

程序设计实践：程序设计风格(Style or Convention)

- 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)(续)

- 变量名与常量名、函数名：
 - ◆ 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
```
 - ◆ 函数名应简短且富于描述(自说明)。如：

```
getchar(), sqrt()...
```

程序设计风格(Style or Convention)(续)

- 一行一条语句;
- 程序要有注释;
- 一个函数不要太长 (建议在100行以内)
- **main函数格式:**

```
int main()
{
    statements;

    ...
    return 0;
}
```

程序设计风格(Style or Convention)(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)(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;  
}
```

风格建议:
程序设计基本风格可参考
B. W. Kernighan &
D. M. Ritchie的书“C程序
设计语言”或尹老师教材
。包括变量和函数命名、
程序缩进格式以及常见语
句用法等。

程序设计实践：测试（Test）

- 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，通常可用样例中数据），以确定程序做了该做的事；
 - ◆ 边界数据（范围边界，如问题3.1中的0和1，问题3.3中a-b-c），以确定程序在数据边界上处理没有错；
 - ◆ 非法数据（范围外，如问题3.3中的a-a），以确定程序没有做不该做的事，即进行了错误处理；

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)

会调试是作为一名程序员应具备的基本能力！

- 调试(Debug): 定位并解决问题
- 调试方法:
 - ◆ 人工：走查 (walkthrough)；
 - ◆ 简单：使用打印语句 (printf)；
 - ◆ 高级：使用编程环境所带的调试工具；
- 调试方式:
 - ◆ 设置/删除断点 (Insert/Remove Breakpoint)
 - ◆ 软件运行到断点处
 - ◆ 单步执行 (Step Over / Step Into)
 - ◆ 查看变量 (Watch)

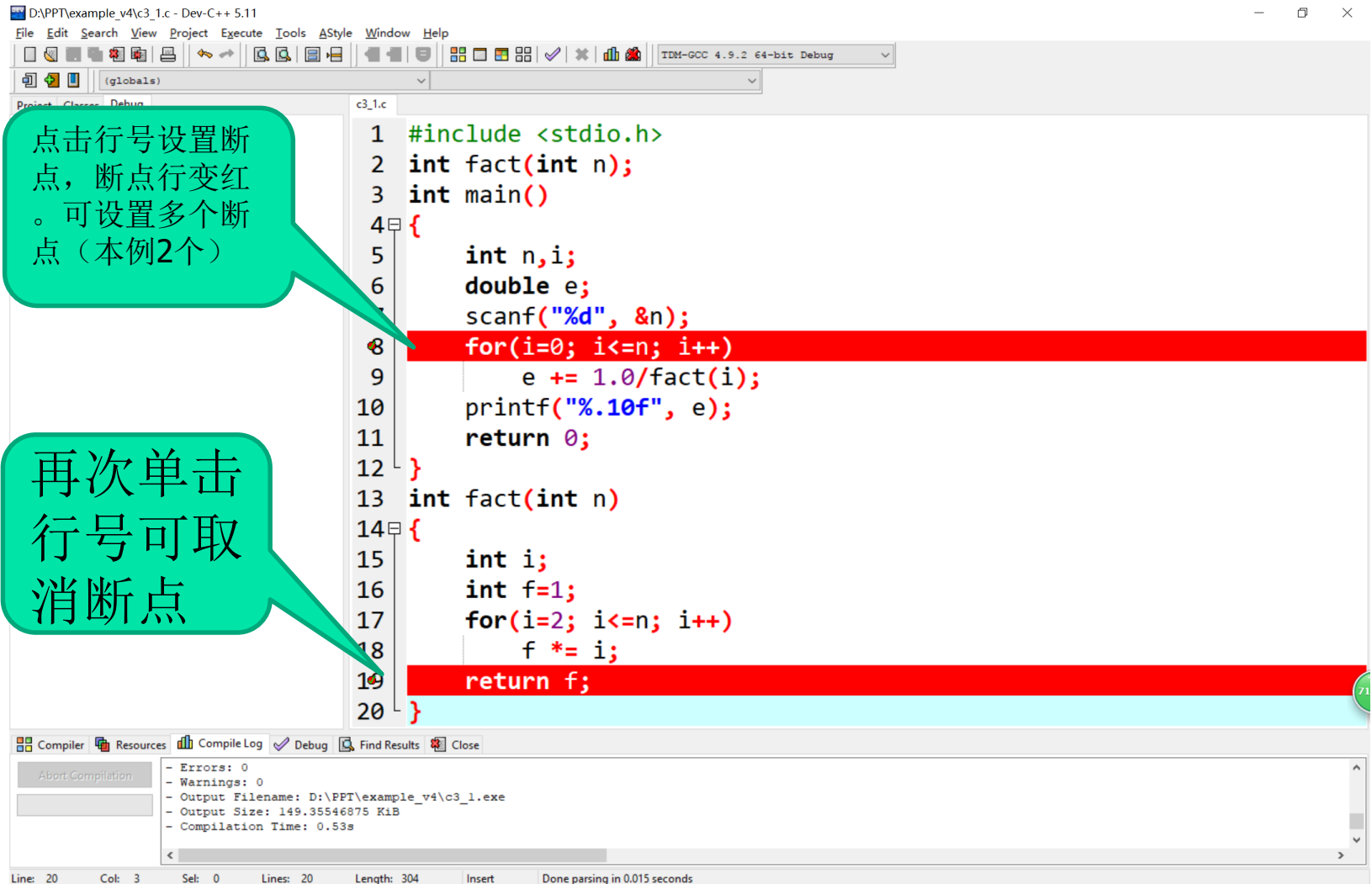
调试策略

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. 用该输入运行程序，程序停在断点处后，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. 如果找到出错点，则修改代码后，重新运行程序，检查bug是否已解决；
5. 如果该断点附近没有出错，则继续在下一个怀疑点前重复前面步骤1-3，直到程序错误排除，执行完全正确后，清除掉所有断点。

Dev-C++断点设置

一般在读入数据之后或者一段重要的功能程序片段后设置断点，本例在读入数据后、计算阶乘后设置了断点



点击行号设置断点，断点行变红。可设置多个断点（本例2个）

再次单击行号可取消断点

```
1 #include <stdio.h>
2 int fact(int n);
3 int main()
4 {
5     int n,i;
6     double e;
7     scanf("%d", &n);
8     for(i=0; i<=n; i++)
9         e += 1.0/fact(i);
10    printf("%.10f", e);
11    return 0;
12 }
13 int fact(int n)
14 {
15     int i;
16     int f=1;
17     for(i=2; i<=n; i++)
18         f *= i;
19     return f;
20 }
```

Compiler Resources Compile Log Debug Find Results Close

Abort Compilation

- Errors: 0
- Warnings: 0
- Output Filename: D:\PPT\example_v4\c3_1.exe
- Output Size: 149.35546875 KiB
- Compilation Time: 0.53s

Line: 20 Col: 3 Sel: 0 Lines: 20 Length: 304 Insert Done parsing in 0.015 seconds

Dev-C++ 查看变量

另一种变量数据查看方法（推荐使用该方法）：

The screenshot shows the Dev-C++ IDE with a C++ program being debugged. The program is `c3_1.c` and contains a factorial function. The variable `e` is shown in the Project Explorer with its value `1.3339772437713657e-322`. The source code editor shows the following code:

```
1 #include <stdio.h>
2 int fact(int n)
3 {
4     if (n == 1)
5         return 1;
6     double e;
7     scanf("%d", &n);
8     for(i=0; i<=n; i++)
9         e = e * i;
10 }
11
12 int main()
13 {
14     int n;
15     int f=1;
16     for(i=2; i<=n; i++)
17         f *= i;
18     return f;
19 }
20 }
```

The following steps are illustrated by callouts:

- 1、鼠标右键单击左侧的空白窗口区域，在弹出的菜单中选择“Add watch”。或者选择下方Debug控制按钮中的“Add watch”按钮。
- 2、在弹出的对话框中输入程序中重要的变量名，每次输入一个。例如：本例中可输入e、n或f。
- 3、输入要监控的变量名后，变量在整个程序运行过程中的数据变化情况将显示在这里

The 'New Variable Watch' dialog is shown with the input field empty. The debug console at the bottom shows the command `display f` and the output `f = 1.3339772437713657e-322`.

在本例中，由于在第19行设置了断点，所以可以通过点击“Continue”按钮快速查看各整数的阶乘结果：

DEV F:\YHH\Course\数据结构与程序设计基础\yhh\PPT\examples\C_example\c3_1a(debug).c - [Debugging] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

(globals)

Project Classes Debug c3_1a(debug).c

n = 13
f = 1932053504

```
1 #include <stdio.h>
2 int fact(int n);
3 int main()
4 {
5     int n,i;
6     double e = 0.0;
7     scanf("%d", &n);
8     for(i=0; i<=n; i++)
9         e += 1.0/fact(i);
10    printf("%.10f", e);
11    return 0;
12 }
13 int fact(int n)
14 {
15     int i,f=1;
16     for(i=1; i<=n; i++)
17         f *= i;
18     return f;
19 }
```

1
!
1
!
2
!
6
!
24
!
120
!
720
!
5040
!
40320
!
362880
!
3628800
!
39916800
!
479001600
!
1932053504
!
1278945280

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

13!的计算结果，出现错误！
13! 应为：
622702080
0

通过上面记录的阶乘计算结果可以发现：从13!的计算结果开始会出现错误，发生溢出！

Compiler Resources Compile Log Debug Find Results Close

Debug Add watch Next line Continue Into instruction
Stop Execution View CPU window Into function Skip function

Send command to GDB: continue
->->stopped

常见错误

- 读double类型数据错误,如:

```
double a;  
scanf("%f", &a);
```

正确用法:

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

- 使用未初始化的局部变量,如:

```
int n;  
while(n++ < 10)  
...
```

正确用法:

```
int n = 0;  
while(n++ < 10)  
...
```

- **scanf**中空格使用错误, 如:

- int data1, data2;
- char op;
- scanf("%d%d%c", &data1,&data2,&op);

正确用法:

```
int data1, data2;  
char op;  
scanf("%d %d %c",  
      &data1,&data2,&op);
```

常见错误

- 数组只能在定义时进行整体初始化赋值；但不能通过赋值运算符进行整体赋值。如下面**初始化数组错误**：

```
int mon[13];  
mon[13]={0,31,28,31,30,31,30,31,31,30,31,30,31}
```

正确用法：

```
int mon[13]={0,31,28,31,30,  
31,30,31,31,30,31,30,31};
```

- 数组访问**越界错误**，如：

```
int i,array[10];  
for(i=1; i<=10; i++)  
    scanf("%d",&array[i]);
```

正确用法：

```
int i,array[10];  
for(i=0; i<10; i++)  
    scanf("%d",&array[i]);
```

- 数组作为函数**参数传递错误**

- ◆ 函数调用时错误,如：
fun(a[10]);

正确用法：

```
fun(a);
```

- 打开（某绝对路径下）**文件错误**

```
in = fopen("d:\input.txt", "r");
```

正确用法：

```
in = fopen("d:\\input.txt", "r");
```