

程序设计基础

(C Programming)

第三讲：程序设计方法-问题分析

北航计算机学院 晏海华

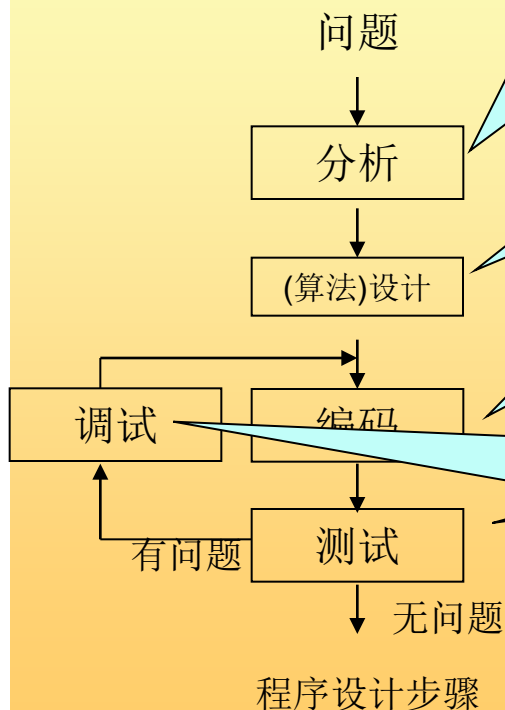


本章目标

- 了解一般程序设计过程
- 通过实例重点掌握问题分析方法
 - 精度计算;
 - 输入数据处理;
 - 字符串操作

程序设计过程

■ 程序设计就是解决问题 包括如下五个步骤



分析问题:

功能: 需要弄清楚软件要完成的功能;

输入: 如果问题有输入, 分析输入是什么及输入数据的类型和数据结构;

处理: 对输入数据做什么处理;

(算法)设计:

设计解决问题的具体方案(步骤、算法)

编码:

将算法用高级语言实现。

测试:

运行编译连接后得到的执行程序, 以验证程序是否按要求解决了问题, 并没有产生副作用。即程序是否做了该做的事, 同时没有做不该做的事。

调试:

如果程序经测试发现问题, 则通过调试手段找到产生错误的代码并修复它。



结构化程序设计（structured programming）

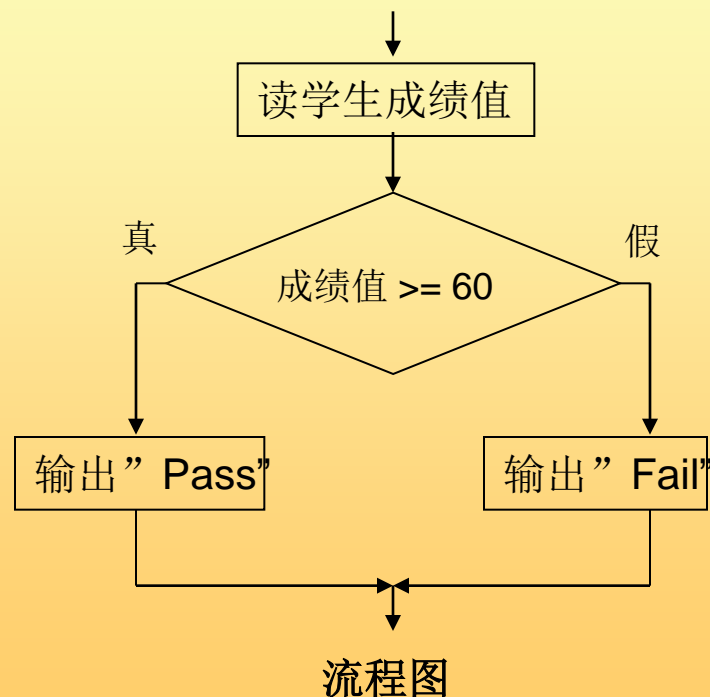
- 常用的程序设计方法为结构化程序设计，其特点为：
 - 自顶向下（top-down）；
 - 逐步细化（stepwise refinement）；
 - 模块化（modular）；

- 任何计算问题的解决都是按指定的顺序执行一系列动作的结果。解决问题的步骤(动作及动作之间的顺序)称为算法 (algorithm)。

算法表示

- 算法即可以用自然语言表述(如前), 也可用用半结构化语言或结构化图形表示, 如:

```
read 学生成绩值  
if 成绩值 >= 60  
    print "Pass"  
else  
    print "Fail"
```





问题3.1: 计算e值

【问题描述】

e(自然对数)值计算公式为 $1 + 1/1! + 1/2! + \dots + 1/n!$ 。输入一个整数n ($0 \leq n \leq 30$)，计算相应e近似值。

【输入形式】

从键盘输入整数n ($0 \leq n \leq 30$)。

【输出形式】

屏幕输出计算结果，要求小数点后保留10位。

【样例输入1】

12

【样例输出1】

2.7182818283

【样例输入2】

13

【样例输出2】

2.7182818284



问题3.1： 问题分析

- 输入： 一个整数（**整型**） （范围： $0 \leq n \leq 30$ ） ；
- 处理： 计算 公式 $1 + 1/1! + 1/2! + \dots + 1/n!$ ；
- 输出： 以%.10f格式输出(小数后保留10位)
- 数据存储形式（数据结构）考虑：
 - 一个整型变量, 用于存储所读入的整数, 如, `int n`;
 - 一个**双精度**浮点变量(Why?), 用于存储计算结果, 如 `double e`;

问题3.1： 算法设计

■ 解决问题3.1的解题步骤（算法一）：

读入一个整数到变量n;

$e = 0.0;$

for (i=0; i<=n; i++) **$e = 1 + 1/1! + 1/2! + \dots + 1/n!$**

$e = e + 1/i!;$

输出e值;

一般会考虑设置一个函数
`int fact(int n)`来计算n!。
(这会有问题)

问题3.1： 代码实现

```
#include <stdio.h>
int fact(int n);
int main()
{
    int n,i;
    double e = 0.0;
    scanf("%d", &n);
    for(i=0; i<=n; i++)
        e += 1.0/fact(i);
    printf("%.10f", e);
    return 0;
}
int fact(int n)
{
    int i, f=1;
    for(i=1; i<=n; i++)
        f *= i;
    return f;
}
```

为什么要用
1.0/fact(i) ?
而不是
1/fact(i) ?

问题3.1：测试

■ 测试数据的考虑

1. 首先选取输入数据区间 ($0 \leq n \leq 30$) 的正常值（通常就是题目要求中的样例值），如问题3.1中所提供的输入样例；

输入：12

期望（正确）输出：2.7182818283

输入：13

期望（正确）输出：2.7182818284

2. 选取输入数据区间边界附近的值（特殊数据），本例中可选取：

- $n=0$ ，期望输出：1.0000000000

- $n=1$ ，期望输出：2.0000000000

- $n=30$ ，期望输出：2.7182818285（无法事先获知）

问题3.1： 测试结果分析

- 以整型计算 $n!$ ，即函数fact返回整数（观察现象what happened?）

$n=13$

实际输出： 2.7182818288

(期望输出： 2.7182818284)

Why? 如何调试?

- 计算结果为float类型，即float e;（观察现象what happened?）

$n=12$

实际输出： 2.7182819841

(期望输出： 2.7182818283)

数据范围与精度

■ int类型数据范围(对于IA32结构)

$-2^{32-1} \sim 2^{32-1} - 1$ (即-2147483648 ~ 2147483647)

12! < int最大值 < **13!**

■ double类型比float类型有更高精度

● float

➤ 尾数: 23位, 指数: 8位

➤ 表示范围: $-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$

● double

➤ 尾数: 52位, 指数: 11位

➤ 表示范围: $-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$

0!	1
1!	1
2!	2
3!	6
4!	24
5!	120
6!	720
7!	5040
8!	40320
9!	362880
10!	3628800
11!	39916800
12!	479001600
13!	1932053504(error!, 溢出) (正确结果为:6227020800)
14!	1278945280(error!, 溢出) (正确结果为:87178291200)



问题3.1：修改后的代码实现

```
#include <stdio.h>
double fact(int n);
int main()
{
    int n,i;
    double e = 0.0;
    scanf("%d", &n);
    for(i=0; i<=n; i++)
        e += 1.0/fact(i);
    printf("%.10f", e);
    return 0;
}
double fact(int n)
{
    int i;
    double f=1.0;
    for(i=1; i<=n; i++)
        f *= i;
    return f;
}
```



问题3.1： 另一种方法

- 观察计算 公式 $1 + 1/1! + 1/2! + \dots 1/(n-1)! + 1/n!$ ，可知：

若前一次迭代 $1/(n-1)!$ 计算结果为 f_{n-1} ，则本次迭代 $1/n!$ 的结果则为 $f_n = f_{n-1} / n$ 。

因此，没有必要每次迭代都重新计算 $n!$ ，显然程序运行效率会更高。

- 具体算法（解题步骤）为：

```
double e, f;  
e=f=1.0;  
for(i=1; i<=n; i++) {  
    f = f/i;  
    e = e+f;  
}  
输出e;
```



问题3.1：另一种方法（代码）

```
#include <stdio.h>
int main()
{
    int n,i;
    double e ,f
    e = f = 1.0;
    scanf("%d", &n);
    for(i=1; i<=n; i++){
        f = f/i;
        e += f;
    }
    printf("%.10f", e);
    return 0;
}
```

考虑为什么循环从**i=1**开始，而不从**i=0**开始？

问题3.1

- 其它方法?



问题3.1：思考*

- 类似3.1问题的公式很多，如sin, cos, π ...等公式，其解决方法是类似的。

$$e^x = 1 + x/1! + x^2/2! + x^3/3! + \dots + x^n/n!$$

$$\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots + (-1)^{n-1}x^{2n-1}/(2n-1)!$$

$$\cos(x) = 1 - x^2/2! + x^4/4! - x^6/6! + \dots + (-1)^n \times x^{2n}/((2n)!)$$

$$\pi/2 = 1 + 1!/3 + 2!/(3 \times 5) + 3!/(3 \times 5 \times 7) + \dots + (n-1)! / (3 \times 5 \times 7 \times \dots \times (2n-1))$$

...



问题3.2：简易计算器

【问题描述】

编写程序实现简单的交互式计算器，能进行整数的 $+$ $-$ $*$ $/$ 运算。

【输入形式】

从键盘读入如下形式的输入行，数据与运算符之间用空格分隔：

120 + 350

【输出形式】

对于 $+$ 、 $-$ 及 $*$ 运算，输出形式如下：

120+350=470

对于 $/$ 运算，输出形式如下（小数后保留两位）：

5/2=2.50



问题3.2：问题分析

- **输入、输出及数据结构考虑**，从问题描述可知，需要如下变量

int data1, data2, result; /*分别存放计算数据1和2，及+，-，*的计算结果（整型）*/

char op; /*存放运算符*/

float result1; /*存放/计算结果（浮点型）*/

- **如何读入数据及运算符（如：120 + 350）？**

- 方法一

int data1, data2;

char op;

scanf("%d", &data1); /*读入第一个数据*/

getchar(); /*跳过一个空格*/

op = getchar(); /*读入运算符*/

scanf("%d", &data2); /*读入第二个数据*/

不足：数据与运算符之间只能有一个空格分隔

- 方法二

int data1, data2;

char op;

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

好处：数据与运算符之间可以有多个空格分隔

空格使得跳过两次输入之间的所有空白字符。

问题3.2：算法设计

```
int data1,data2,result1;
```

```
float result2;
```

```
char op;
```

从标准输入中读入整数data1，运算符op，整数data2。

判断op:

若为 '+', 则result1 = data1 + data2;

若为 '-', 则result1 = data1 - data2;

若为 '*', 则result1 = data1 * data2;

若为 '/', 则result2 = data1 / data2;

若op为 '+','-'或'*', 输出结果result1;

若op为 '/', 输出结果result2;

多路选择，可使用if-else if语句实现。在此，更适合switch语句。

注意：由于data1和data2为整数，除法运算结果仍为整数。要用强制类型转换才能得到小数位。

result2 = (float)data1/data2;

加运算

减运算

乘运算

除运算

多路选择



问题3.2： 代码实现(if -else if)

```
//c3_2.c
#include <stdio.h>
int main()
{
    int data1,data2, result1;
    float result2;
    char op;
    scanf("%d %c %d", &data1, &op, &data2);
    if ( op=='+' )
        result1 = data1+data2;
    else if ( op=='-' )
        result1 = data1-data2;
    else if ( op=='*' )
        result1 = data1*data2;
    else if ( op=='/' )
        result2 = (float)data1/data2;
    else
        printf("Input error!\n");
    if(op == '+' || op == '-' || op == '*')
        printf("%d%c%d=%d\n", data1,op,data2,result1);
    else if(op == '/')
        printf("%d%c%d=%.2f\n", data1,op,data2,result2);
    return 0;
}
```

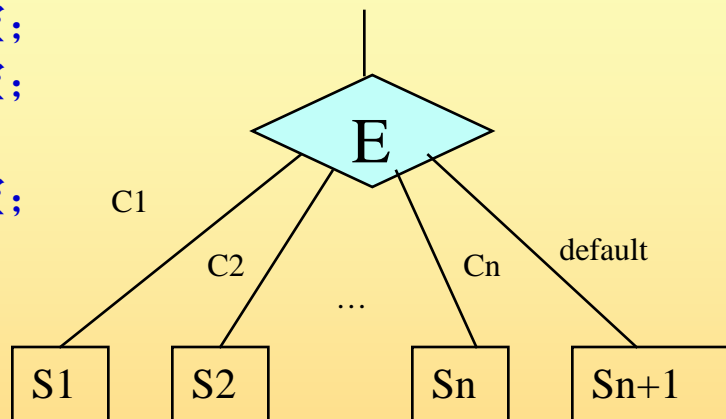
另一种多路选择：switch语句

■ 基本形式：

```
switch (表达式) {  
    case 常量表达式1: 语句1或空;  
    case 常量表达式2: 语句2或空;  
    ...  
    case 常量表达式n: 语句n或空;  
    default: 语句n+1或空;  
}
```

■ 语义动作作为：

- ① 先计算表达式的值；
- ② 该值与每一个case后的常量进行比较；
- ③ 若匹配，则控制就转向该常量后的语句；
- ④ 若不匹配，若有default，则转向default后的语句，否则什么也不做；





多路选择：switch语句(续)

注意：

1. 常量表达式必须是整型（if_else if可能根据任意条件来进行多路选择）；
2. 在同一个switch中不应出现两个具有同样的情况常量；
3. default语句如果有，只允许出现一次，default可出现在switch中的任何位置，通常放在最后；
4. case和default本身不改变控制流（这与pascal中的case语句不同），中断离开switch要用break；
5. case后的语句可以是单个语句，也可以是复合语句（但不带开头和结尾的花括号）

C中switch语句与Pascal中case一个不同是：C有default语句。

因此，switch语句特别适合于依据一组整型常量值来进行判断的多路选择。



问题3.2： 代码实现(switch)

```
//c3_2.c
#include <stdio.h>
int main()
{
    int data1,data2, result1;
    float result2;
    char op;
    scanf("%d %c %d", &data1, &op, &data2);
    switch ( op ) {
        case '+': result1 = data1+data2; break;
        case '-': result1 = data1-data2; break;
        case '*': result1 = data1*data2; break;
        case '/': result2 = (float)data1/data2; break;
        default:    printf("Input error!\n"); break;
    }
    if(op == '+' || op == '-' || op == '*')
        printf("%d%c%d=%d\n", data1,op,data2,result1);
    else if(op == '/')
        printf("%d%c%d=%.2f\n", data1,op,data2,result2);
    return 0;
}
```

测试数据:

120 + 350

12 - 12

35 * 2

3 / 2

123+12

123 + 12

123 & 12

问题3.2：常见问题

■ 在switch中遗漏break

```
//c3_2.c
#include <stdio.h>
int main()
{
    int data1,data2, result1;
    float result2;
    char op;
    scanf("%d %c %d", &data1, &op, &data2);
    switch ( op ) {
        case '+': result1 = data1 + data2;
        case '-': result1 = data1 - data2;
        case '*': result1 = data1 * data2;
        case '/': result1 = data1 / data2;
        default:
    }
    if(op == '+' || op == '-' || op == '*' || op == '/')
        printf("%d\n", result1);
    else if(op == '/')
        printf("%f\n", result2);
    return 0;
}
```

测试数据为:

120 + 350

Why?

统计空白字符、数字字符及其它字符出现次数:

```
switch(c) {
    case ' ':
    case '\t':
    case '\n': nwhite++; break;
    case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9':
        ndigit[c-'0']++; break;
    default: nother++;
}
...
```

```
...
if(c==' ' || c == '\t' || c == '\n')
    nwhite++;
else if ( c >= '0' && c <= '9')
    ndigit[c-'0']++;
else
    nother++;
...
```

switch与if_else if

- switch语句只适合于依据一组整型常量值来进行判断的多路选择。当条件分支较多，且条件判断是根据一组整数值分别判断执行时，通常选择switch语句，如简易计算器例子。
- if_else if可用于依据任意条件（如一个范围）进行判断的多路选择，也就是说，它有更好的适用性。如将百分制成绩转换为五级评分例子。



break和continue语句

- break: 迫使程序从包含它的最内层循环体或开关语句中跳出（循环只能跳出一层）。


- continue: 迫使程序从包含它的最内层循环体立即执行下一次循环。

- goto标号: while(.....)

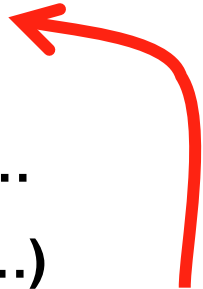
- 标号作

- 只允许层转向

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



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



问题3.2：思考

- 本问题程序一次运行只能进行单个运算符计算。若要使程序能支持多运算符的混合运算，并输入等号(=)结束输入，如， $1+2-3*5-3/2=$ ，如何实现？难点：
 - 如何读入数据及运算符？
 - 如何保证计算时运算符的优先级？
- 更进一步，若要程序支持圆括号来改变计算次序，如， $(1+2-3)*(5-3)/2=$ ，如何实现？



问题3.3：扩展字符

【问题描述】

编写程序将含有缩记符号的字符串扩展为等价的完整字符串，例如将**a-d**扩展为**abcd**。该程序可以处理大小写字母和数字，并可以处理**a-b-c**、**a-z0-9**与**-a-z**等类似的情况。要求扩展符'-'两边的字符只要右边的大于左边就扩展（即**Z-b**情况也要扩展），并且'-'两边不能有空格。

【输入形式】

从键盘输入包含扩展符的字符串，字符串中可以包含空格

【输出形式】

输出扩展后的字符串

【输入样例1】

a-c-u-B

【输出样例1】

abcdefghijklmnopqrstu-B

【输入样例2】

a-b-c a-a 0-4

【输出样例2】

abc a-a 01234

【样例1说明】

扩展输入**a-c-u**为：**abcdefghijklmnopqrstu**，而**B**比**u**值小，所以无法扩展，直接输出。



问题3.3：问题分析

■ 数据结构考虑

由于问题简单，可用两个字符数组来分别存放输入串和展开后字符串，如：

```
char s1[512], s2[512];
```

■ 如何读入一个包含空格的字符串？（scanf函数，为什么？）

● 方法一：

```
char c, s[512];  
for(i=0; (c=getchar()) != '\n'; i++)  
    s[i] = c;  
s[i] = '\0';
```

注意：不要忘记加字符串最后的结束符（\0）！！！！

● 方法二（简单）：使用gets标准库函数 gets(s);



标准输入输出：行输入输出

- 行输入函数：

`char * gets (char s[])`

从标准输入读取完整的一行（以回车结束），将读取的内容存入s字符数组中，并用字符串结束符'\0'取代行尾的'\n'。若读取错误或遇到输入结束则返回NULL。

- 行输出函数

`int puts (char s[])`

将字符数组s中的内容(以'\0'结束)输出到标准输出上，并在末尾添加一个换行符。

问题3.3：算法设计

解决该问题的常见方法是先将要扩展的字符串读到一个字符数组(**s1**)中并设立另一个字符数组用于存放扩展后的字符串(**s2**)，然后依次检查所读入字符串(**s1**)中字符。

当 $s1[i+1] == '-'$ && $s1[i] < s1[i+2]$ 时，
将 $s1[i]$ 至 $s1[i+2]$ 之间的字符展开到 $s2$ 中

否则

将 $s1[i]$ 原封不动的写到 $s2$ 中

下面是该问题的详细算法描述：

```
char s1[512], s2[512]; /*s1用来读入字符串，s2用来存放扩展后字符串*/
```

```
从标准输入中读入字符串到s1中;
```

```
i = j = 0; /* i和j分别为s1和s2字符数组中当前存放字符的位置 */
```

```
while s1[i] != '\0'
```

```
    将s1当前字符放到s2中，即s2[j] = s1[i];
```

```
    if s1[i+1] == '-' && s1[i] < s1[i+2]
```

```
        将s1[i]与s1[i+2]区间字符写到s2中; /*注意不含s1[i+2]字符*/
```

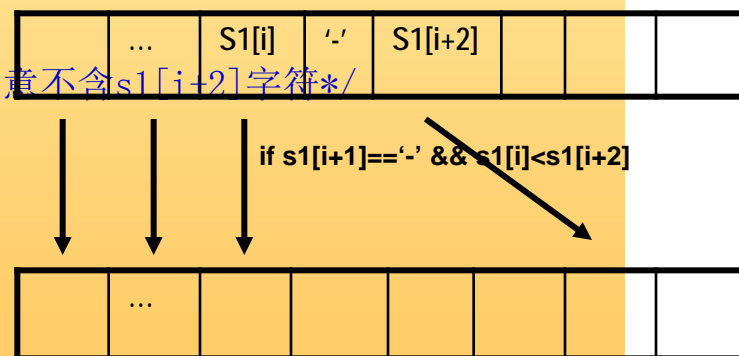
```
        将当前s1读位置移至s1[i+2], 即i = i+2;
```

```
    否则
```

```
        i加1 ;
```

```
给s2置结束符，即s2[j] = '\0' ;
```

```
输出字符串s2;
```





问题3.3：代码实现

```
//c3_3.c
#include <stdio.h>
#define MAXLINE 512
int main()
{
    char c,s1[MAXLINE],s2[MAXLINE];
    int i,j;
    i = j = 0;
    gets(s1);
    while(s1[i] != '\0'){
        s2[j++] = s1[i];
        if(s1[i+1] == '-' && s1[i] < s1[i+2]){
            for(c = s1[i+1]; c < s1[i+2]; c++)
                s2[j++] = c;
            i = i+2;
        }
        else i++;
    }
    s2[j] = '\0';
    puts(s2);
    return 0;
}
```



问题3.3：测试

测试数据	期望结果	数据类型
a-d	abcd	正常
a-c-u-B	abcdefghijklmnopqrstu-B	特殊
a-d d-a	abcd d-a	特殊
Z-b	Z[\]^_`ab	特殊
a-b-c	abc	边界
-a-8	-a-8	非正常
a-a	a-a	非正常



问题3.3：常见问题及分析

■ 处理带空格的输入串

- 不要用scanf来读输入串，建议用gets或getchar来读输入串；

■ 字符串没有结束标志

- 在用getchar读入一个字符串及生成扩展字符串s2时，一定要给字符串置一个结束符(' \0')；
(通过实例演示一下现象)



问题3.3：常见问题及分析（续）

- 用a-d, a-b, a-d-g作输入观察下面程序现象。如何调试？

```
//c3_3a.c
#include <stdio.h>
#define MAXLINE 512
int main()
{
    char c,s1[MAXLINE],s2[MAXLINE];
    int i,j;
    i = j = 0;
    gets(s1);
    while(s1[i] != '\0'){
        s2[j++] = s1[i];
        if(s1[i+1] == '-' && s1[i] < s1[i+2]){
            for(c = s1[i+1]; c <= s1[i+2]; c++)
                s2[j++] = c;
            i = i+3;
        }
        else i++;
    }
    s2[j] = '\0';
    puts(s2);
    return 0;
}
```

问题3.3：思考

工欲善其事，
必先利其器。

- 若要求扩展符 ‘-’ 两边的字符为同类(即均为小写字母、大写字母或数字字符)时才扩展，即出现 Z-b、8-B 这种情况将不扩展，程序如何修改？

算法分析：

其实在扩展字符操作前，在判断

```
s1[i+1] == '-' && s1[i] < s1[i+2]
```

中增加一个判断扩展符两边的字符是否同类的函数即可。

```
if(s1[i+1] == '-' && s1[i] < s1[i+2] && isCongener(s1[i],s1[i+2])){
```

判断两个字符是否是同类的函数实现如下：

```
int isCongener(char c1, char c2)
{
    if(isupper(c1) && isupper(c2))
        return 1;
    if(islower(c1) && islower(c2))
        return 1;
    if(isdigit(c1) && isdigit(c2))
        return 1;
    return 0;
}
```

isupper, islower, isdigit
均为系统标准库函数。使用前要加：
`#include <ctype.h>`

问题3.4：多项式相加

【问题描述】编写一个程序实现两个一元多项式相加。

【输入形式】从标准输入中读入两行以空格分隔的整数，每一行代表一个多项式，且该多项式中各项的系数均为0或正整数，最高幂次不超过50。对于多项式 $a^n x^n + a^{n-1} x^{n-1} + \dots + a^1 x^1 + a^0 x^0$ ($n \leq 50$) 的输入方法如下：
 a^n n a^{n-1} $n-1$...
 a^1 1 a^0 0

即相邻两个整数分别表示表达式中一项的系数和指数。在输入中只出现系数不为0的项。

【输出形式】将运算结果输出到屏幕。将系数不为0的项按指数从高到低的顺序输出，每次输出其系数和指数，均以空格分隔。最后要求换行。

【样例输入】

```
54 8 2 6 7 3 25 1 78 0
43 7 4 2 8 1
```

【样例输出】

```
54 8 43 7 2 6 7 3 4 2 33 1 78 0
```

【样例说明】输入的两行分别代表如下表达式：

$$54x^8 + 2x^6 + 7x^3 + 25x + 78$$

$$43x^7 + 4x^2 + 8x$$

其和为

$$54x^8 + 43x^7 + 2x^6 + 7x^3 + 4x^2 + 33x + 78$$

问题3.4：问题分析

- 如何读取输入的数据？
 - 输入的数据项个数未知（最多不超过 $51 \times 2 = 102$ ）
 - 每行数据输入时没有明显结束标志（但以'\n'结束）
- 如何保存输入的多项式系数和指数（数据结构设计）？

问题3.4：问题分析（续）

■ 如何读取输入的数据？

54 8 2 6 7 3 25 1 78 0

● 方法一：

设变量a, n分别存储读入项和相应项系数；

```
while(1) {
    scanf("%d%d%c", &a,
    保存a和n;
    if(c == '\n')
        break;
```

break语句，用于跳出循环。

```
}
```

或

```
do {
    scanf("%d%d%c", &a, &n, &c);
    保存a和n;
} while(c != '\n');
```

do_while循环，一种在循环尾部进行判断的循环。

● 方法二：

下面方法可按行依次读入

```
gets(buf);
for(i=0;buf[i]!='\0';i++){
    if(b[i]==' '|| buf[i]=='\t') continue;
    for(n=0;buf[i]>='0' &&buf[i]<='9';i++)
        n = n*10+buf[i]-'0';
    保存整数n;
```

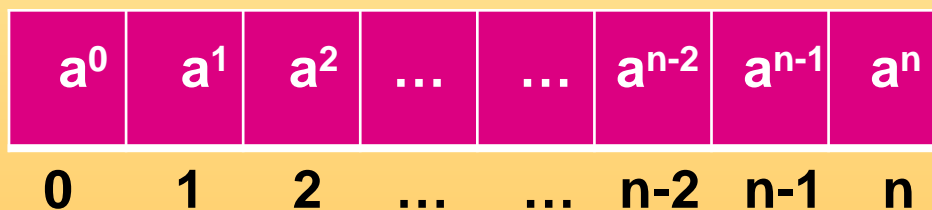
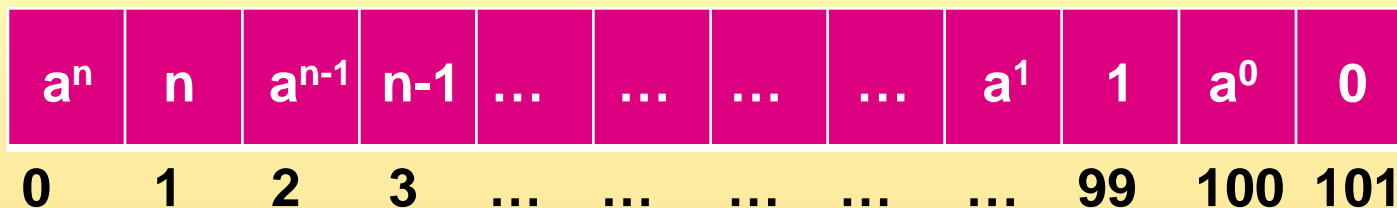
gets(char s[])函数从标准输入中读入一行（以回车结束）。

方法一的缺点是回车之前不能有空格。

问题3.4：问题分析（续）

- 如何保存输入的多项式系数和指数（数据结构设计）？

54 8 2 6 7 3 25 1 78 0



将 a^n 中的 a 保存在下标为 n 的数组元素中，数组长度为51(0~50)，访问系数方式为: `array[n] = a;`



问题3.4： 算法设计

1. `int array1[51], int array2[51];`//分别用项式和第二个多项式;

2. 初始化数组array1和array2元素值均为0;

3. 读入第一行数据保存在数组array1中;

4. 读入第二行数据保存在数组array2中;

5. 依次将数组array1和array2相应元素相加, 并将结果放到array1中, 即:

`array1[i]=array1[i]+array2[i];`

6. 依次从后往前输出数组array1中不为0的元素。

用一个函数来分别读入一个多项式的系数和指数, 如,

`void getExp(int a[];`

问题3.4：代码实现

```
//c3_4.c
```

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

```
#define LENGTH 51
```

```
void getExp(int array[]);
```

```
int main()
```

```
{
```

```
    int array1[LENGTH]={0}, array2[LENGTH]={0}, i;
```

```
    getExp(array1);
```

```
    getExp(array2);
```

```
    for (i=0;i<LENGTH;i++)
```

```
        array1[i] += array2[i];
```

```
    for (i=LENGTH-1;i>=0;i-- )
```

```
        if(array1[i] != 0)
```

```
            printf("%d %d ",array1[i], i);
```

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

```
}
```

初始化数组，元素全为0

从标准输入中读取数据到数组中

两多项式相应项相加

输出多项式系数不为0的项。

```
void getExp(int array[])
```

```
{
```

```
    int a,n;
```

```
    char c;
```

```
    do {
```

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

```
        array[n] = a;
```

```
    } while ( c != '\n');
```



问题3.4：测试

■ 正常数据：

54 8 2 6 7 3 25 1 78 0 ✓

43 7 4 2 8 1 ✓

期望结果：

54 8 43 7 2 6 7 3 4 2 33 1 78 0 ✓

■ 边界数据1：

105 50 23 49 46 25 12 1 57 0 ✓

203 50 22 48 21 25 13 2 9 1 200 0 ✓

期望结果：

308 50 23 49 22 48 67 25 13 2 21 1 257 0 ✓

■ 边界数据2：

12 50 ✓

25 0 ✓

期望结果：

12 50 25 0 ✓

问题3.4：常见问题分析

- 未初始化数组（观察下面用正常测试数据程序运行时现象，如何调试？）

```
//c3_4a.c
#include <stdio.h>
#define LENGTH 51
void getExp(int array[]);
int main()
{
    int array1[LENGTH],array2[LENGTH],i;

    getExp(array1);
    getExp(array2);

    for (i=0;i<LENGTH;i++)
        array1[i] += array2[i];

    for (i=LENGTH-1;i>=0;i-- )
        if(array1[i] != 0)
            printf("%d %d ",array1[i], i);
    printf("\n");
}
```



问题3.4：常见问题分析（续）

- 最后一对数据没有读入，例如：（观察用正常测试数据程序运行时现象，如何调试？）

```
/*c3_4b.c*/
void getExp(int array[ ])
{
    int a,n,c;
    scanf("%d%d",&a,&n);
    while ( (c = getchar()) != '\n' ) {
        array[n]=a;
        scanf("%d%d",&a,&n);
    }
}
```

- 如何修改？

修改方法二：

```
void getExp(int array[ ])
{
    int a,n,c;
    scanf("%d%d",&a,&n);
    array[n] = a;
    while ( (c = getchar()) != '\n' ) {
        scanf("%d%d",&a,&n);
        array[n]=a;
    }

    scanf("%d%d",&a,&n);
    while ( (c = getchar()) != '\n' ) {
        array[n]=a;
        scanf("%d%d",&a,&n);
    }
    array[n] = a;
}
```

问题3.4：思考

■ 当前数据结构有何优点和不足？

- 优点：算法简单
- 不足：有空置，如：输入多项式 $12x^{50}$

0	0	0	0	0	0	12
0	1	2	3	48	49	50

■ 其它数据结构及算法？

- 考查如下多项式存储方式，如何实现（算法）？

54 8 2 6 7 3 25 1 78 0

54	8	2	6	7	3	25	1	78	0	...	0
----	---	---	---	---	---	----	---	----	---	-----	---

- 链表，特别适合指数大小没有限定的情况（以后介绍）。

问题3.4：思考

- 如何实现任意（多项式项数、最高幂未知的）多项式相加？
- 问题3.4实现了两个多项式相加运算，如何实现两个多项式相乘？



问题3.5：超长正整数加法

【问题描述】

编写程序实现两个超长正整数（每个最长80位数字）的加法运算。

【输入形式】

从键盘读入两个整数，不考虑输入高位可能为0的情况。

1. 第一行是超长正整数A;
2. 第二行是超长正整数B;

【输出形式】

输出只有一行，是两个长整数的运算结果，从高到低依次输出各位数字。
各位数字紧密输出。

【输入样例】

```
134098703578230056
234098
```

【输出样例】

```
134098703578464154
```

【样例说明】

进行两个正整数加法运算， $134098703578230056 + 234098 = 134098703578464154$ 。

问题3.5：问题分析

- 如何读入和存储超长整数？（为何不能用长整数类型long int n; scanf("%ld",&n);来存储和读入超长整数？）

- 方法一：用字符串方式来读入和存储超长整数

```
char intstr[81];  
scanf("%s", intstr);
```

- 方法二：用整数数组来存储超长整数，用字符方式依次读入超长整数的每位数字

```
int lint[80];  
char d,i=0;  
while((d=getchar())!= '\n')  
    lint[i++] = d - '0';
```

- 其它方法？

问题3.5：解题思路

- 在程序中数组中数据是从左至右（从高位到低位）方式存储的，而整数相加是从右至左（从低位到高位），同时由于被加数长短不一，造成计算和转换非常不方便。一种解决方法是整数相加前将**两个整数位串首尾颠倒**，与计算机存储方式一致。
- 如何进行超长整数加？

无论以什么方式存储超长整数，每位相加结果和进位计算方式为：

$\text{digit}_i = (\text{digit1}_i + \text{digit2}_i + \text{carry}) \% 10$ （carry为上一次计算产生的进位）

$\text{carry} = (\text{digit1}_i + \text{digit2}_i + \text{carry}) / 10$ （得到新的进位）

注意：

- 若以字符串方式存储超长整数，则在计算每位加和进位时，应考虑数字字符和整数数字之间的转换。
- 应考虑进位传递问题
 - 当较短整数最后一位加完后仍有进位情况，如123456789+678；
 - 当较长整数最后一位处理完后仍有进位情况，如9999999+1；



问题3.5：算法设计

- 基于上述分析，假设以字符串形式存储整数，其超长整数相加算法如下：

```
char istr1[81], istr2[81]; int carry, sum; /*car
```

```
分别读入字符串istr1和istr2; //假设istr1中存放
```

```
将istr1和istr2串首尾颠倒;
```

```
While istr2[i] != '\0'
```

```
    sum = istr1[i] - '0' + istr2[i] - '0' + carry
```

```
    istr1[i] = sum % 10 + '0' ;
```

```
    carry = sum / 10;
```

```
    i++;
```

```
While istr1[i] != '\0' && carry
```

```
    sum = istr1[i] - '0' + carry;
```

```
    istr1[i] = sum % 10 + '0' ;
```

```
    carry = sum / 10;
```

```
    i++
```

```
If carry > 0
```

```
    istr1[i++] = carry + '0' ;
```

```
    istr1[i] = '\0' ;
```

```
将istr1串首尾颠倒;
```

以较短整数为基准，两个整数相加。

考虑较短整数最后一位加完后仍有进位产生。

考虑极端情况较长整数最后一位处理完后仍有进位。如999999999+1

问题3.5：代码实现

```
void addLInt(char s1[], char s2[])
{
    int i=0,tmp,c=0;
    char s[LENGTH];
    if(strlen(s1) < strlen(s2)){ /* 交换字符串，确保字符串
        strcpy(s, s1);
        strcpy(s1,s2);
        strcpy(s2,s);
    }
    reverse(s1); reverse(s2); /* 颠倒字符串 */
    while(s2[i] != '\0'){ /*较短的依次与较长的相加 */
        tmp = s1[i]-'0' + s2[i]-'0' + c;
        s1[i] = tmp%10 + '0';
        c = tmp/10;
        i++;
    }
    while(s1[i] != '\0' && c){ /* 较短的加完后，若有进位，
        tmp = s1[i]-'0' + c;
        s1[i] = tmp%10 + '0';
        c = tmp/10;
        i++;
    }
    if(c) /* 判断最后是否还有进位 */
        s1[i++] = c + '0';
    s1[i] = '\0';
    reverse(s1);
}
```

```
/* 主函数 */
#include <stdio.h>
#include <string.h>
#define LENGTH 81
void addLInt(char s1[], char s2[]);
void reverse(char s[]);
int main()
{
    char intstr1[LENGTH],intstr2[LENGTH];
    scanf("%s %s",intstr1, intstr2);
    addLInt(intstr1, intstr2);
    printf("%s", intstr1);
    return 0;
}
```

```
void reverse(char s[])
{
    int i,j,c;;
    for(i=0,j=strlen(s)-1; i<j; i++,j--){
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

常用标准字符串处理库函数

使用**strcpy**、**strcat**函数之前，必须保证**s**有足够的空间容纳操作后的字符串！

■ #include <string.h>

`int strlen(char s[]);` /*计算字符串长度, 字符串以\0结果*/

`char *strcpy(char s[], char t[]);` /*将字符串t拷贝到字符串s中*/

`char *strcat(char s[], char t[]);` /*将字符串t拷贝到字符串s尾部*/

`int strcmp(char s[], char t[]);` /*比较两个字符串,若s>t,则返回大于0的数;若s<t,则返回小于0的数;若相等, 返回0 */

子曰：工欲善其事，必先利其器。...

■ 正常数据，如：

- 134098703578230056
234098
- 234098
134098703578230056

■ 边界数据，如：

- [illegible]

1

问题3.5：思考

- 考虑其它解决方法？
- 在此基础上进一步考虑如何实现超长整数相减、相乘及相除？

本讲结束