于存入学生成绩的变量score。
- 尽量避免单个字符的变量名, 除非是一次性的临时变量, 如循环变量。临时变量通常被取名为i, j, k, m和n。
- 常量应该全部大写, 单词间用下划线隔开。如

:

```
const int MIN_WIDTH = 4 ;  
#define MAX_LENGTH 100
```

有关编程命名规范最著名的是“匈牙利命名法”

程序设计风格(Style or Convention)(一)(续)

■ main函数格式:

```
int main()  
{  
  
    statements;  
  
    ...  
  
    return 0;  
  
}
```

程序设计风格(Style or Convention)(一)(续)

■ if语句格式:

if-else语句应该具有如下格式:

```
if (condition) {  
    statements;  
}
```

```
if (condition) {  
    statements;  
}
```

```
else {  
    statements;  
}
```

```
if (condition) {  
    statements;  
}  
else if (condition) {  
    statements;  
}  
else {  
    statements;  
}
```

■ while语句格式:

一个while语句应该具有如下格式:

```
while (condition) {  
    statements;  
}
```

■ for语句格式:

一个for语句应该具有如下格式:

```
for (initialization; condition; update) {  
    statements;  
}
```

■ do_while语句格式:

一个do-while语句应该具有如下格式:

```
do {  
    statements;  
} while (condition);
```

程序设计风格(Style or Convention)(一)(续)

■ switch语句格式:

一个switch语句应该具有如下格式:

```
switch (condition) {  
    case ABC:  
        statements;  
        /* falls through */  
    case DEF:  
        statements;  
        break;  
    case XYZ:  
        statements;  
        break;  
    default:  
        statements;  
        break;  
}
```


运算符优先级及结合律

优先级	运算符	结合律	
	初等量运算符		初等量运算符
1	() [] . -> 单目运算符		
2	- ! ~ ++ -- & * (类型名) sizeof 双目运算符	右结合	单目运算符
3	* / %		算术运算符
4	+ -		
5	<< >>		移位运算符
6	< <= >= >		关系运算符
7	== !=		
8	&		按位运算符
9	^		
10			
11	&&		逻辑运算符
12	 三目运算符		
13	? :	右结合	
	赋值运算符		
14	*= /= %= += -= <<= >>= &= ^= = 逗号运算符	右结合	
15	,		

本讲结束