计算机导论与程序设计 [CS006001-60]

段江涛

机电工程学院

西安电子科技大学 XIDIAN UNIVERSITY

2019 年 12 月

lecture-1 主要内容

1. 课程介绍
2. 导论简介
3. C 语言程序设计
4. 开发工具
5. Algorithm + Data Structures = Programs
6. 初识 C 语言程序 (作业)

课程内容

- 计算机导论：了解计算机的基本知识；掌握计算机操作基本技能。
- 程序设计：掌握结构化程序设计方法。会读、会编、会调试 C 语言程序。
- 教材
  - 大学计算机，龚尚福，贾澎涛，西安电子科技大学出版社
  - C 程序设计第五版，谭浩强，清华大学出版社

考核

1. 导论部分计算机应用成绩 (C1): 10%。计算机基本操作技能。
2. 导论部分课程报告成绩 (C2): 10%。撰写课程学习小论文。
3. 单元测验成绩 (C3、C4): 40%。根据机试系统给出的练习题目编写程序，通过调试得到正确结果并通过**机试系统提交**。
4. 平时作业成绩 (C5): 10%。主要考核对每堂课知识点的复习、理解和掌握程度。
5. 期末考试成绩 (C6): 30%。主要考程序设计思想、逻辑思维、程序设计方法、程序调试能力。**考试形式为机试**。

计算机导论主要内容

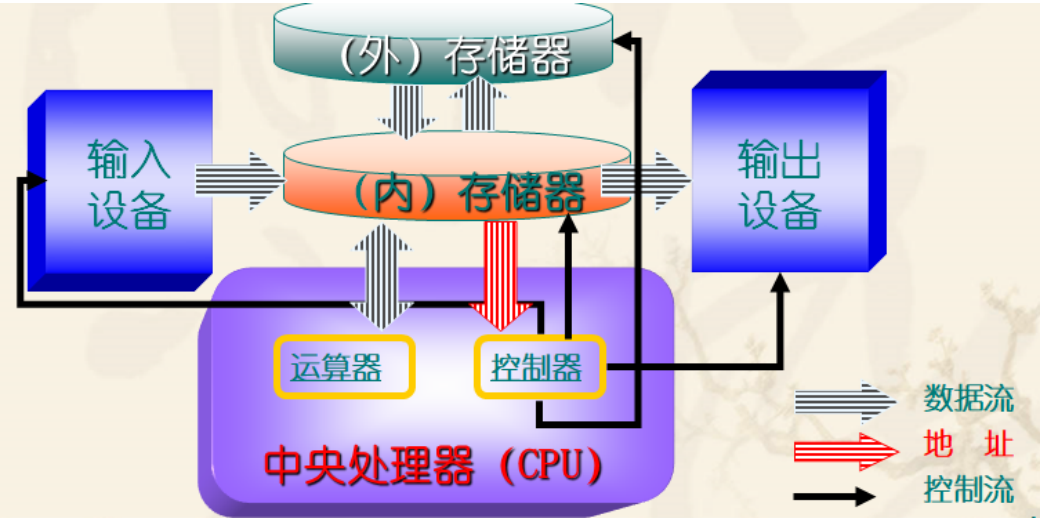
总体要求: 了解计算机的基本知识; 掌握计算机操作基本技能。

- 计算机系统组成
- 计算机工作原理
- 操作系统
- 字处理: Microsoft Word
- 电子表格: Microsoft Excel
- 演示文稿: Microsoft PowerPoint

计算机工作原理

工作原理: “存储程序” + “程序控制”

1. 以二**进制**形式表示数据和指令
2. 将程序存入存储器中, 由控制器自动读取并执行
3. 外部存储器存储的程序和所需数据 $\Longrightarrow$ 计算机内存 $\Longrightarrow$ 在程序控制下由 CPU 周而复始地取出指令、分析指令、执行指令 $\Longrightarrow$ 操作完成。



10 进制,2 进制,16 进制的幂展开式

$$
\begin{aligned}
(D)_{10} &= D_{n-1} \times 10^{n-1} + D_{n-2} \times 10^{n-2} + \cdots + D_1 \times 10^1 + D_0 \times 10^0 \\
&+ D_{-1} \times 10^{-1} + D_{-2} \times 10^{-2} + \cdots + D_{-m+1} \times 10^{-m+1} + D_{-m} \times 10^{-m} \\
(B)_2 &= B_{n-1} \times 2^{n-1} + B_{n-2} \times 2^{n-2} + \cdots + B_1 \times 2^1 + B_0 \times 2^0 \\
&+ B_{-1} \times 2^{-1} + B_{-2} \times 2^{-2} + \cdots + B_{-m+1} \times 2^{-m+1} + B_{-m} \times 2^{-m} \\
(H)_2 &= H_{n-1} \times 16^{n-1} + H_{n-2} \times 16^{n-2} + \cdots + H_1 \times 16^1 + H_0 \times 16^0 \\
&+ H_{-1} \times 16^{-1} + H_{-2} \times 16^{-2} + \cdots + 16_{-m+1} \times 16^{-m+1} + H_{-m} \times 16^{-m}
\end{aligned}
$$

进制对照表

二进制	十六进制	二进制	十六进制
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Example

$$(123)_{10} = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0;$$

$$3 = 123\%10, 2 = 123/10\%10, 1 = 123/10/10\%10$$

$$(77)_{10} = (0100 \quad 1101)_2 = 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4$$

$$+ 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$(77)_{10} = (4D)_{16} = 4 \times 16^1 + 13 \times 16^0$$

数值在计算机中的表示 (以 8bit 编码为例)

- 原码：正数的符号为 0，负数的符号为 1，其它位按一般的方法表示数的绝对值。

$$x = (+103)_{10} \qquad [x]_{原} = (01100111)_2$$

$$x = (-103)_{10} \qquad [x]_{原} = (11100111)_2$$

- 反码: 正数的反码与原码相同；负数的反码是符号位不变，其他位按位取反
- 补码: 正数的补码与其原码相同；负数的补码为其反码最末位加 1. 即，补码 = 反码 +1

$$(77)_{10} = (0100 \quad 1101)_2, \qquad (-77)_{10} = (1100 \quad 1101)_2$$

$$(-77)_{补} = 2^8 - 77 = 1111 \quad 1111 - 0100 \quad 1101 + 0000 \quad 0001$$

$$= 1011 \quad 0010 + 0000 \quad 0001 = 1011 \quad 0011$$

机内以补码形式存储有符号数

1. 对于正数,原码 = 反码 = 补码
2. 对于负数,补码 = 反码 + 1
   反码 = 符号位不变,其他位按位取反
3. 补码是可逆的,即再对补码求补得到原码。
4. 引入补码后,使减法统一为加法。

数值表示示例

+77
原码
0 1 0 0 1 1 0 1
反码
0 1 0 0 1 1 0 1
补码
0 1 0 0 1 1 0 1
−77
原码
1 1 0 0 1 1 0 1
反码
1 0 1 1 0 0 1 0
补码
1 0 1 1 0 0 1 1

ASCII 编码表 $B_6B_5B_4B_3B_2B_1B_0$

$B_6B_5B_4$ / $B_3B_2B_1B_0$	000	001	010	011	100	101	110	111
0000	NUL	DLE	空格	0	@	P	、	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	•	>	N	^	n	～
1111	SI	US	/	?	O	_	o	DEL

- ASCII 码连续排列 ‘0’～‘9’, ‘A’～‘Z’, ‘a’～‘z’
- 数字 = 编码值 - ‘0’ 9=‘9’-‘0’
- 大小字符间隔: ‘a’ - ‘A’ = 32

计算机程序

指令

可以被计算机理解并执行的基本操作命令。

程序

一组计算机能识别和执行的指令。一个特定的指令序列用来完成一定的功能。

软件

与计算机系统操作有关的计算机程序、规程、规则，以及可能有的文件、文档及数据。

计算机语言

机器语言

计算机能直接识别和接受的二进制代码称为**机器指令**。机器指令的集合就是该计算机的**机器语言**。

特点：难学，难记，难检查，难修改，难以推广使用。依赖具体机器难以移植。

```
B8  7F 01
BB  21 02
03   D8
B8  1F 04
2B  C3
```

汇编语言

机器语言的符号化。用英文字母和数字表示指令的**符号语言**。

特点：相比机器语言简单好记，但仍然难以普及。汇编指令需通过**汇编程序**转换为机器指令才能被计算机执行。依赖具体机器难以移植。

```
MOV  AX  383
MOV  BX  545
ADD   BX  AX
MOV  AX  1055
SUB    AX  BX
```

高级语言

高级语言更接近于人们习惯使用的自然语言和数学语言。

特点：功能强大，不依赖于具体机器。用高级语言编写的**源程序**需要通过**编译程序**转换为机器指令的**目标程序**。

```
int  x =1055, y = 383, z = 545
int S;
S = x-(y+z);
S=1055-(383+545)
```

高级语言的发展

结构化语言

规定:

程序必须由具有良好特性的基本结构(**顺序结构、选择结构、循环结构**)构成，程序中的流程不允许随意跳转，程序总是由上而下顺序执行各个基本结构。

特点:

程序结构清晰，易于编写、阅读和维护。

C 语言的特点

1. 语言简洁、紧凑，使用方便、灵活
2. 运算符丰富
3. 数据类型丰富
4. C 语言是完全模块化和结构化的语言

   具有结构化的控制语句 (顺序、选择、循环结构)

   用函数作为程序的模块单位，便于实现程序的模块化
5. 兼具高级语言和低级语言的功能

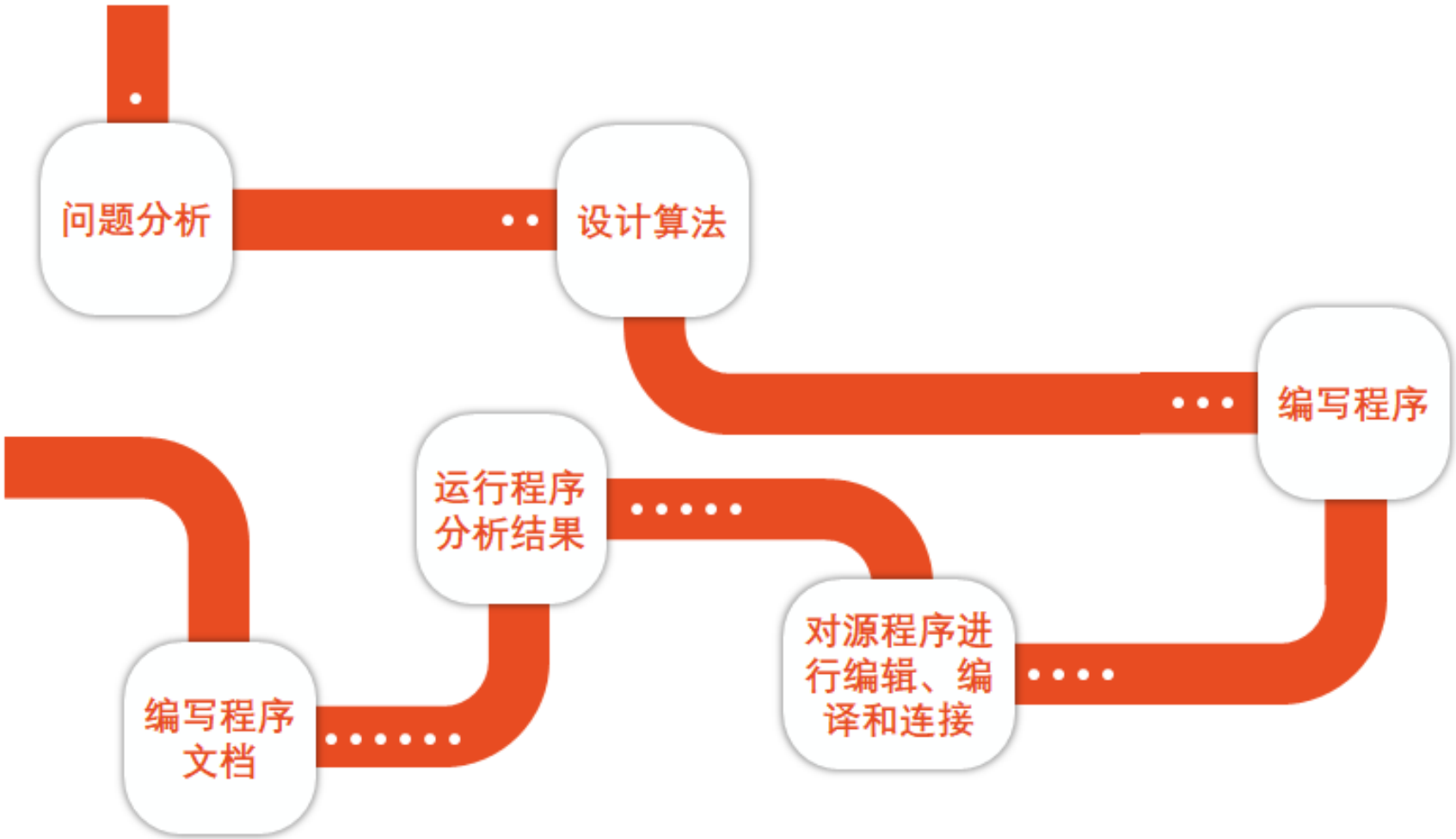
   允许直接访问物理地址

   能进行位 (bit) 操作

   能实现汇编语言的大部分功能

   可以直接对硬件进行操作

程序设计的任务



第一个 C 语言程序

```
#include<stdio.h>                // standard input/output编译预处理指令
int main()                       // 主函数
{                                // 函数开始标志
    printf("Hello␣World!");      // 输出一行信息
    return 0;                    // 函数执行完毕返回函数值0
}                                // 函数结束标志
```

求两个整数之和

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a,b,sum;                   // 定义a,b,sum为整型变量
    a=123;                         // 对a,b赋值
    b=456;
    sum=a+b;                       // 计算a+b, 并把结果存放在变量sum中
    printf("sum␣is␣%d\n",sum);     // 输出结果
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

lecture-2 算法—程序的灵魂

1 课程介绍

2 导论简介

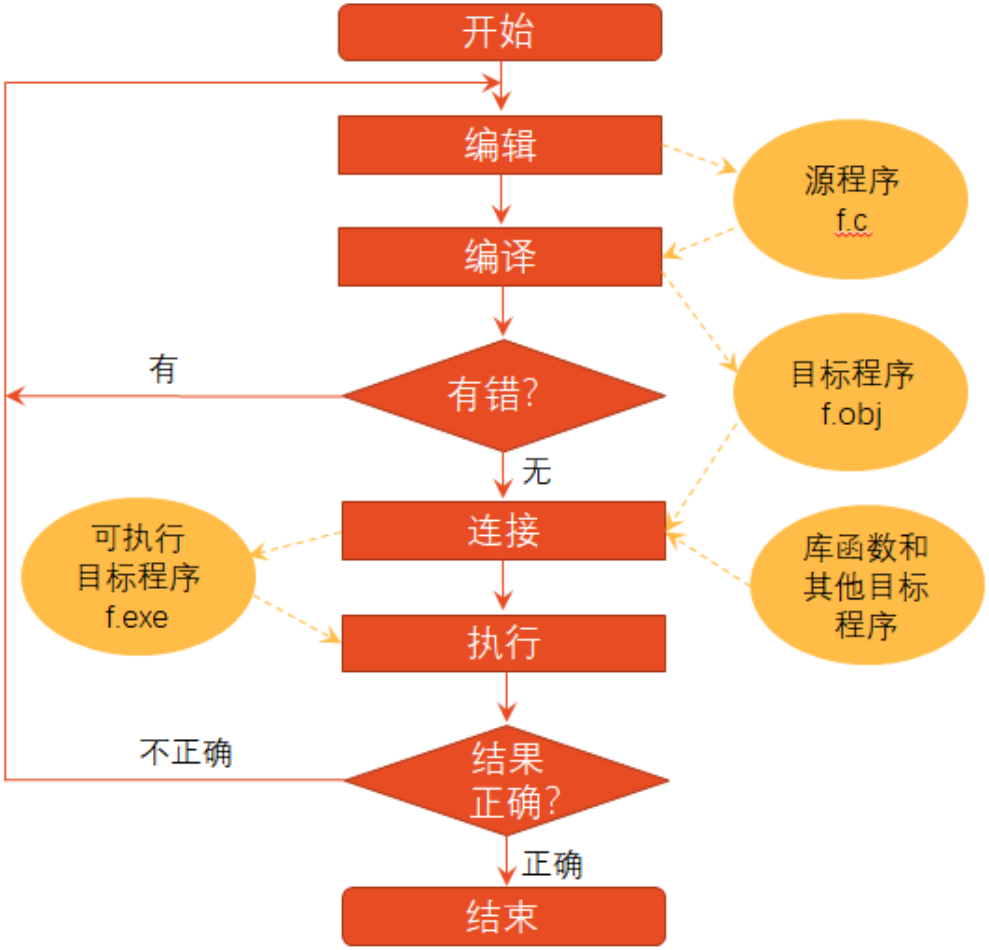
3 C 语言程序设计

4 开发工具

5 Algorithm + Data Structures = Programs

6 初识 C 语言程序 (作业)

运行 C 程序的步骤与方法



集成开发环境—编译系统

- Bloodshed Dev-C++
- Turbo C
- Visual C++6.0
- Visual Studio(VS2015 , VS Community 2019 等)

Bloodshed Dev-C++ 集成开发环境

D:\百度云同步盘\备课\2018\计算机导论与程序设计\C语言课件\程序讲解\第1章\c1-1.c - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 64-bit Release

(globals)

Project Classes Debug

c1-1.c

```
1  #include <stdio.h>                    // 这是编译预处
2  int main( )                           // 定义主函数
3  {                                     // 函数开始的标
4   printf ("This is a C program.\n");   // 输出所指定的
5   return 0;                            // 函数执行完毕
6  }
7
```

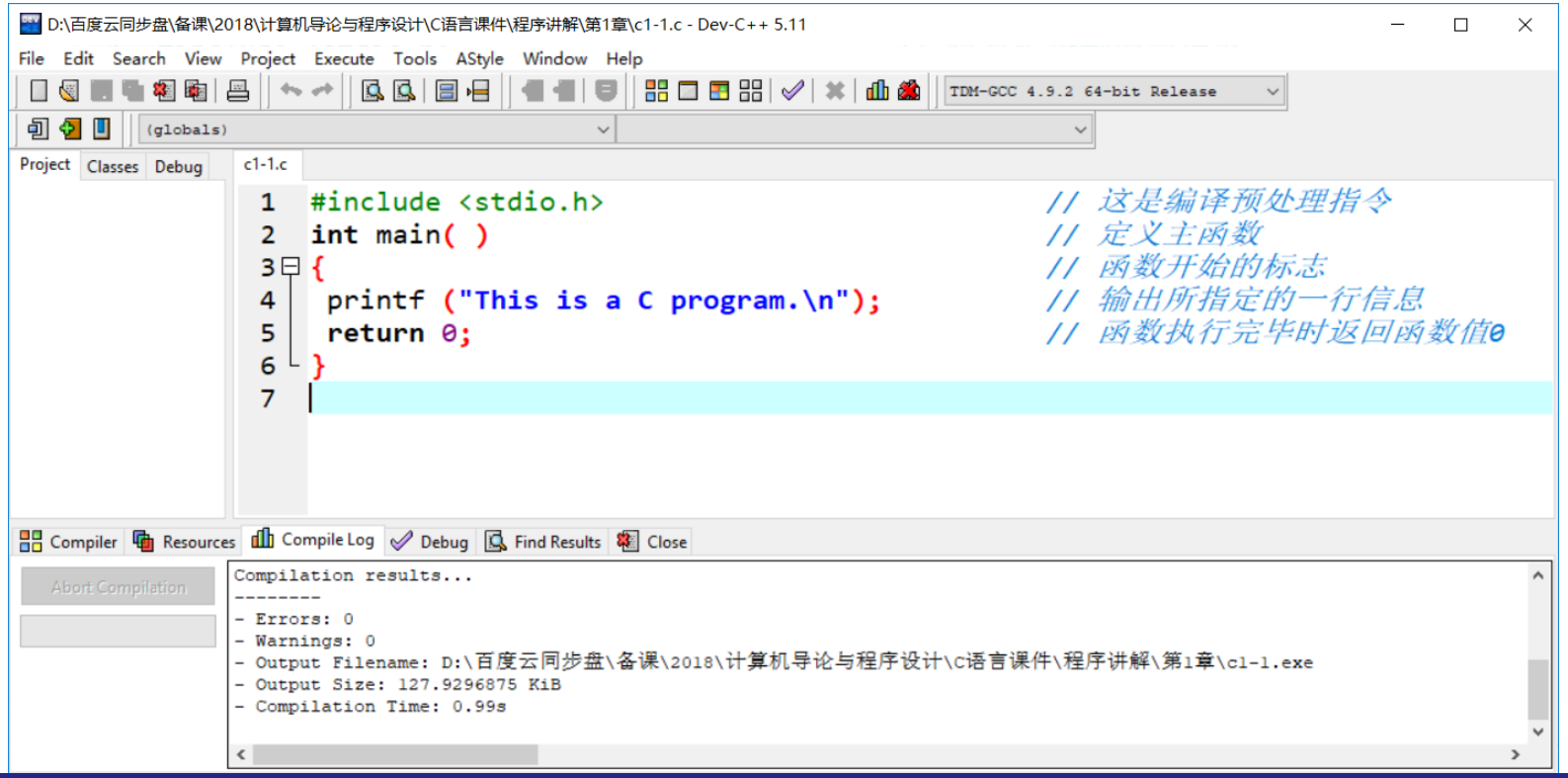
Compiler Resources Compile Log Debug Find Results Close

Abort Compilation

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: D:\百度云同步盘\备课\2018\计算机导论与程序设计\C语言课件\程序讲解\第1章\c1-1.e
- Output Size: 127.9296875 KiB
- Compilation Time: 0.99s
```

Bloodshed Dev-C++ 集成开发环境

- 选择“文件”菜单, 选择“源文件”, 编辑程序。
- 保存时,保存为.cpp 或.c 文件。
- 选择“编译和运行”菜单,生成.exe 文件,运行程序。



数据结构与算法

算法 + 数据结构 = 程序

Algorithm + Data Structures = Programs

沃思 (Niklaus Wirth)

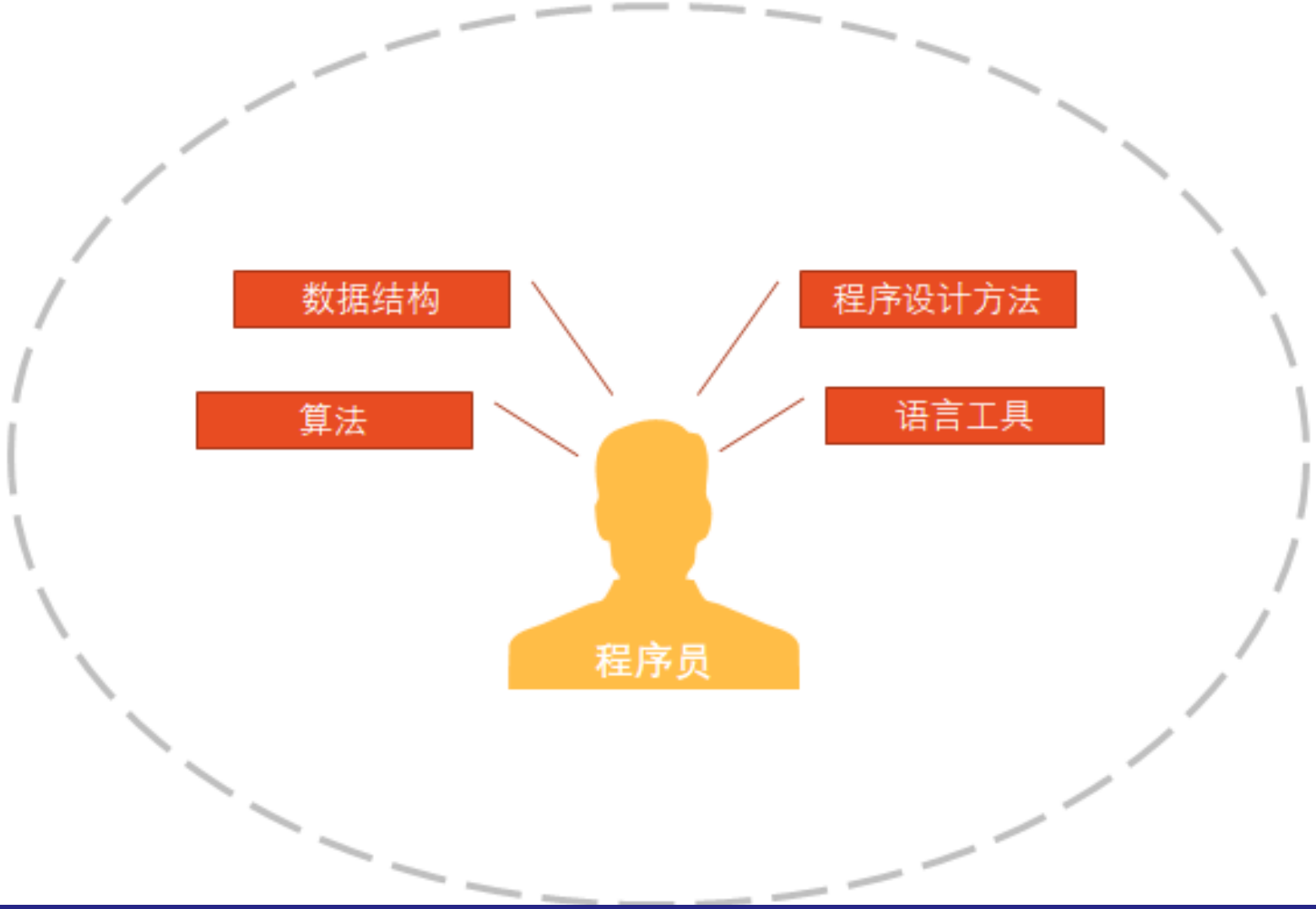
- 数据结构

  对数据的描述。在程序中要指定用到哪些数据,以及这些数据的类型和数据的组织形式。

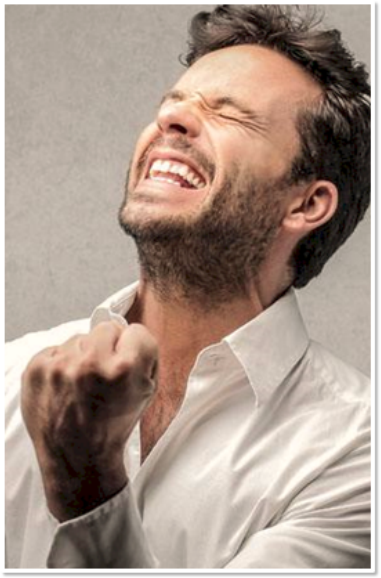
- 算法

  对操作的描述。即要求计算机进行操作的步骤

程序员的工作



算法



算法

- 广义地说，为解决一个问题而采取的方法和步骤，就称为“算法”。
- 对同一个问题，可以有不同的解题方法和步骤。
- 为了有效地进行解题，不仅需要保证算法正确，还要考虑算法的质量，选择合适的算法。

算法

数值运算算法

如求一个方程的根，计算一个函数的定积分等。

数值运算的目的是求数值解。

由于数值运算往往有现成的模型，可以运用数值分析方法，因此对数值运算的算法的研究比较深入，算法比较成熟。

非数值运算算法

如图书检索，人事管理等。

计算机在非数值运算方面的应用远超在数值运算方面的应用。非数值运算的种类繁多，要求各异，需要使用者参考已有的类似算法，重新设计解决特定问题的专门算法。

简单的算法举例 [例 2.1(p17)]

Example (例 2.1(p17))

求 $1 \times 2 \times 3 \times 4 \times 5$

算法 (一) 步骤

- S1 先求 1 乘以 2，得到结果 2
- S2 将步骤 1 得到的乘积 2 再乘以 3，得到结果 6
- S3 将 6 再乘以 4，得 24
- S4 将 24 再乘以 5，得 120

思考

求 $1 \times 3 \times 5 \times 7 \times 9$

算法 (二) 步骤

- S1 $p = 1$, 表示将 1 存放在变量 p 中
- S2 $i = 2$, 表示将 2 存放在变量 i 中
- S3 $p = p * i$, 使 p 与 i 相乘，乘积仍放在变量 p 中
- S4 $i = i + 1$, 使变量 i 的值加 1
- S5 if (i<=5) goto S3
  else 算法结束，最后得到 p 的值就是 5! 的值。

简单的算法举例 [例 2.2(p18)]

Example (例 2.2(p18))

有 50 个学生,要求输出成绩在 80 分以上的学生的学号和成绩.

算法步骤

```
float g[50]={100,90.5,30.8,· · · }; \\ 表示 50 名学生成绩
int i = 0; \\ 表示第 i 个学生学号
while(i<50)
{
    printf(" 第%d 个学生成绩%d, " , i+1, g[i]);
    i = i + 1;
}
```

算法 1 例 2.3(p18): 判定 2000—2500 年中的每一年是否为闰年.

```
int year=2000, char R;
while (year<=2500) do
    R='N';
    if (year 能被 4 整除, 但是不能被 100 整除) then R='Y';
    else if (year 能被 100 整除, 并且能被 400 整除) then R='Y';
    end if
    if (R=='Y') then printf("%d 是闰年", year);
    else    printf("%d 不是闰年", year);
    end if
    year = year + 1;
end while
```

算法 2 例 2.4(p19): 求 $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots + \frac{1}{99} - \frac{1}{100}$.

```
int sign=1, deno=2;
float sum = 1.0;
while (deno<=100) do
    sign = -1 * sign;
    sum = sum + sign*1.0/deno;
    deno = deno + 1;
end while
printf("sum=%f\n", sum);
```

sign: 表示当前项的数值符号

deno: 表示当前项的分母

sum: 表示当前项的累加和

算法 3 例 2.5(p20): 给出一个大于或等于 3 的正整数, 判断它是不是一个素数.

```
int n, i=2;
scanf("%d",&n); \\ 输入 n 的值;
while (i < n) do
    if (n 能被 i 整除) then { printf("%d 不是素数", n); return; }
    end if
    i = i + 1 ;
end while
printf("%d 是素数", n);
```

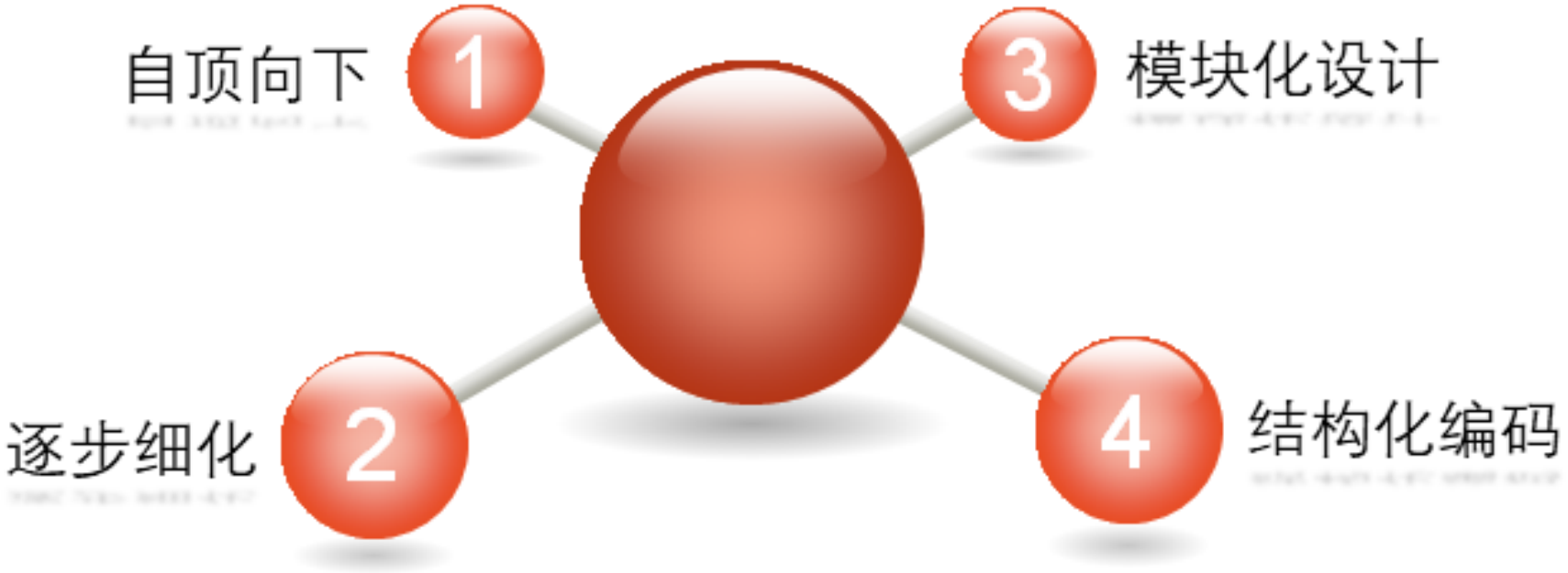
Notes

实际上, n 不必检查被 $2 \sim (n-1)$ 之间的整数整除, 只须检查能否被 $2 \sim \sqrt{2}$ 间的整数整除即可。

算法的特性

1. **有穷性**
   一个算法应包含有限的操作步骤，而不能是无限的
2. **确定性**
   算法中的每一个步骤都应当是确定的，而不应当是含糊的、模棱两可的
3. **有零个或多个输入**
   所谓输入是指在执行算法时需要从外界取得必要的信息
4. **有一个或多个输出**
   算法的目的是为了求解，“解”就是输出
5. **有效性**
   算法中的每一个步骤都应当能有效地执行，并得到确定的结果

结构化程序设计方法



求 5! 的 C 语言程序。[作业: 请抄写以下各页,并试着分析理解。]

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int i,p;  // p表示被乘数, i表示乘数
    p=1;
    i=2;
    while(i<=5)
    {
        p=p*i;
        i++; // i = i + 1
    }
    printf("%d\n",p);
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

变量在使用之前首先要定义它的数据类型

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a,b;   // 定义变量a, b为整型数值, 同类型变量可以在一条语句中定义。
    float f;   // 定义变量f为单精度浮点数
    double d;  // 定义变量d为双精度浮点数
    char c;    // 定义变量c为单个英文字母
    a=10;
    b=20;
    f=10.2;
    d=20.3;
    c='A';
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

常用格式描述符与数据类型的对应关系

格式符	对应的数据类型	备注
%d	int	
%f	float	
%c	char	
%lf	double	
%.2f	float	保留两位小数, 四舍五入。不适用于 scanf()。
%.2lf	double	保留两位小数, 四舍五入。不适用于 scanf()。
%x	int	十六进制显示
%ld	long int	

详见 p73, 表 3.6

输出语句 printf(“原样输出, % 格式符”, 对应变量值);

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a=10,b;     // 定义变量a, b为整型数值, 定义变量时, 可以指定变量的初值
    float f=10.2;   // 定义变量f为单精度浮点数
    double d; // 定义变量d为双精度浮点数
    char c;    // 定义变量c为单个英文字母
    f=10.2;
    d=20.3;
    c='A';
    printf("a=%d,b=%d,c=%c,f=%f,d=%lf\n",a,b,c,f,d); // \n为换行符
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

输入语句 scanf(“% 变量格式符”, & 变量名);

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a=10,b;     // 定义变量a, b为整型数值, 定义变量时, 可以指定变量的初值
    float f=10.2;   // 定义变量f为单精度浮点数
    double d; // 定义变量d为双精度浮点数
    char c='A';    // 定义变量c为单个英文字母, 字符输入以后讲
    printf("请输入整数a,b,␣空格隔开:\n"); // 提示语句[可选]
    scanf("%d%d",&a,&b);
    printf("请输入浮点数f,d,␣空格隔开:\n"); // 提示语句[可选]
    scanf("%f%lf",&f,&d);
    printf("a=%d,b=%d,c=%c,f=%f,d=%lf\n",a,b,c,f,d); // \n为换行符
    return 0;                         // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

if(条件表达式){ 表达式为真 (非 0) 时执行语句; }

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a=10;     // 定义变量a为整型数值，定义变量时，可以指定变量的初值
    if(a>=10)
    {
        printf("a>=10\n"); // \n为换行符
    }
    else
    {
        printf("a<10\n"); // \n为换行符
    }
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

while(条件表达式){ 表达式为真 (非 0) 时执行的语句;}

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a=10;     // 定义变量a为整型数值，定义变量时，可以指定变量的初值
    while(a>=0)
    {
      printf("a=%d\n",a); // \n为换行符
      a--; // a= a - 1
    }
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

lecture-3 主要内容

最简单的 C 语言程序设计—顺序程序设计

1 课程介绍

2 导论简介

3 C 语言程序设计

4 开发工具

5 Algorithm + Data Structures = Programs

6 初识 C 语言程序 (作业)

DevC++ 5.0 以前的版本

```
#include<stdio.h>                    // standard input/output编译预处理指令
#include <stdlib.h>                  // standard library function, eg. for
    system()函数
int main()                           // 主函数
{                                    // 函数开始标志
   printf("%d\n",p);
   system( "pause" );                // 窗口暂停, DevC++ 5.0以前的版本。[机试系统提
    交时, 一定注释或删除该语句]
   return 0;                         // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

求 5! 的 C 语言程序

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int i,p;  // p表示被乘数, i表示乘数
    p=1;
    i=2;
    while(i<=5)
    {
        p=p*i;
        i++; // i = i + 1
    }
    printf("%d\n",p);
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

变量在使用之前首先要定义它的数据类型

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a,b;   // 定义变量a, b为整型数值, 同类型变量可以在一条语句中定义。
    float f;   // 定义变量f为单精度浮点数
    double d;  // 定义变量d为双精度浮点数
    char c;    // 定义变量c为单个英文字母
    a=10;
    b=20;
    f=10.2;
    d=20.3;
    c='A';
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

if(条件表达式){ 表达式为真 (非 0) 时执行语句; }

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a=10;     // 定义变量a为整型数值，定义变量时，可以指定变量的初值
    if(a>=10)
    {
        printf("a>=10\n"); // \n为换行符
    }
    else
    {
        printf("a<10\n"); // \n为换行符
    }
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

while(条件表达式){ 表达式为真 (非 0) 时执行的语句;}

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a=10;     // 定义变量a为整型数值，定义变量时，可以指定变量的初值
    while(a>=0)
    {
      printf("a=%d\n",a); // \n为换行符
      a--; // a= a - 1
    }
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

常用格式描述符与数据类型的对应关系

格式符	对应的数据类型	备注
%d	int	
%f	float	
%c	char	
%lf	double	
%.2f	float	保留两位小数, 四舍五入。不适用于 scanf()。
%.2lf	double	保留两位小数, 四舍五入。不适用于 scanf()。
%x	int	十六进制显示
%ld	long int	

详见 p73, 表 3.6

输出语句 printf("原样输出, % 格式符", 对应变量值);

```
#include<stdio.h>                // standard input/output编译预处理指令
int main()                       // 主函数
{                                // 函数开始标志
   int a=10,b;     // 定义变量a, b为整型数值, 定义变量时, 可以指定变量的初值
   float f=10.2;   // 定义变量f为单精度浮点数
   double d; // 定义变量d为双精度浮点数
   char c;    // 定义变量c为单个英文字母
   f=10.2;
   d=20.356;
   c='A';
   printf("a=%d,b=%d,c=%c,f=%f,d=%.2lf\n",a,b,c,f,d); // %.2f, %.2lf保留
    两位小数
   return 0;                     // 函数执行完毕返回函数值0
}                                // 函数结束标志
```

变量 b 没有被赋值, 将是一个随机值。

输入语句 scanf("% 变量格式符", & 变量名);

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a=10,b;     // 定义变量a, b为整型数值, 定义变量时, 可以指定变量的初值
    float f=10.2;   // 定义变量f为单精度浮点数
    double d; // 定义变量d为双精度浮点数
    char c='A';    // 定义变量c为单个英文字母, 字符输入以后讲
    printf("请输入整数和浮点数,␣空格隔开:\n"); // 提示语句[可选]
    scanf("%d%f",&a,&f);  // 尽量简单, 不要有其它字符和'\n'
    printf("请输入两个浮点数,␣空格隔开:\n"); // 提示语句[可选]
    scanf("%f%lf",&f,&d);
    printf("a=%d,b=%d,c=%c,f=%f,d=%lf\n",a,b,c,f,d); // \n为换行符
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

字符输出函数 putchar

```
#include<stdio.h>
int main()
{
    char a = 'B',b = 'O',c = 'Y'; //定义3个字符变量并初始化
    putchar(a); //向显示器输出字符B
    putchar(b); //向显示器输出字符O
    putchar(c); //向显示器输出字符Y
    putchar ('\n'); //向显示器输出一个换行符
    return 0;
}
```

字符输入函数 getchar, 遇到回车, 开始从缓冲区中接收字符。

```
#include<stdio.h>
int main()
{
    char a,b,c;   //定义字符变量a,b,c
    a = getchar();   //从键盘输入一个字符，送给字符变量a
    b = getchar();   //从键盘输入一个字符，送给字符变量b
    c = getchar();   //从键盘输入一个字符，送给字符变量c
    putchar(a);   //将变量a的值输出
    putchar(b);   //将变量b的值输出
    putchar(c);   //将变量c的值输出
    printf("\na=%d,b=%d,c=%d,a=%c,b=%c,c=%c\n",a,b,c,a,b,c);
    return 0;
}
```

字符输入函数 getchar, 遇到回车, 开始从缓冲区中接收字符。

```
char a,b,c;  //定义字符变量a,b,c
a = getchar();  //从键盘输入一个字符，送给字符变量a
b = getchar();  //从键盘输入一个字符，送给字符变量b
c = getchar();  //从键盘输入一个字符，送给字符变量c
putchar(a);  //将变量a的值输出
putchar(b);  //将变量b的值输出
putchar(c);  //将变量c的值输出
printf("\na=%d,b=%d,c=%d,a=%c,b=%c,c=%c\n",a,b,c,a,b,c);
```

从键盘输入 abc 回车, 观察结果, 应该是正确的结果。遇到回车, 开始从缓冲区中接收字符。

字符输入函数 getchar, 遇到回车, 开始从缓冲区中接收字符。

```
a = getchar();  //从键盘输入一个字符，送给字符变量a
b = getchar();  //从键盘输入一个字符，送给字符变量\n
c = getchar();  //从键盘输入一个字符，送给字符变量b
putchar(a);  //将变量a的值输出a
putchar(b);  //将变量b的值输出\n
putchar(c);  //将变量c的值输出b
printf("\na=%d,b=%d,c=%d,a=%c,b=%c,c=%c\n",a,b,c,a,b,c);
```

再运行一次程序, 输入 a 回车, 输入 b 回车, 输入 c 回车, 观察结果。

```
a
b
a
b
a=97, b=10, c=98, a=a, b=
, c=b
```

根据学生的反馈,数据的输入语句没有完全听明白。

下一讲, 进一步详细讲解。

欢迎同学们在群里踊跃发言,任何不理解的知识点请指出来,我将在以后的讲课中,有针对性的讲清楚大家有疑惑的问题。一些与编程无关的俏皮话之类的东西就不要发了,争取把我们这个群建立成纯净的,对大家学习课程有帮助的程序设计讨论群。力争 100 名学生一个都不掉队,考个好成绩。加油!!

lecture-4 主要内容

最简单的 C 语言程序设计—顺序程序设计

1 课程介绍

2 导论简介

3 C 语言程序设计

4 开发工具

5 Algorithm + Data Structures = Programs

6 初识 C 语言程序 (作业)

数据输入再讨论

在开发平台,以具体的示例,详细讲解以下内容:

- %d, %f, %c, %lf 格式符的使用
- char c; scanf("%c", &c); 接收输入的字符
- char c; c=getchar() 接收输入的字符
- 避免数字,字符在一条语句中输入的情况, 如:
  scanf("%d%c%d",...);
- 重点理解字符缓冲区的概念,以及消费无用字符的技巧。

lecture-5 主要内容

最简单的 C 语言程序设计—顺序程序设计

1. 课程介绍
2. 导论简介
3. C 语言程序设计
4. 开发工具
5. Algorithm + Data Structures = Programs
6. 初识 C 语言程序 (作业)

常量

```
#include<stdio.h>
#define PI 3.14  // 符号常量, 注意没有分号
int main()
{
   int a = 123; // 整型常量
   float f = 12.2, f1=123E-1; // 实型常量
   char c1 = 'A', c2='\n'; // 字符常量
   char s[50] = "boy"; // 字符串常量
   printf("半径为%d的圆周长是%f\n",a,2*PI*a);
   printf("回车换行\n");
   printf("单引号\',双引号\"转义字符前缀\,\n");
   return 0;
}
```

转义字符, 见 p40, 表 3.1

常变量

```
#include<stdio.h>
#define PI 3.14  // 符号常量，注意没有分号
int main()
{
   int r = 123; // 整型变量
   const int a = 425; // 常变量
   r = 100; // 合法，因为r是变量，可以随时更改它的值
   a = 100; // 不合法，因为a是常变量，不能更改
   printf("半径为%d的圆周长是%f\n",r,2*PI*r);
   return 0;
}
```

标识符

标识符就是一个对象的名字。用于标识变量、符号常量、函数、数组、类型等。

以字母或下划线开始; 区分大小写; 不能使用关键字; 最好有含义。

```
#include<stdio.h>
int main()
{
    int r = 123; // 合法整型变量名
    int 3a; // 不合法的变量名
    int brake; // 不合法的变量名，因为break是关键字，被系统使用。
    int Radius; // 变量名最好有含义
    int radius; // 与Radius是不同的变量，C语言是到小写敏感的语言
    return 0;
}
```

C 语言关键字

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

整型、浮点型数据类型

不同的类型分配不同的长度和存储形式。整型数据存储空间和值的范围见 p45, 表 3.2

```
#include<stdio.h>
int main()
{
    int a = 123; // 整型变量
    long int b = 1E+8; // 长整型变量
    unsigned int u = 0XFF; // 无符号整型, 最高为不作为符号位处理
    float f = 10.2; // 单精度浮点数
    double d = 1E-8; // 双精度浮点数
    printf("%x,%d\n",u,u); // ff, 255
    printf("%d,%ld,%x,%f,%lf\n",a,b,u,f,d);
    return 0;
}
```

字符类型 (ASCII 字符表见附录 A)

```
#include<stdio.h>
int main()
{
    char c1 = 'A', c2 = 'a', c3 = '\n'; // 字符型变量
    printf("%c,%c,%d\n",c1,c2,c3); // A,a,10
    // 整型变量的整数值就是ASCII编码值
    printf("%c,0X%x,%d\n",c1,c1,c1); // A,0X41,65
    c1 = c1 + 1; // 在表达式中, char类型看作int处理
    printf("%c,0X%x,%d\n",c1,c1,c1); // B,0X42,66
    c1 = c1 + 32;  // 转换为小写字母
    printf("%c,0X%x,%d\n",c1,c1,c1); // b,0X62,98
    printf("%d\n",'9'-'0'); // 数字 = 编码值- '0'
    printf("%c,%d,%c,%d\n','A','A','a','a'); // 输出字符和相应的ASIII编码
    return 0;
}
```

算术运算符 +, −, *, /, %, ++, −−

整数 = 整数/整数, 结果不会四舍五入。

```
#include<stdio.h>
int main()
{
    int a=5, b=2;  float c=5,d=2,f;
    f = a/b; printf("%f\n",f); // 2.000000
    f = c/d; printf("%f\n",f); // 2.500000
    f = (float)a/b; printf("%f\n",f); // 2.500000
    printf("%f\n",5.0/2); // 2.500000
    printf("%d\n",2a); // 错误
    printf("%d\n",2*a); // 正确
    return 0;
}
```

余数 r=a%b, a,b 必须是整数。

```
#include<stdio.h>
int main()
{
    int a = 123;
    while(a != 0) // 当a不等于0时, 执行循环体。
    {
        printf("%d␣",a%10); // 3 2 1
        a = a/10;  // 或 a /= 10;
    }
    printf("\n");
    return 0;
}
```

算术运算符 ++, −−

++i, - -i: 先加 (减)1, 再使用。i++, i- -: 先使用, 再加 (减)1

```
#include<stdio.h>
int main()
{
    int a,b=10;
    a = ++b;
    printf("a=%d,b=%d\n",a,b); // a=11,b=11
    a = b++;
    printf("a=%d,b=%d\n",a,b); // a=11,b=12
    a--;
    b--;
    printf("a=%d,b=%d\n",a,b); // a=10,b=11
    return 0;
}
```

数学库函数

数学库函数，详见附录 E p365.

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

int abs(int x);	求整数 x 的绝对值
double fabs(double x);	求浮点数 x 的绝对值
double sqrt(double x);	计算 $\sqrt{x}$
double pow(double x, double y);	计算 x^y
int rand(void);	产生-90 32767 的随机整数
double log(double x);	求 $\log_e x$, 即 $\ln x$
double log10(double x);	求 $\log_{10} x$

Example (例 3.5 p64)

求 $ax^2 + bx + c = 0$ 方程的根。a, b, c 由键盘输入，设 $b^2 - 4ac > 0$。

```
#include<stdio.h>
#include<math.h>    // 数学库函数
int main()
{
    double a,b,c,x1,x2,delta;
    scanf("%lf%lf%lf",&a,&b,&c);
    if(b*b-4*a*c <= 0) { printf("输入错误!"); return 0; } // 程序结束
    delta = sqrt(b*b-4*a*c);
    x1 = -b + delta/(2*a);
    x2 = -b - delta/(2*a);
    printf("x1=%lf,x2=%lf\n",x1,x2);
    return 0;
}
```

Example (例 3.1 p37)

有人用温度计测量出用华氏法表示的温度 (如 64°F), 今要求把它转换为以摄氏法表示的温度 (如 17.8°C)。

$$c = \frac{5}{9}(f - 32)$$

其中,f 代表华氏温度, c 代表摄氏温度

Example (例 3.2 p38)

计算存款利息。有 1000 元,想存一年。有 3 种方法可选:

(1) 活期,年利率为 r1;

(2) 一年期定期,年利率为 r2;

(3) 存两次半年定期,年利率为 r3。

请分别计算出一年后按 3 种方法所得到的本息和。

$$p1 = p_0(1 + r_1), p2 = p_0(1 + r_2), p3 = p_0(1 + \frac{r_3}{2})(1 + \frac{r_3}{2})$$

Example

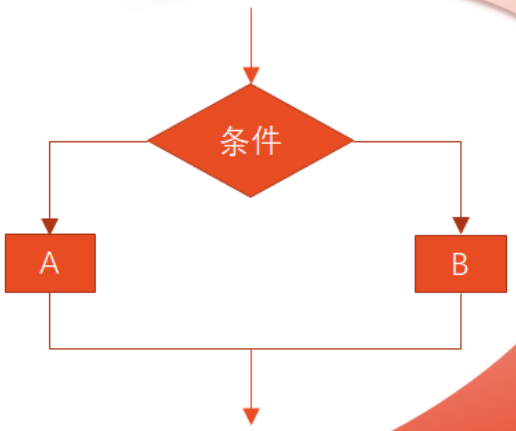
1. 求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$
2. 求 $1\times2\times3\times4\times5$
3. 自定义各变量类型和值, 求 $y=|x^3+\log_{10}x|$
4. 自定义各变量类型和值, 求 $y=\frac{3ae}{cd}$
5. 自定义各变量类型和值, 求 $y=\frac{ax+\frac{a+x}{4a}}{2}$
6. m 是一个已知 3 位整数, 从左到右用 a,b,c 表示各位数字, 求数 bac 的值。

lecture-6 主要内容

选择结构程序设计

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- while(表达式){···}
- do{···}while(表达式);
- for(表达式 1; 表达式 2; 表达式 3){···}
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6}, {7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
- 顺序举例
- 使用函数进行模块化程序设计
- 实参和形参间的数据传递 (值传递和地址传递)
- 顺序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 使用 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例例

选择结构和条件判断



C 语言有两种选择语句

- if 语句, 用来实现**两个分支**的选择结构
- switch 语句, 用来实现**多分支**的选择结构

if(条件表达式){ 表达式为真 (非 0) 时执行语句; }

```
#include<stdio.h>                    // standard input/output编译预处理指令
int main()                           // 主函数
{                                    // 函数开始标志
    int a=10;     // 定义变量a为整型数值，定义变量时，可以指定变量的初值
    if(a>=10)
    {
        printf("a>=10\n"); // \n为换行符
    }
    else
    {
        printf("a<10\n"); // \n为换行符
    }
    return 0;                        // 函数执行完毕返回函数值0
}                                    // 函数结束标志
```

[例 4.1 p84] 求 $ax^2 + bx + c = 0$ 方程的根。a, b, c 由键盘输入。

```
#include<stdio.h>
#include<math.h>    // 数学库函数
int main()
{
   double a,b,c,x1,x2,delta;
   scanf("%lf%lf%lf",&a,&b,&c);
   if(b*b-4*a*c < 0)
   { printf("This␣equation␣hasn\'t␣real␣roots!\n"); }
   else
   {
      delta = sqrt(b*b-4*a*c);
      x1 = -b + delta/(2*a);  x2 = -b - delta/(2*a);
      printf("x1=%.2lf,x2=%.2lf\n",x1,x2);
   }
   return 0;
}
```

[例 4.2 p85] 输入两个实数, 按由小到大的顺序输出这两个数。

```
#include<stdio.h>
int main()
{
    float a,b,t;
    scanf("%f%f",&a,&b);
    //不好: scanf("%f,%f",&a,&b);
    if(a>b)
    {   //将a和b的值互换
        t=a;
        a=b;
        b=t;
    }
    printf("%.2f,%.2f\n",a,b);
    return 0;
}
```

两个变量值的互换

a=b; //把变量 b 的值赋给变量 a, a 的值等于 b 的值

b=a; //再把变量 a 的值赋给变量 b, 变量 b 值没有改变

因此, 为了实现互换, 必须借助于第三个变量。

[例 4.3 p86] 输入 3 个数 a, b, c, 要求按由小到大的顺序输出。

```
#include<stdio.h>
int main()
{
    float a,b,c,t;
    scanf("%f%f%f",&a,&b,&c); //不好: scanf("%f,%f,%f",&a,&b,&c);
    if(a>b)
    {
       t=a; a=b; b=t; //借助变量t, 实现变量a和变量b互换值
    } //互换后, a小于或等于b
    if(a>c)
    {
       t=a; a=c; c=t; //借助变量t, 实现变量a和变量c互换值
    } //互换后, a小于或等于c
    if(b>c) //还要
    {
       t=b; b=c; c=t; //借助变量t, 实现变量b和变量c互换值
    }  //互换后, b小于或等于c
    printf("%.2f,%.2f,%.2f\n",a,b,c); //顺序输出a,b,c的值
    return 0;
}
```

if(条件表达式){ 表达式为真 (非 0) 时执行语句; }

条件表达式: 关系表达式; 逻辑表达式; 数值表达式。

形式 1(无 else)

```
// 形式1(无else)
if(条件表达式)
{
    多条语句(复合语句);
}
```

形式 2

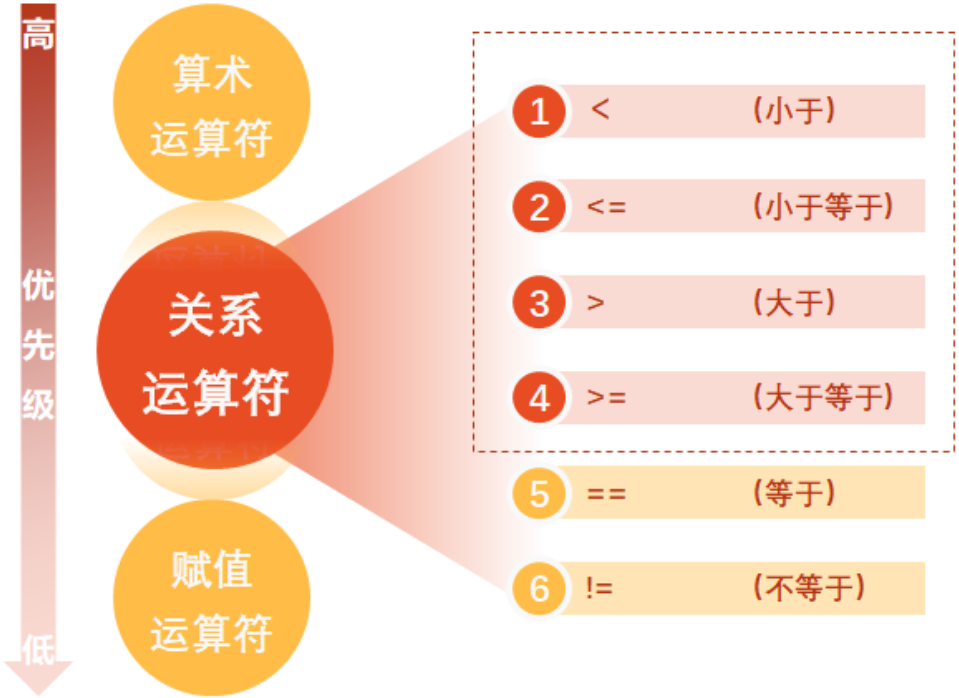
```
// 形式2
if(条件表达式)
{
    多条语句(复合语句);
}
else
{
    多条语句(复合语句);
}
```

形式 3(排除式)

```
// 形式3(排除式)
if(条件表达式1)
{
    多条语句(复合语句);
}
else if(条件表达式2)//可多个
{
    多条语句(复合语句);
}
else
{
    多条语句(复合语句);
}
```

关系运算符及其优先次序

```
int a=5,b=10,c=20; //以int为例
if(a<b+c) // 相当于a<(b+c)
{ ... }
if(a<=b+c)
{ ... }
if(a>b+c)
{ ... }
if(a>=b+c)
{ ... }
if(a==b+c)//a是否等于(b+c),与a=(b+c)
    不同
{ ... }
if(a!=b+c) // a不等于(b+c)
{ ... }
```



分析:

```
if(a>b==c){...}
if(a=b>c) {...}
```

关系表达式的值, 非 0 即真

关系表达式

- 用关系运算符将两个数值或数值表达式连接起来的式子,称为关系表达式。
- 关系表达式的值是一个逻辑值,即 “真” 或 “假”。
- 在 C 的逻辑运算中, 以 “1” 代表 “真”,以 “0” 代表 “假”。

```
int a=3, b=2, c=1, d1, d2;
d1 = a > b; // d1=1
d2 = a > b > c; //自左至右结合,d2=0
if(d1)
{ printf("执行此语句") }
if(d2)
{ printf("不执行此语句"); }
```

```
if(d1 = a > b) // d1的值就是表达式的值
{ printf("执行此语句"); }
if(d2 = a > b > c)//d2的值就是表达式的
    值
{ printf("不执行此语句"); }
```

逻辑运算符

```
int a=5,b=10,c=0; //以int为例
if(!a) // 逻辑非(NOT), a是非0, 所以!a的值是0
{ ... }
if(a && b) // 逻辑与(AND), a,b均为非0, 所以(a && b)的值为1
{ ... }
if(a || c) // 逻辑或(OR), a,c之一是非0, 即为真
{ ... }
```

逻辑运算符真值表

a	b	!a	!b	a&&b	a||b
真 (非 0)	真 (非 0)	假 (0)	假 (0)	真 (1)	真 (1)
真 (非 0)	假 (0)	假 (0)	真 (1)	假 (0)	真 (1)
假 (0)	真 (非 0)	真 (1)	假 (0)	假 (0)	真 (1)
假 (0)	假 (0)	真 (1)	真 (1)	假 (0)	假 (0)

- “&&” 和 “‖” 是双目运算符，要求有两个运算对象 (操作数);
  “!” 是单目运算符，只要有一个运算对象
- 由高到低优先次序: !(非)→&&(与)→‖(或);
  逻辑运算符中的 “&&” 和 “||” 低于关系运算符，“!” 高于算术运算符
- 逻辑运算结果不是 0 就是 1，不可能是其他数值。
  而运算对象可以是 0(假) 或任何非 0 的数值 (按 “真” 对待)

逻辑运算示例 (1)

判别用 year 表示的某一年是否闰年，可以用一个逻辑表达式来表示。闰年的条件是符合下面二者之一: (1) 能被 4 整除，但不能被 100 整除。(2) 能被 100 整除，又能被 400 整除。

```
int year;
scanf("%d",&year);
\\ 闰年
if(year%4 == 0 && year%100 != 0)
{ printf("%d是闰年\n", year); }
else if(year%100 == 0 && year%400 == 0)
{ printf("%d是闰年\n", year); }
else
{ printf("%d不是闰年\n", year)}
```

逻辑运算示例 (2)

判别用 year 表示的某一年是否闰年,可以用一个逻辑表达式来表示。闰年的条件是符合下面二者之一: (1) 能被 4 整除,但不能被 100 整除。(2)能被 100 整除,又能被 400 整除。

```
int year, flag = 'N';
scanf("%d",&year);
\\ 闰年
if(year%4 == 0 && year%100 != 0)
{ flag = 'Y'; }
else
{
   if(year%100 == 0 && year%400 == 0)
   { flag = 'Y'; }
}
if(flag == 'Y')
{ printf("%d是闰年\n", year); }
else
{ printf("%d不是闰年\n", year); }
```

逻辑运算示例 (3)

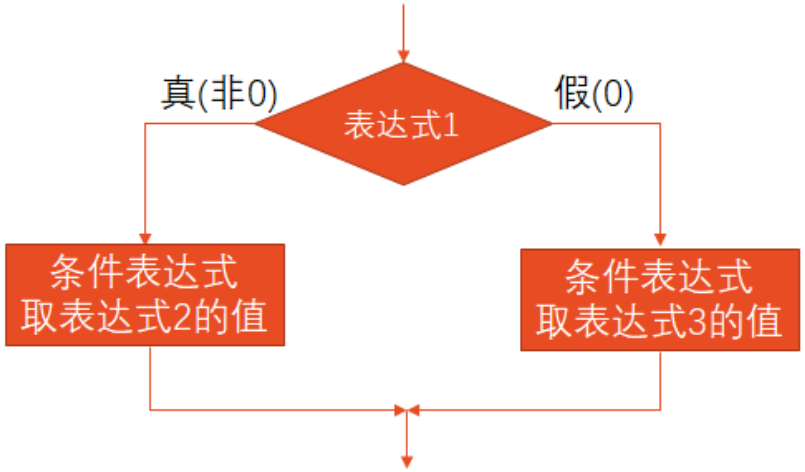
判别用 year 表示的某一年是否闰年,可以用一个逻辑表达式来表示。闰年的条件是符合下面二者之一: (1) 能被 4 整除,但不能被 100 整除。(2)能被 100 整除,又能被 400 整除。

```
int year;
scanf("%d",&year);
\\ 闰年
if((year%4 == 0 && year%100 != 0)||(year%100 == 0 && year%400 == 0))
{ printf("%d是闰年\n", year); }
else
{ printf("%d不是闰年\n", year)}
```

条件运算符和条件表达式

```
int a,b,max;
scanf("%d%d",&a,&b);
if(a>b)
{ max = a; }
else
{ max = b; }
// 等效为
max = (a>b) ? a : b;
// 或
a>b ? (max=a) : (max=b);
// 甚至用在语句中
printf("%d\n",a>b ? a : b);
```

表达式 1 ? 表达式 2 : 表达式 3



例: 大写转小写字母

[例 4.4, p96] 输入一个字符, 判别它是否为大写字母, 如果是, 将它转换成小写字母; 如果不是, 不转换。然后输出最后得到的字符。

```
char ch;
scanf("%c",&ch);
ch = (ch>='A' && ch<='Z') ? (ch+32) : ch;
// 等效于
if(ch>='A' && ch<='Z')
{
    ch = ch+32; // 可简写为 ch += 32;
}
printf("ch=%c\n",ch);
```

数学表达式与 C 语言表达式的不同

```
int a = 100;
if(20 <= a && a <= 30) // 表达式的值为假(0), 条件表达式与数学含义相同
{ ...  }
if(20 <= a <= 30) // (20<=a)<=30, 表达式为真(1), 条件表达式与数学含义不同
{ ...  }
// 类似的
// if(a==20) 与 if(a=20)意义不同
if(a==20) //表达式的值是假(0), a的值没有变化
{ ... }
if(a=20) //表达式的值是10, 非0, 表示为真, 并且a被赋值为20(赋值语句)
{
    printf("%d\n",a); // 20
}
printf("%d\n",a); // 20
```

用 switch 语句实现多分支选择结构

switch(int 或 char 型表达式)

```
int a;
scanf("%d",&a)
switch(a)
{
    case 10: 多条语句1;
             break;
    case 20: 多条语句2;
             break;
    case 30: 多条语句3;
             break;
    default: 多条语句4;
}
```

$\Longrightarrow$

```
int a;
scanf("%d",&a)
if(a == 10)
{ 多条语句1; }
else if(a == 20 )
{ 多条语句2; }
else if(a == 30)
{ 多条语句3; }
else
{ 多条语句4; }
```

```c
char a; // 或 int a;
scanf("%c",&a)
swach(a)
{
    case 'A':
    case 'a': 多条语句1;
              break;
    case 'B':
    case 'b': 多条语句2;
              break;
    case 'C':
    case 'c': 多条语句3;
              break;
    default: 多条语句4;
}
```

```c
char a; // 或 int a;
scanf("%d",&a)
if(a == 'A' || a == 'a')
{ 多条语句1; }
else if(a == 'B' || a == 'b')
{ 多条语句2; }
else if(a == 'C' || a == 'c')
{ 多条语句3; }
else
{ 多条语句4; }
```

[例 4.10,p99] 运输公司对用户计算运输费用。路程越远,运费越低。标准如下:

$s < 250$	没有折扣
$250 \le s < 500$	2% 折扣
$500 \le s < 1000$	5% 折扣
$1000 \le s < 2000$	8% 折扣
$2000 \le s < 3000$	10% 折扣
$3000 \le s$	15% 折扣

```
int c,s; //c是分类整数, s是距离
float p,w,d,f; //单价,重量,折扣,运费
// 运费 f = p*w*s*(1-d%)
scanf("%f%f%d",&p,&w,&s);
if(s>=3000) { c = 12; } else {c = s/250; }
switch(c) {
   case 0: d=0; break;
   case 1: d=2; break;
   case 2: case 3: d=5; break;
   case 4: case 5: case 6: case 7:
        d=8; break;
   case 8: case 9: case 10: case 11:
        d=10 break;
   case 12: d=15; break;
}
f = p*w*s*(1-d/100);  printf("%.2f\n",f);
```

lecture-7 主要内容

循环结构程序设计

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- while(表达式){···}
- do{···}while(表达式);
- for(表达式 1; 表达式 2; 表达式 3){···}
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6}, {7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
- 顺序举例
- 使用函数进行模块化程序设计
- 实参和形参间的数据传递 (值传递和地址传递)
- 顺序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 使用 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例例

为什么需要循环控制

- 要向计算机输入全班 50 个学生的成绩;(重复 50 次相同的输入操作)
- 分别统计全班 50 个学生的平均成绩; (重复 50 次相同的计算操作)

```
float score1,score2,score3,score4,score5,aver; // 5门课成绩及平均成绩
// 输入第1个学生5门课的成绩
scanf("%f%f%f%f%f",&score1,&score2,&score3,&score4,&score5);
// 求第1个学生平均成绩
aver=(score1+score2+score3+score4+score5)/5;
printf("aver=%7.2f",aver); // 输出第1个学生平均成绩
// 输入第2个学生5门课的成绩
scanf("%f%f%f%f%f",&score1,&score2,&score3,&score4,&score5);
// 求第2个学生平均成绩
aver=(score1+score2+score3+score4+score5)/5;
printf("aver=%7.2f",aver);// 输出第2个学生平均成绩
...
```

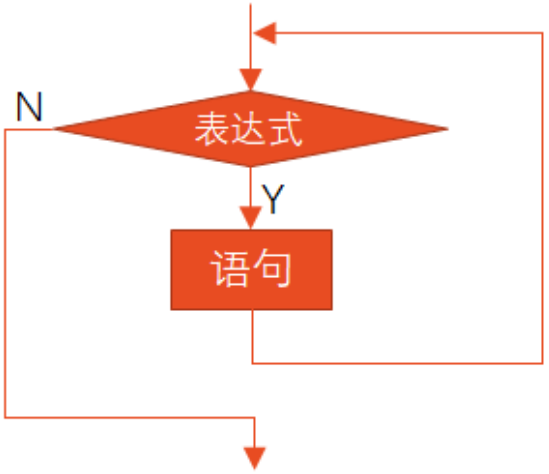
用循环控制处理重复操作

- 要向计算机输入全班 50 个学生的成绩;(重复 50 次相同的输入操作)
- 分别统计全班 50 个学生的平均成绩; (重复 50 次相同的计算操作)

```
float score1,score2,score3,score4,score5,aver; // 5门课成绩及平均成绩
int i=1;            // 设整型变量i初值为1
while( i<=50 )      // 当i的值小于或等于50时执行花括号内的语句
{
    scanf("%f%f%f%f%f",&score1,&score2,&score3,&score4,&score5);
    aver=(score1+score2+score3+score4+score5)/5;
    printf("aver=%7.2f",aver);
    i++;   // 每执行完一次循环使i的值加1
}
```

while(表达式){···}

```
while( 表达式
)
{
    // 循环体
    执行多条语句
;
}
```



while 循环特点

每轮循环: 首先判断表达式的值, 若 "真" (以非 0 值表示) 时, 就执行循环体语句; 为 "假" (以 0 表示) 时, 就不执行循环体语句。

易犯错误

```
while(表达式);
{
    // 循环体
```

变体

```
while(1)
{
  if(表达式) break; // 退出循环
```

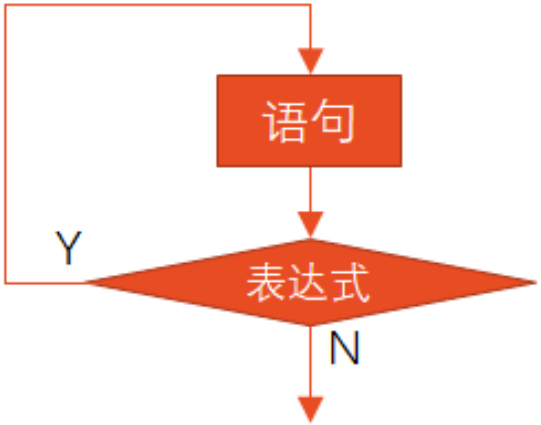
[例 5.1] 求 $1+2+3+\cdots+100$, 即 $\sum_{i=1}^{100} i$。

```
int i=1,sum=0;          //定义变量i的初值为1,sum的初值为0
while(i <= 100)         //当i>100, 条件表达式i<=100的值为假, 不执行循环体
{                       //循环体开始
    sum=sum+i;              //第1次累加后, sum的值为1
    i++;                    //加完后, i的值加1, 为下次累加做准备
}                       //循环体结束
printf("sum=%d\n",sum); //输出1+2+3…+100的累加和
```

1. 循环体如果包含一个以上的语句,应该用花括号括起来,作为复合语句出现。
2. 不要忽略给 i 和 sum 赋初值,否则它们的值是不可预测的,结果显然不正确。
3. 在循环体中应有使循环趋向于结束的语句。如本例中的 $i++$; 语句。如果无此语句,则 i 的值始终不改变,循环永远不结束。

do{···}while(表达式);

```
do
{
    // 循环体
    执行多条语句;
} while( 表达式 );
```



循环特点

先无条件地执行循环体,然后判断循环条件是否成立。。

易犯错误

```
do
{
    // 循环体
    执行多条语句;
```

变体

```
do
{
    执行多条语句;
    if(表达式) break; // 退出循环
```

[例 5.2] 求 $1+2+3+\cdots+100$, 即 $\sum_{i=1}^{100} i$。

```
int i=1,sum=0;          //定义变量i的初值为1,sum的初值为0
do
{                       //循环体开始
    sum=sum+i;              //第1次累加后，sum的值为1
    i++;                    //加完后，i的值加1，为下次累加做准备
}while(i <= 100);       //当i>100，条件表达式i<=100的值为假，不执行循环体
printf("sum=%d\n",sum); //输出1+2+3…+100的累加和
```

1. 在一般情况下,用 while(){$\cdots$} 语句和用 do{$\cdots$}while(); 语句处理同一问题时,若二者的循环体部分是一样的,那么结果也一样。
2. 但是如果 while 后面的表达式一开始就为假 (0 值) 时,两种循环的结果是不同的。

while(表达式){···} 与 *do*{···}*while*(表达式);

[例 5.3] 求 $\sum_{i=n}^{100} i$。考虑输入 $n > 100$ 时的情况，以下程序的不同。

```
int i,sum=0;
scanf("%d",&i);
while(i <= 100)
{
  sum=sum+i;
  i++;
}
printf("sum=%d\n",sum);
```

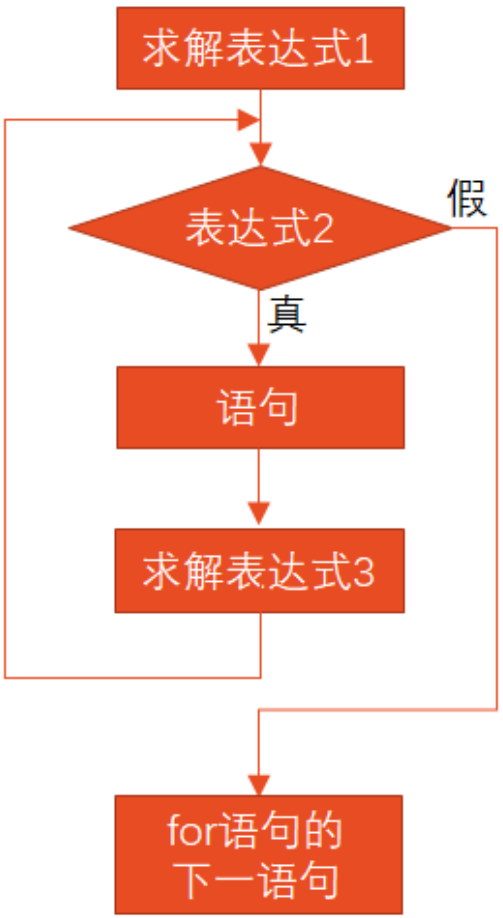
```
int i,sum=0;
scanf("%d",&i);
do
{
  sum=sum+i;
  i++;
}while(i <= 100);
printf("sum=%d\n",sum);
```

for(表达式 1; 表达式 2; 表达式 3){···}

```
for(表达式1;表达式2;表达式3)
{
    // 循环体
    执行多条语句;
}
```

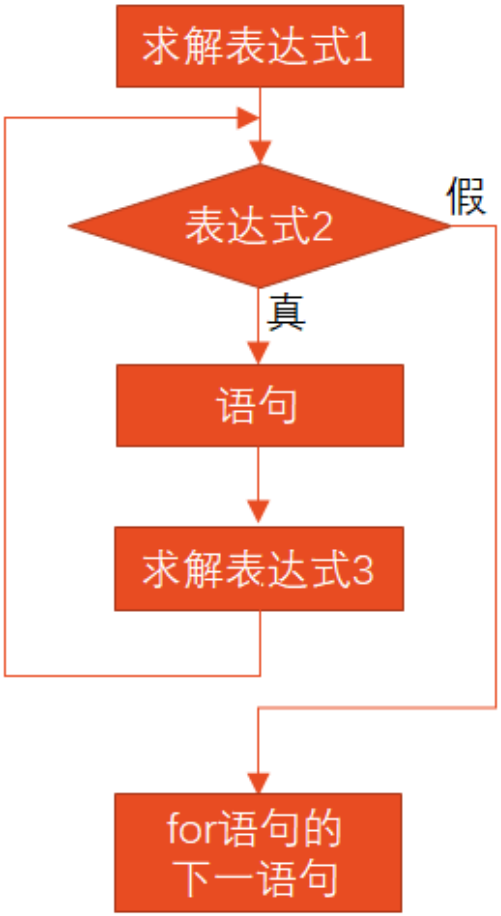
≡

```
表达式1;
while(表达式2)
{
    // 循环体
    执行多条语句;
    表达式3;
}
```



for(表达式 1; 表达式 2; 表达式 3){···}

```
for(表达式1;表达式2
    ;表达式3)
{
    // 循环体
    执行多条语句;
}
```



- 表达式 1: 设置初始条件,只执行一次。可以为零个、一个或多个变量 (逗号隔开) 设置初值。
- 表达式 2: 是循环条件表达式,用来判定是否继续循环。在每次执行循环体前先执行此表达式 (包括第 1 次循环),决定是否继续执行循环。
- 表达式 3: 作为循环的调整,例如使循环变量增值,它是在执行完循环体后才进行的。

for(表达式 1; 表达式 2; 表达式 3){···}, 省略表达式

```
int i;
for(i=0;i<=100;i++)
{
  printf("%d\n",i);
}
```

```
int i=0;
for(;i<=100;i++)
{
  printf("%d\n",i);
}
```

```
int i=0;
for(i=0;;i++)
{
  if(i>100) break;//退出循环
  printf("%d\n",i);
}
```

```
int i;
for(i=0;i<=100;)
{
  printf("%d\n",i);
  i++;
}
```

```
int i=0;
for(;i<=100;)
{
  printf("%d\n",i);
  i++;
}
```

```
int i=0;
for(;;)
{
  if(i>100) break;//退出循环
  printf("%d\n",i);
  i++
}
```

循环的嵌套

01

```
while()
{   ⋮
    while()
    {…}
}
```

内层循环

02

```
do
{   ⋮
    do
    {…}
    while();
}while();
```

内层循环

03

```
for( ; ; )
{
    for( ; ; )
    {…}
}
```

内层循环

04

```
while()
{   ⋮
    do
    {…}
    while();
    ⋮
}
```

内层循环

05

```
for( ; ; )
{   ⋮
    while()
    {…}
    ⋮
}
```

内层循环

06

```
do
{   ⋮
    for( ; ; )
    {…}
}while();
```

内层循环

几种循环的比较

1. 3 种循环都可以用来处理同一问题,一般情况下它们可以互相代替。
2. 在 while 循环和 do…while 循环中,只在 while 后面的括号内指定循环条件,因此为了使循环能正常结束,应在循环体中包含**使循环趋于结束的语句** (如 i++ 等)。
3. for 循环可以在表达式 3 中包含使循环趋于结束的操作,甚至可以将循环体中的操作全部放到表达式 3 中 (逗号隔开)。因此 for 语句的功能更强,凡用 while 循环能完成的,用 for 循环都能实现。
4. 用 while 和 do…while 循环时,循环变量初始化的操作应在 while 和 do…while 语句之前完成。而 for 语句可以在表达式 1 中实现循环变量的初始化。
5. while 循环、do…while 循环和 for 循环都可以用 break 语句跳出循环,用 continue 语句结束本次循环。

break,continue 改变循环执行的状态

```
while(表达式)
{
  printf("语句1");
  if(条件表达式) break; //提前终止循环
  printf("语句1");
}
```

```
while(表达式)
{
   printf("语句1");
   if(条件表达式) continue; //结束本次
    循环, 进入下轮循环
   printf("语句1");
}
```

用 break 语句提前终止循环

[例 5.4] 在全系 1000 名学生中举行慈善募捐,当总数达到 10 万元时就结束,统计此时捐款的人数以及平均每人捐款的数目。

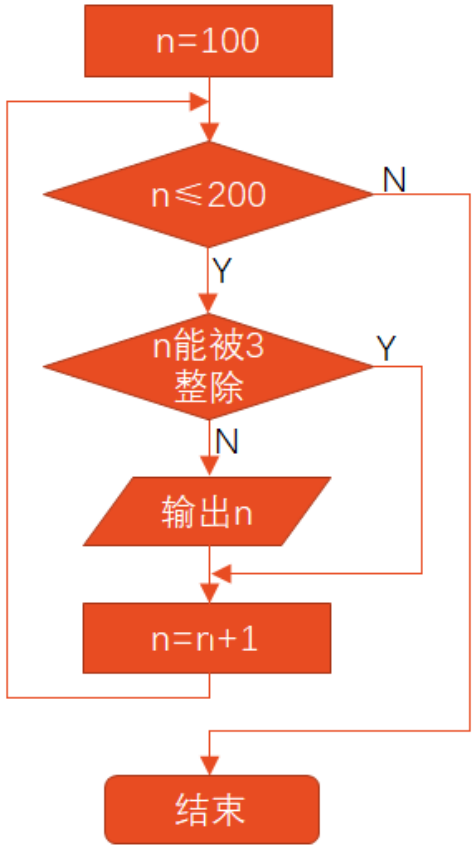
```
#define SUM 100000  //指定符号常量SUM代表10万
float amount,aver,total;
int i;
for (i=1,total=0; i<=1000; i++)// 表达式1给多个变量赋初值，用逗号隔开。
{
  printf("please␣enter␣amount:");
  scanf("%f",&amount);
  total = total + amount;
  if(total >= SUM) break;
}
aver=total/i;
printf("num=%d\naver=%10.2f\n",i,aver);
```

注意: break 语句只能用于循环语句和 switch 语句之中,而不能单独使用。

用 continue 语句提前结束本次循环

[例 5.4] 要求输出 100~200 之间的不能被 3 整除的数。

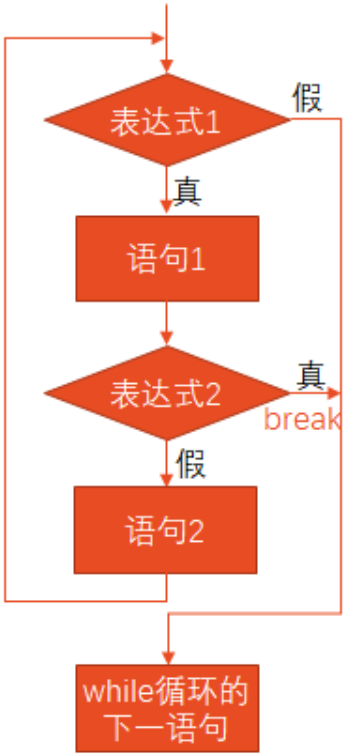
```
#include <stdio.h>
int main()
{
  int n;
  for (n = 100;n <= 200;n++)
  {
    if (n%3==0) continue;
    printf("%d␣",n);
  }
  printf("\n");
  return 0;
}
```



break 语句和 continue 语句的区别

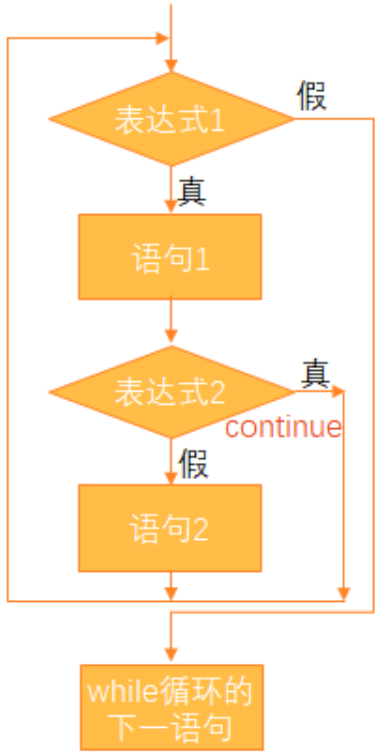
break;

```
while(表达式1)
{
        语句1;
        if(表达式2) break;
        语句2;
}
```



continue;

```
while(表达式1)
{
        语句1;
        if(表达式2) continue;
        语句2;
}
```



注意: continue 语句只结束本次循环,而非终止整个循环。break 语句结束整个循环,不再判断执行循环的条件是否成立。

break 语句和 continue 语句的区别

[例 5.5] 输出以下 4 × 5 的矩阵。

```
int i,j,n=0;
for(i=1;i<=4;i++)
{
  for(j=1;j<=5;j++,n++) //n用来累计输
     出数据的个数
  {
    if(n%5==0) printf("\n"); //控制
    在输出5个数据后换行
    if (i==3 && j==1) break;
    printf("%d\t",i*j);
  }
}
```

```
int i,j,n=0;
for(i=1;i<=4;i++)
{
    for(j=1;j<=5;j++,n++) //n用来累计
     输出数据的个数
    {
     if(n%5==0) printf("\n"); //控制
    在输出5个数据后换行
     if (i==3 && j==1) continue;
     printf("%d\t",i*j); // \t就是
    Tab键，是特殊字符，表示多个空格
    }
}
```

注意事项小结

1. while( ){ }; do { } while( ); for(;;){ } 执行顺序;
2. 循环变量的开始和结束条件;
3. 循环体是复合语句时,必须用 { } 扩起来;
4. 必要时,用 break 结束整个循环,用 continue 结束本次循环;
5. 关键是找出循环规律,必要时设计流程图,指导代码实现。

lecture-8 主要内容

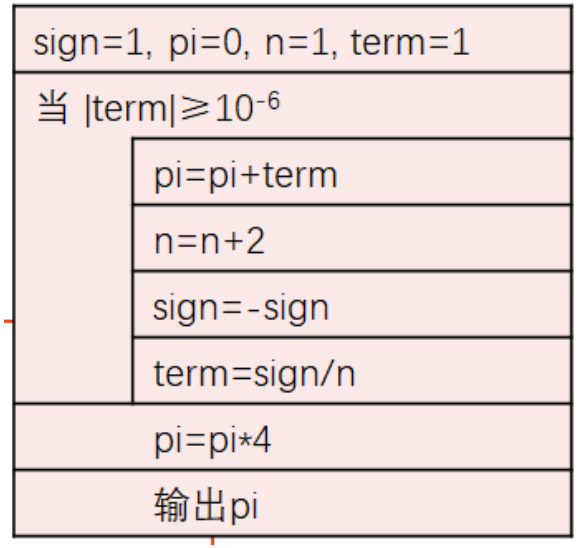
循环结构程序设计举例

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- while(表达式){···}
- do{···}while(表达式);
- for(表达式 1; 表达式 2; 表达式 3){···}
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6}, {7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
- 顺序举例
- 使用函数进行模块化程序设计
- 实参和形参间的数据传递 (按传递和地址传递)
- 顺序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与数传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 使用 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例

[例 5.7] 用公式 $\frac{\pi}{4} \approx 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$ 求 π 的近似值, 直到发现某一项的绝对值小于 10^{-6} 为止 (该项不累加)。

解题思路: 找规律

1. 每项的分子都是 1。
2. 后一项的分母是前一项的分母加 2。
3. 第 1 项的符号为正,从第 2 项起,每一项的符号与前一项的符号相反。在每求出一项后,检查它的绝对值是否大于或等于 10-6。



```c
#include <stdio.h>
#include <math.h> //程序中用到数学函数fabs，应包含头文件math.h
int main()
{
  int sign=1; //sign用来表示数值的符号
  double pi=0.0,n=1.0,term=1.0; //pi开始代表多项式的值，最后代表π的值，n代表分
    母，term代表当前项的值
  while(fabs(term)>=1e-6) //检查当前项term的绝对值是否大于或等于10-6
  {
    pi=pi+term; //把当前项term累加到pi中
    n=n+2;      //n+2是下一项的分母
    sign=-sign; //sign代表符号，下一项的符号与上一项符号相反
    term=sign/n; //求出下一项的值term
  }
  pi=pi*4; //多项式的和pi乘以4，才是π的近似值
  printf("pi=%10.8f\n",pi);  //输出π的近似值
  return 0;
}
```

[例 5.8] 求 Fibonacci(斐波那契) 数列的前 40 个数。这个数列有如下特点: 第 1, 2 两个数为 1, 1。从第 3 个数开始,该数是其前面两个数之和。即该数列为 1,1,2,3,5,8,13,..., 用数学方式表示为:

$$\begin{cases} F_1 = 1 & (n = 1) \\ F_2 = 1 & (n = 2) \\ F_n = F_{n-1} + F_{n-2} & (n \geq 3) \end{cases}$$

这是一个有趣的古典数学问题: 有一对兔子,从出生后第 3 个月起每个月都生一对兔子。小兔子长到第 3 个月后每个月又生一对兔子。假设所有兔子都不死,问每个月的兔子总数为多少?

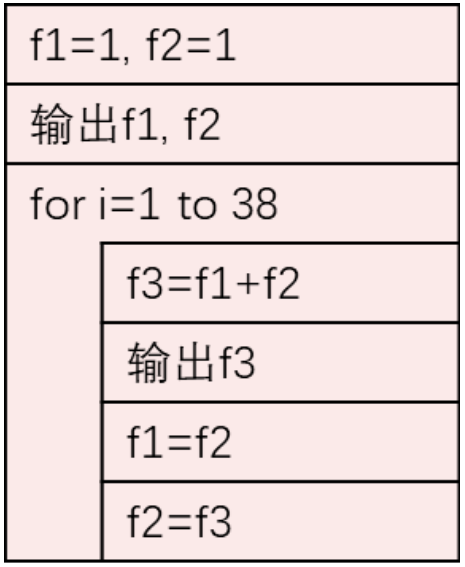
兔子繁殖的规律

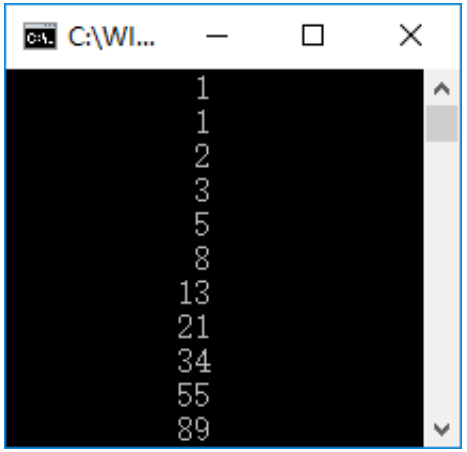
月数	小兔子对数	中兔子对数	老兔子对数	兔子总对数
1	1	0	0	1
2	0	1	0	1
3	1	0	1	2
4	1	1	1	3
5	2	1	2	5
6	3	2	3	8
7	5	3	5	13
⋮	⋮	⋮	⋮	⋮

不满 1 个月的为小兔子, 满 1 个月不满 2 个月的为中兔子, 满 2 个月以上的为老兔子。

解法一: 利用递推 (迭代) 公式: $F_1 = F_2 = 1; F_3 = F_1 + F_2; F_1 = F_2; F_2 = F_3;$

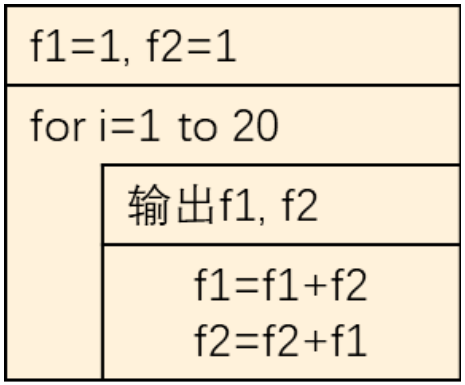
```
#include <stdio.h>
int main()
{
    int f1=1,f2=1,f3;
    int i;
    printf("%12d\n%12d\n",f1,f2);
    for(i=1; i<=38; i++)
    {
        f3=f1+f2;
        printf("%12d\n",f3);
        f1=f2;
        f2=f3;
    }
    return 0;
}
```

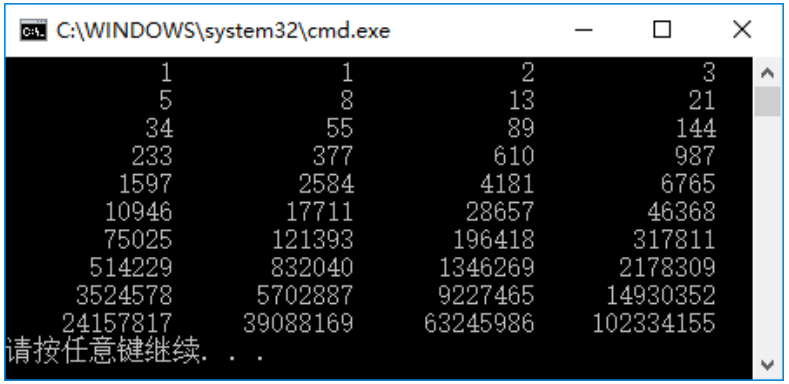




解法二: 利用递推 (迭代) 公式: $F_1 = F_2 = 1; F_1 = F_1 + F_2; F_2 = F_1 + F_2;$

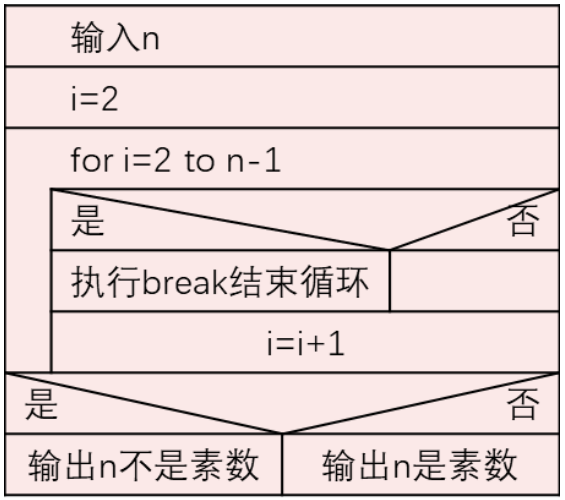
```c
#include <stdio.h>
int main()
{
  int f1=1,f2=1;
  int i;
  for(i=1; i<=20; i++)
  {
    printf("%12d%12d",f1,f2);
    if(i%2==0) // 等效 if(!(i%2))
       printf("\n");
    f1=f1+f2;
    f2=f2+f1;
  }
  return 0;
}
```





[例 5.9] 输入一个大于 3 的整数 n,判定它是否为素数 (prime,又称质数)。

```
#include <stdio.h>
int main()
{
  int n,i;
  printf("please␣enter␣a␣integer␣number,n=?");
  scanf("%d",&n);
  for (i=2;i<n;i++)
    if(n%i==0) break;
  if(i<n) // for提前结束
    printf("%d␣is␣not␣a␣prime␣number.\n",n);
  else // for正常结束
    printf("%d␣is␣a␣prime␣number.\n",n);
  return 0;
}
```



只要在循环结束后检查循环变量 i 的值,就能判定循环是提前结束还是正常结束的。从而判定 n 是否为素数。这种判断循环结束的方法以后会常用到。

优化: n 不必被 $2 \sim (n-1)$ 内的各整数去除, 只须将 n 被 $2 \sim \sqrt{n}$ 之间的整数除即可。因为 n 的每一对因子, 必然有一个小于 n, 另一个大于 n。

```
#include <stdio.h>
#include <math.h>
int main()
{
   int n,i,k;
   printf("please enter a integer number,n=?");
   scanf("%d",&n);
   k=sqrt(n); // 自动转换为整数(不会四舍五入), 相当于k=(int)sqrt(n);
   for (i=2;i<=k;i++)
     if(n%i==0) break;
   if(i<=k)
      printf("%d is not a prime number.\n",n);
   else
      printf("%d is a prime number.\n",n);
   return 0;
}
```

使用标志变量, 判断循环结束条件。

```
int n,i,k,flag=1; // flag: 标志变量
k=sqrt(n); // 自动转换为整数(不会四舍五入), 相当于k=(int)sqrt(n);
for (i=2;i<=k;i++)
   if(n%i==0) { flag=0; break; }
if(!flag) // for提前结束
   printf("%d␣is␣not␣a␣prime␣number.\n",n);
else // for正常结束
   printf("%d␣is␣a␣prime␣number.\n",n);
```

```
int n,i,k,flag=1; // flag: 标志变量
k=sqrt(n); // 自动转换为整数(不会四舍五入), 相当于k=(int)sqrt(n);
for (i=2;i<=k && flag;i++)
   if(n%i==0) { flag=0; }
if(!flag)
   printf("%d␣is␣not␣a␣prime␣number.\n",n);
else
   printf("%d␣is␣a␣prime␣number.\n",n);
```

使用标志变量, 判断循环结束条件。

```
int n,i,k,flag=1; // flag: 标志变量
k=sqrt(n); // 自动转换为整数(不会四舍五入), 相当于k=(int)sqrt(n);
for (i=2;i<=k;i++)
   if(n%i==0) { flag=0; break; }
if(!flag) // for提前结束
   printf("%d is not a prime number.\n",n);
else // for正常结束
   printf("%d is a prime number.\n",n);
```

```
int n,i,k,flag=1; // flag: 标志变量
k=sqrt(n); // 自动转换为整数(不会四舍五入), 相当于k=(int)sqrt(n);
for (i=2;i<=k && flag;i++)
   if(n%i==0) { flag=0; }
if(!flag)
   printf("%d is not a prime number.\n",n);
else
   printf("%d is a prime number.\n",n);
```

[例 5.10] 求 100 ∼ 200 间的全部素数。

```
#include <stdio.h>
#include <math.h>
int main()
{
  int n,i,k;
  for (n=101;i<=200;n+=2) //n从101变化到200, 对每个奇数n进行判定
  {
     k=sqrt(n); // 自动转换为整数(不会四舍五入), 相当于k=(int)sqrt(n);
     for(i=2;i<=k;i++)
        if(n%i==0) break;
     if(i>k)
        printf("%d␣",n);
  }
  printf("\n");
  return 0;
}
```

[例 5.10] 求 100 ∼ 200 间的全部素数。

```
#include <stdio.h>
#include <math.h>
int main()
{
  int n,i,k;
  for (n=101;i<=200;n+=2) //n从101变化到200, 对每个奇数n进行判定
  {
     k=sqrt(n); // 自动转换为整数(不会四舍五入), 相当于k=(int)sqrt(n);
     for(i=2;i<=k;i++)
        if(n%i==0) break;
     if(i>k)
        printf("%d␣",n);
  }
  printf("\n");
  return 0;
}
```

[例 5.11] 译密码。为使电文保密,往往按一定规律将其转换成密码,收报人再按约定的规律将其译回原文。例如,可以按以下规律将电文变成密码: 将字母 A 变成字母 E,a 变成 e,即变成其后的第 4 个字母,W 变成 A,X 变成 B,Y 变成 C,Z 变成 D。

```
char c;
c=getchar(); //输入一个字符给字符变量c
while(c!='\n') //检查c的值是否为换行符'\n'
{
  if((c>='a' && c<='z') || (c>='A' && c<='Z')) //c如果是字母
  {
     if((c>='W' && c<='Z') || (c>='w' && c<='z')) c = c-22; //如果是26个
    字母中最后4个字母之一就使c-22
     else  c =c + 4;   //如果是前面22个字母之一, 就使c + 4
  }
  printf("%c",c); //输出已改变的字符
  c=getchar();    //再输入下一个字符给字符变量c
}
printf("\n");
```

[例 5.11] 译密码。为使电文保密,往往按一定规律将其转换成密码,收报人再按约定的规律将其译回原文。例如,可以按以下规律将电文变成密码: 将字母 A 变成字母 E,a 变成 e,即变成其后的第 4 个字母,W 变成 A,X 变成 B,Y 变成 C,Z 变成 D。

```
char c;
c=getchar(); //输入一个字符给字符变量c
while(c!='\n') //检查c的值是否为换行符'\n'
{
  if((c>='a' && c<='z') || (c>='A' && c<='Z')) //c如果是字母
  {
     if((c>='W' && c<='Z') || (c>='w' && c<='z')) c = c-22; //如果是26个
    字母中最后4个字母之一就使c-22
     else  c =c + 4;   //如果是前面22个字母之一, 就使c + 4
  }
  printf("%c",c); //输出已改变的字符
  c=getchar();    //再输入下一个字符给字符变量c
}
printf("\n");
```

在循环条件中接收输入的字符是一种常见技巧。

```
char c;
while((c=getchar())!='\n') //检查c的值是否为换行符'\n'
{
    if((c>='a' && c<='z') || (c>='A' && c<='Z')) //c如果是字母
    {
        if((c>='W' && c<='Z') || (c>='w' && c<='z')) c = c-22; //如果是26个
     字母中最后4个字母之一就使c-22
        else  c =c + 4;   //如果是前面22个字母之一，就使c + 4
    }
    printf("%c",c); //输出已改变的字符
}
printf("\n");
```

lecture-9 主要内容

循环结构程序设计举例 (续)

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- while(表达式){···}
- do{···}while(表达式);
- for(表达式 1; 表达式 2; 表达式 3){···}
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6}, {7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
- 顺序举例
- 使用函数进行模块化程序设计
- 实参和形参间的数据传递 (按传递和地址传递)
- 顺序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与数传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 复习 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例

附加题 1: 求 $s = a + aa + aaa + \cdots + a \cdots a$, 其中 a 是一个 $1 \sim 9$ 的数字。例如 $a = 2, n = 4$ 时, $s = 2 + 22 + 222 + 2222$, a 和 n 由键盘输入。

```
int i,s,n,term = 0;
for(i=1,s=0; i<=n; i++) // 初始化循环
    变量用逗号隔开
{
    term = term*10 + a;
    s += term;
}
```

附加题 1: 求 $s = a + aa + aaa + \cdots + a\cdots a$, 其中 a 是一个 $1 \sim 9$ 的数字。例如 $a = 2, n = 4$ 时, $s = 2 + 22 + 222 + 2222$, a 和 n 由键盘输入。

```
int i,s,n,term = 0;
for(i=1,s=0; i<=n; i++) // 初始化循环
    变量用逗号隔开
{
    term = term*10 + a;
    s += term;
}
```

附加题 2: 韩信点兵。韩信有一队兵, 他想知道有多少人, 便让士兵排队报数:

按从 1 至 5 报数, 最末一个士兵报的数为 1;

按从 1 至 6 报数, 最末一个士兵报的数为 5;

按从 1 至 7 报数, 最末一个士兵报的数为 4;

按从 1 至 11 报数, 最末一个士兵报的数为 10;

计算韩信至少有多少兵。

```
int x=1;
for(;;x++)  // 循环体仅含if()结构, 看作一条语句, '{}'可省略
    if(x%5==1 && x%6==5 && x%7==4 && x%11==10)
    {
        printf("%d\n",x);
        break;
    }
```

附加题 2: 韩信点兵。韩信有一队兵, 他想知道有多少人, 便让士兵排队报数:

按从 1 至 5 报数, 最末一个士兵报的数为 1;

按从 1 至 6 报数, 最末一个士兵报的数为 5;

按从 1 至 7 报数, 最末一个士兵报的数为 4;

按从 1 至 11 报数, 最末一个士兵报的数为 10;

计算韩信至少有多少兵。

```
int x=1;
for(;;x++)   // 循环体仅含if()结构, 看作一条语句, '{}'可省略
    if(x%5==1 && x%6==5 && x%7==4 && x%11==10)
    {
        printf("%d\n",x);
        break;
    }
```

附加题 3-1: 求水仙花数。如果一个三位数的个位数、十位数和百位数的立方和等于该数自身,则称该数为水仙花数。

编程求出所有的水仙花数。

解法一: 采用三重循环

```
int i,j,k; // 百、十、个位
for(i=1;i<=9;i++)     // 百位
  for(j=0;j<=9;j++)   // 十位
    for(k=0;k<=9;k++)   // 个位
      if(i*100+j*10+k == i*i*i+j*j*j+k*k*k)
        printf("%d\n",i*100+j*10+k);
```

附加题 3-1: 求水仙花数。如果一个三位数的个位数、十位数和百位数的立方和等于该数自身,则称该数为水仙花数。

编程求出所有的水仙花数。

解法一: 采用三重循环

```
int i,j,k; // 百、十、个位
for(i=1;i<=9;i++)     // 百位
  for(j=0;j<=9;j++)   // 十位
    for(k=0;k<=9;k++)   // 个位
      if(i*100+j*10+k == i*i*i+j*j*j+k*k*k)
        printf("%d\n",i*100+j*10+k);
```

附加题 3-2: 求水仙花数。如果一个三位数的个位数、十位数和百位数的立方和等于该数自身,则称该数为水仙花数。

编程求出所有的水仙花数。

解法二: 采用一重循环

```
int m,i,j,k;
for(m=100;m<=999;m++)
{
    i=m/100; j=m/10%10; k=m%10;
    if(i*100+j*10+k == i*i*i+j*j*j+k*k*k)
        printf("%d\n",i*100+j*10+k);
}
```

思考: 输出共有多少个水仙数?

附加题 3-2: 求水仙花数。如果一个三位数的个位数、十位数和百位数的立方和等于该数自身,则称该数为水仙花数。

编程求出所有的水仙花数。

解法二: 采用一重循环

```
int m,i,j,k;
for(m=100;m<=999;m++)
{
    i=m/100; j=m/10%10; k=m%10;
    if(i*100+j*10+k == i*i*i+j*j*j+k*k*k)
        printf("%d\n",i*100+j*10+k);
}
```

思考: 输出共有多少个水仙数?

附加题 3-2: 求水仙花数。如果一个三位数的个位数、十位数和百位数的立方和等于该数自身,则称该数为水仙花数。

编程求出所有的水仙花数。

解法二: 采用一重循环

```
int m,i,j,k;
for(m=100;m<=999;m++)
{
    i=m/100; j=m/10%10; k=m%10;
    if(i*100+j*10+k == i*i*i+j*j*j+k*k*k)
        printf("%d\n",i*100+j*10+k);
}
```

思考: 输出共有多少个水仙数?

附加题 3-3: 求整数区间 $[a, b]$ 中水仙花数的个数。

```
int n=0; //计数
int a,b; // a,b 区间
int i,t;   // 循环变量，代表a,b区间的每个数
int sum; // i的各位立方和
scanf("%d%d",&a,&b);
for(i=a;i<=b;i++) // 考察i是否水仙数
{
  sum = 0; t=i; // 临时变量记住i; 易遗漏每次内层循环前sum要归0
  while(t!=0) // 累加各位立方
  { sum+=pow(t%10,3); t=t/10; }
  if(sum==i) n++; // i是水仙数
}
printf("%d\n",n);
```

附加题 3-3: 求整数区间 $[a, b]$ 中水仙花数的个数。

```
int n=0; //计数
int a,b; // a,b 区间
int i,t;   // 循环变量，代表a,b区间的每个数
int sum; // i的各位立方和
scanf("%d%d",&a,&b);
for(i=a;i<=b;i++) // 考察i是否水仙数
{
  sum = 0; t=i; // 临时变量记住i; 易遗漏每次内层循环前sum要归0
  while(t!=0) // 累加各位立方
  { sum+=pow(t%10,3); t=t/10; }
  if(sum==i) n++; // i是水仙数
}
printf("%d\n",n);
```

附加题 4: 百钱百鸡, 已知公鸡 5 个钱 1 只, 母鸡 3 个钱 1 只, 小鸡 1 个钱 3 只, 用 100 个钱买了 100 只鸡。问公鸡、母鸡、小鸡各几只?

```
int x,y,z; // 公鸡、母鸡、小鸡个数
for(x=0;x<=100;x++)
   for(y=0;y<=100;y++)
     for(z=0;z<=100;z++)
       if(5*x+3*y+z/3 == 100 && x+y+z == 100 && z%3 == 0) // 全部条件!
         printf("%d,%d,%d\n",x,y,z);
```

如何考虑无解的情况?

附加题 4: 百钱百鸡, 已知公鸡 5 个钱 1 只, 母鸡 3 个钱 1 只, 小鸡 1 个钱 3 只, 用 100 个钱买了 100 只鸡。问公鸡、母鸡、小鸡各几只?

```
int x,y,z; // 公鸡、母鸡、小鸡个数
for(x=0;x<=100;x++)
   for(y=0;y<=100;y++)
     for(z=0;z<=100;z++)
       if(5*x+3*y+z/3 == 100 && x+y+z == 100 && z%3 == 0) // 全部条件!
          printf("%d,%d,%d\n",x,y,z);
```

如何考虑无解的情况?

附加题 4: 百钱百鸡, 已知公鸡 5 个钱 1 只, 母鸡 3 个钱 1 只, 小鸡 1 个钱 3 只, 用 100 个钱买了 100 只鸡。问公鸡、母鸡、小鸡各几只?

```
int x,y,z; // 公鸡、母鸡、小鸡个数
for(x=0;x<=100;x++)
   for(y=0;y<=100;y++)
     for(z=0;z<=100;z++)
       if(5*x+3*y+z/3 == 100 && x+y+z == 100 && z%3 == 0) // 全部条件!
          printf("%d,%d,%d\n",x,y,z);
```

如何考虑无解的情况?

附加题 5-1: 求整数 a, b 的最大公约数,当两个数中有一个为 0 时,公约数是不为 0 的那个整数; 当两个整数互质时最大公约数为 1。输入两个整数 a 和 b,求最大公约数。

```
#include <stdio.h>
int main()
{
  int a,b,t,i;
  scanf("%d%d",&a,&b); // 机试系统不要想当然给提示语句, 除非题目要求
  if(a<b) { t=a; a=b; b=t; } // 交换a,b,使a是较大者
  for(i=b;i>0;i--)
    if(a%i==0 && b%i==0){ t=i; break; } // 求得最大公约数
  if(i==0)  // 如果循环结束, 还未求得公约数,
  {
    if(b==0) t = a;
    else t=1; // a,b互质
  }
  printf("%d\n",t);
  return 0;
}
```

附加题 5-1: 求整数 a, b 的最大公约数,当两个数中有一个为 0 时,公约数是不为 0 的那个整数; 当两个整数互质时最大公约数为 1。输入两个整数 a 和 b,求最大公约数。

```
#include <stdio.h>
int main()
{
  int a,b,t,i;
  scanf("%d%d",&a,&b); // 机试系统不要想当然给提示语句, 除非题目要求
  if(a<b) { t=a; a=b; b=t; } // 交换a,b,使a是较大者
  for(i=b;i>0;i--)
    if(a%i==0 && b%i==0){ t=i; break; } // 求得最大公约数
  if(i==0)  // 如果循环结束, 还未求得公约数,
  {
    if(b==0) t = a;
    else t=1; // a,b互质
  }
  printf("%d\n",t);
  return 0;
}
```

求整数 a, b 的最大公约数, 欧几里得算法

古希腊数学家欧几里德在其著作 «The Elements» 中最早描述了这种算法。

定理: 两个整数的最大公约数等于其中较小的那个数和两数相除余数的最大公约数。

伪代码分析

```
a,b的最大公约数, [则: a=mb+r, m=a/b; r=a%b]
循环, b作被除数, 分母是余数r,
[则: n=b/r; r=nb; a=mb+nb=(m+n)b; 如果一个整数能整除b, 必整除a]
直到r=0, 本轮循环的a(上轮循环的b)就是最大公约数。
while(1)
{
    r = a%b; // 注意b为0时, 不能计算余数, a就是最大公约数
    if(r==0) { gcd=a; break; } // 本轮循环的a(上轮循环的b)就是最大公约数
    a=b; b=r; // 准备下一轮迭代
}
```

求整数 a, b 的最大公约数, 欧几里得算法

```
int a,b,r,t;
scanf("%d%d",&a,&b); // 机试系统不要想当然给提示语句, 除非题目要求
if(a<b) { t=a; a=b; b=t; } // 交换a,b,使a是较大者
while(1)
{
    if(b==0) { t=a; break; } // 分母为0时, a就是最大公约数
    r = a%b;
    if(r==0) {t=b; break;} // 本轮循环的a(上轮循环的b)就是最大公约数
    a=b; b=r; // 准备下一轮迭代
}
printf("%d\n",t);// 输出最大公约数
```

附加题 6: 给出一个百分制的成绩,要求输出成绩等级'A','B','C','D','E'。90 分以上为'A',80 ~ 89 分为'B',70 ~ 79 分为'C',60 ~ 69 分为'D',60 分以下为'E'。

```
int grade;
scanf("%d",&grade);
grade /= 10; // 等效于 grade=grade/10;
switch(grade)
{
  case 0: case 1: case 2: case 3: case 4:
  case 5:  printf("E"); break;
  case 6:  printf("D"); break;
  case 7:  printf("C"); break;
  case 8:  printf("B"); break;
  case 9:
  case 10:  printf("A"); break;
}
```

思考: 如果输入成绩等级, 输出分数段, 如何修改程序?

附加题 6: 给出一个百分制的成绩,要求输出成绩等级'A','B','C','D','E'。90 分以上为'A',80 ~ 89 分为'B',70 ~ 79 分为'C',60 ~ 69 分为'D',60 分以下为'E'。

```
int grade;
scanf("%d",&grade);
grade /= 10; // 等效于 grade=grade/10;
switch(grade)
{
  case 0: case 1: case 2: case 3: case 4:
  case 5:  printf("E"); break;
  case 6:  printf("D"); break;
  case 7:  printf("C"); break;
  case 8:  printf("B"); break;
  case 9:
  case 10:  printf("A"); break;
}
```

思考: 如果输入成绩等级, 输出分数段, 如何修改程序?

注意事项小结

1. while( ){ }; do { } while( ); for(;;){ } 执行顺序;
2. 循环变量的开始和结束条件;
3. 循环体是复合语句时,必须用 { } 扩起来;
4. 必要时,用 break 结束整个循环,用 continue 结束本次循环;
5. 关键是找出循环规律,必要时设计流程图,指导代码实现。

lecture-10 主要内容

利用数组处理批量数据

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- while(表达式){···}
- do{···}while(表达式);
- for(表达式 1; 表达式 2; 表达式 3){···}
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6}, {7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
- 顺序举例
- 使用函数进行模块化程序设计
- 变参和形参间的数据传递 (按传递和地址传递)
- 顺序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *pi;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 复习 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例

为什么需要数组

- 要向计算机输入全班 50 个学生的成绩一门课的成绩;
- 用 50 个 float 型简单变量表示学生的成绩
  - **烦琐**,如果有 1000 名学生怎么办呢?
  - 没有反映出这些**数据间的内在联系**,实际上这些数据是同一个班级、同一门课程的成绩,它们具有相同的属性。

```
float s0,s1,s2,...,s49; // 50名学生一门课的成绩
```

```
float s[50]; // 50名学生一门课的成绩
int i; // 表示数组下标
for(i=0;i<50;i++) scanf("%f",&s[i]);
```

数组

1. 数组是一组有序数据的集合。数组中各数据的排列是有一定规律的,下标代表数据在数组中的序号。
2. 用数组名和下标即可唯一地确定数组中的元素。
3. 数组中的每一个元素都属于同一个数据类型。

为什么需要数组

- 要向计算机输入全班 50 个学生的成绩一门课的成绩;
- 用 50 个 float 型简单变量表示学生的成绩
  - **烦琐**,如果有 1000 名学生怎么办呢?
  - 没有反映出这些**数据间的内在联系**,实际上这些数据是同一个班级、同一门课程的成绩,它们具有相同的属性。

```
float s0,s1,s2,...,s49; // 50名学生一门课的成绩

float s[50]; // 50名学生一门课的成绩
int i; // 表示数组下标
for(i=0;i<50;i++) scanf("%f",&s[i]);
```

数组

1. 数组是一组有序数据的集合。数组中各数据的排列是有一定规律的,下标代表数据在数组中的序号。
2. 用数组名和下标即可唯一地确定数组中的元素。
3. 数组中的每一个元素都属于同一个数据类型。

定义数组: int a[10];

定义一维数组:

元素类型　数组名 [常量表达式 (表示元素个数—数组的长度)]

```
#define NUM 100
float s[50]; // 50名学生一门课的成绩
int a[10]; // 10个元素的整型数组
double b[NUM]; // 常量NUM个元素的double数组
char c[50]; // 50个元素的char型数组
```

Notes

数组元素的下标从 0 开始, int a[10]; 10 个整型元素, 则最大下标值为 9, 不存在数组元素 a[10]

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

引用数组: int i=0; a[i]

```
int a[10]; // 10个元素的整型数组
int i;
a[0]=10; // 给a数组的第一个元素赋值
a[9]=10; // 给a数组的最后一个元素赋值
printf("%d,%d",a[0],a[9]);
for(i=0;i<10;i++) a[i] = i+1; // 给数组的第i个元素赋值
for(i=0;i<10;i++) printf("%d\t",a[i]); // 输出数组a的10个元素
for(i=0;i<10;i++) scanf("%d",&a[i]); // 输入10个整数, 存入数组a中。注意'&'
```

Notes

数组元素的下标从 0 开始,int a[10]; 10 个整型元素,则最大下标值为 9,不存在数组元素 a[10]

[例 6.1] 对 10 个字符型数组元素依次赋值为'a','b',· · · 。要求按逆序输出。

```
#include <stdio.h>
#define N 10
int main()
{
  char c[N];
  int i;
  for(i=0;i<N;i++) c[i]='a'+i;
  for(i=N-1;i>=0;i--) printf("%c␣", c[i]);
  printf("\n");
  return 0;
}
```

[例 6.1] 对 10 个字符型数组元素依次赋值为'a','b',· · · 。要求按逆序输出。

```
#include <stdio.h>
#define N 10
int main()
{
  char c[N];
  int i;
  for(i=0;i<N;i++) c[i]='a'+i;
  for(i=N-1;i>=0;i--) printf("%c␣", c[i]);
  printf("\n");
  return 0;
}
```

初始化数组: int a[5]={1,2,3,4,5};

为了使程序简洁,常在定义数组的同时给各数组元素赋值,这称为数组的初始化。

```
int a[10]={1,2,3,4,5,6,7,8,9,10}; // 在定义数组时对全部数组元素赋予初值。
char c[10]={'a','b'}; // 可以只给数组中的一部分元素赋值。其他元素的值不确定
double d[]={10.0,10.2,10.3}; // 等效于 double d[3]={10.0,10.2,10.3}
d[2] = 20.2; // 修改第3个元素
```

[例 6.2] 用数组来处理求 Fibonacci 数列问题。

```
#include <stdio.h>
#define N 20
int main()
{
  int i;
  int f[N]={1,1};  //对最前面两个元素f[0]和f[1]赋初值1
  for(i=2;i<N;i++)
     f[i]=f[i-2]+f[i-1]; //先后求出f[2]~f[19]的值
  for(i=0;i<N;i++)
  {
     if(i%5==0) printf("\n"); //控制每输出5个数后换行
     printf("%12d",f[i]); //输出一个数
  }
  printf("\n");
  return 0;
}
```

冒泡排序 (第 1 趟比较)

int a[4]={9,8,5,0}; // N=4 个元素, 要求从小到大顺序排列

9
8
5
0

第 1 趟原始数据

8
9
5
0

第 1 趟第 1 次相邻两数比较

8
5
9
0

第 1 趟第 2 次相邻两数比较

8
5
0
9

第 1 趟第 3 次相邻两数比较 (第 1 个大数沉底)

第 j=1 趟比较,N-j=3 次相邻两数比较

冒泡排序 (第 1 趟比较)

int a[4]={9,8,5,0}; // N=4 个元素, 要求从小到大顺序排列

9
8
5
0

第 1 趟原始数据

8
9
5
0

第 1 趟第 1 次相邻两数比较

8
5
9
0

第 1 趟第 2 次相邻两数比较

8
5
0
9

第 1 趟第 3 次相邻两数比较 (第 1 个大数沉底)

第 j=1 趟比较,N-j=3 次相邻两数比较

冒泡排序 (第 1 趟比较)

int a[4]={9,8,5,0}; // N=4 个元素, 要求从小到大顺序排列

9
8
5
0

第 1 趟原始数据

8
9
5
0

第 1 趟第 1 次相邻两数比较

8
5
9
0

第 1 趟第 2 次相邻两数比较

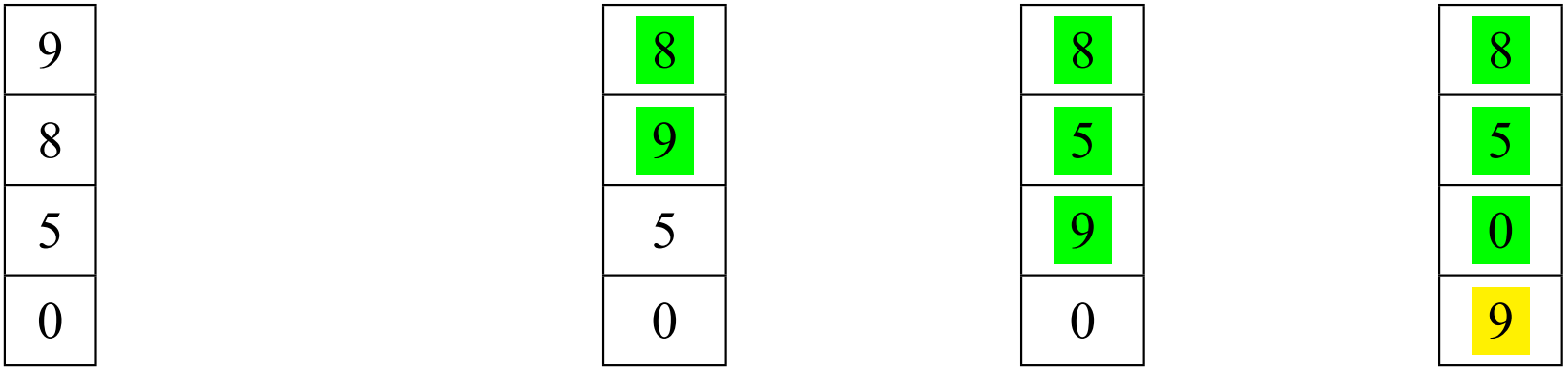
8
5
0
9

第 1 趟第 3 次相邻两数比较 (第 1 个大数沉底)

第 j=1 趟比较,N-j=3 次相邻两数比较

冒泡排序 (第 1 趟比较)

int a[4]={9,8,5,0}; // N=4 个元素, 要求从小到大顺序排列



第 1 趟原始数据

第 1 趟第 1 次相邻两数比较

第 1 趟第 2 次相邻两数比较

第 1 趟第 3 次相邻两数比较 (第 1 个大数沉底)

第 j=1 趟比较,N-j=3 次相邻两数比较

冒泡排序 (第 2 趟比较)

int a[4]={9,8,5,0}; // N=4 个元素, 要求从小到大顺序排列

8
5
0
9

第 2 趟原始数据

5
8
0
9

第 2 趟第 1 次相邻两数比较

5
0
8
9

第 2 趟第 2 次相邻两数比较 (第 2 大沉底)

第 j=2 趟比较,N-j=2 次相邻两数比较

冒泡排序 (第 2 趟比较)

int a[4]={9,8,5,0}; // N=4 个元素, 要求从小到大顺序排列

8
5
0
9

第 2 趟原始数据

5
8
0
9

第 2 趟第 1 次相邻两数比较

5
0
8
9

第 2 趟第 2 次相邻两数比较 (第 2 大沉底)

第 j=2 趟比较,N-j=2 次相邻两数比较

冒泡排序 (第 2 趟比较)

int a[4]={9,8,5,0}; // N=4 个元素, 要求从小到大顺序排列

8
5
0
9

第 2 趟原始数据

5
8
0
9

第 2 趟第 1 次相邻两数比较

5
0
8
9

第 2 趟第 2 次相邻两数比较 (第 2 大沉底)

第 j=2 趟比较,N-j=2 次相邻两数比较

冒泡排序 (第 3 趟比较)

int a[4]={9,8,5,0}; // N=4 个元素, 要求从小到大顺序排列

5
0
8
9

第 3 趟原始数据

0
5
8
9

第 3 趟第 1 次相邻两数比较 (第 3 大沉底)

第 j=3 趟比较,N-j=1 次相邻两数比较

冒泡排序 (第 3 趟比较)

int a[4]={9,8,5,0}; // N=4 个元素, 要求从小到大顺序排列

5
0
8
9

第 3 趟原始数据

0
5
8
9

第 3 趟第 1 次相邻两数比较 (第 3 大沉底)

第 j=3 趟比较,N-j=1 次相邻两数比较

冒泡排序 (总结)

- #define N 10 // 数组长度
- $(N-1)$ 趟外层循环, $(j = 1, 2, \cdots, N-1)$, 表示第 j 趟比较。
- $(N-j)$ 次内层循环, $(i = 0, 1, \cdots, N-1-j)$ 相邻元素两两比较, 必要时交换。
- 注意检查数组边界条件, 不要越界。分别进行 N=1, 2, 3, 4 个数排序演练。

冒泡排序 (核心程序)

```
#define N 10 // 数组长度
int a[N]={9,8,7,6,5,4,3,2,1,0},i,j,t;
for(j=1;j<=N-1;j++) //进行N-1次循环，实现N-1趟比较
{
  for(i=0;i<=N-1-j;i++) //在每一趟中进行N-j次比较相邻元素两两比较
  {
    if(a[i]>a[i+1]) //相邻两个数比较，注意检查数组不要越界
       { t=a[i]; a[i]=a[i+1]; a[i+1]=t; } // 交换
  }
}
printf("\n␣the␣sorted␣numbers␣:\n");
for(i=0;i<N;i++)
   printf("%d␣",a[i]);
```

冒泡排序 (优化)

优化：第 j 趟排序中，没有进行相邻元素的交换，表示数据已经排序好，没有必要进行此后的 $(N-j)$ 趟排序。

```
#define N 10 // 数组长度
int a[N]={7,8,7,6,5,6,7,8,9,10},i,j,t,flag;
for(j=1;j<=N-1;j++) //进行N-1次循环，实现N-1趟比较
{
   flag = 0; // 每趟排序，初始化flag，表示未进行交换
   for(i=0;i<=N-1-j;i++) //在每一趟中进行N-j次相邻元素两两比较
   {
     if(a[i]>a[i+1]) //相邻两个数比较，注意检查数组不要越界
       { t=a[i]; a[i]=a[i+1]; a[i+1]=t; flag=1; } // 交换，设置标志变量
   }
   if(!flag) break; // 表示第j趟未交换，排序好了！
}
printf("\n␣the␣sorted␣numbers␣:\n");
for(i=0;i<N;i++) printf("%d␣",a[i]);
```

冒泡排序 (输出每趟排序的结果)

优化：第 j 趟排序中，没有进行相邻元素的交换，表示数据已经排序好，没有必要进行此后的 $(n-j)$ 趟排序。

```
#define N 10 // 数组长度
int a[N]={7,8,7,6,5,6,7,8,9,10},i,j,t,flag;
for(j=1;j<=N-1;j++) //进行N-1次循环，实现N-1趟比较
{
   flag = 0; // 每趟排序，初始化flag，表示未进行交换
   for(i=0;i<=N-1-j;i++) //在每一趟中进行N-j次相邻元素两两比较
   {
      if(a[i]>a[i+1]) //相邻两个数比较，注意检查数组不要越界
       { t=a[i]; a[i]=a[i+1]; a[i+1]=t; flag=1; } // 交换，设置标志变量
   }
   printf("\n␣第%d趟排序:␣\n", j);
   for(t=0;t<N;t++) printf("%d␣",a[t]); // 临时变量t的复用
   if(!flag) break; // 表示第j趟未交换，排序好了！
}
printf("\n␣the␣sorted␣numbers␣:\n");
for(i=0;i<N;i++) printf("%d␣",a[i]);
```

lecture-11 主要内容

二维数组和字符数组

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- while(表达式){···}
- do{···}while(表达式);
- for(表达式 1; 表达式 2; 表达式 3){···}
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6}, {7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
- 顺序举例
- 使用函数进行模块化程序设计
- 实参和形参间的数据传递 (值传递和地址传递)
- 顺序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 使用 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例

二维数组应用场景

Example

有 3 个小分队,每队有 6 名队员,要把这些队员的工资用数组保存起来以备查。

	队员 1	队员 2	队员 3	队员 4	队员 5	队员 6
第 1 分队	1000	2000	1500	2400	3000	5000
第 2 分队	3000	4000	2500	2300	2500	4000
第 3 分队	4000	5000	1200	2300	3200	5500

```
float pay[3][6]; // 定义3行6列的二维数组, 称为3x6矩阵(matrix)。
```

pay[0][0]	pay[0][1]	pay[0][2]	pay[0][3]	pay[0][4]	pay[0][5]
pay[1][0]	pay[1][1]	pay[1][2]	pay[1][3]	pay[1][4]	pay[1][5]
pay[2][0]	pay[2][1]	pay[2][2]	pay[2][3]	pay[2][4]	pay[2][5]

定义二维数组: int a[10][20];

定义二维数组:

元素类型　数组名 [行数][列数]

```
int a[5][10]; // 定义5行10列的整型二维数组
float pay[3][6]; // 定义3行6列的单精度浮点型二维数组
double pay1[3][6]; // 定义3行6列的双精度浮点型二维数组
char c[2][20]; // 定义2行20列的字符型二维数组

// 行数列数定义为常量
#define M 5 // 行数
#define N 10 // 列数
int a[M][N]; // 定义M行N列的整型二维数组
```

二维数组的存储

```
float a[3][4]; // 定义3行4列的单精度浮点型二维数组
```

1000	a[0][0]
1004	a[0][1]
1008	a[0][2]
1012	a[0][3]
1016	a[1][0]
1020	a[1][1]
1024	a[1][2]
1028	a[1][3]
1032	a[2][0]
1036	a[2][1]
1040	a[2][2]
1044	a[2][3]

Notes

用矩阵形式(如 3 行 4 列形式)表示二维数组,是逻辑上的概念,能形象地表示出行列关系。而在内存中,各元素是连续存放的,不是二维的,是线性的。

数组元素的下标从 0 开始,int a[5][10]; 50 个整型元素,先行后列存储数组元素, 第 1 行第 1 列元素是 a[0][0], 第 5 行第 10 列元素是 a[4][9]。

a[0][0]	a[0][1]	a[0][2]	a[0][3]	⇐ 第 0 行
a[1][0]	a[1][1]	a[1][2]	a[1][3]	⇐ 第 1 行
a[2][0]	a[2][1]	a[2][2]	a[2][3]	⇐ 第 2 行

引用数组: int i=0,j=0; a[i][j]

```
#define M 5  // 行数
#define N 10 // 列数
float a[M][N]; // 定义M行N列的单精度浮点型二维数组
int i,j;

a[0][0]=10;  // 给第0行第0列元素赋值
a[M-1][N-1]=20; // 给第M-1行第N-1列元素赋值
```

```
// 数组元素赋值
for(i=0;i<M;i++) //遍历第i行
  for(j=0;j<N;j++) //遍历第j列
    a[i][j]=i*j; //给第i行第j列元素赋值

// 键盘输入数组元素
for(i=0;i<M;i++)
   for(j=0;j<N;j++)
     scanf("%f", &a[i][j]);
```

```
// 输出数组元素
for(i=0;i<M;i++)
   for(j=0;j<N;j++)
   {
      printf("%d\t",a[i][j]);
      if(j==N-1) printf("\n");
   }
```

输出每个小分队的平均工资

```
#define M 100 // 估计最大的行数
#define N 100 // 估计最大的列数
float pay[M][N]; //按照最大值定义数组的大小
int m,n,i,j; //m,n表示实际pay数组行数和列数
float sum; //每个小分队所有人员的工资总和
scanf("%d%d",&m,&n); // 输入小分队数和队员数, 就是pay数组的实际行数和列数
for(i=0;i<m;i++)  // 依次输入每个队员的工资
  for(j=0;j<n;j++) scanf("%f", &pay[i][j]); // 注意取地址符'&'
for(i=0;i<m;i++)
{
   sum=0; // 内层循环执行前, 必须置零。易忘记!!!
   for(j=0;j<n;j++)
      sum += pay[i][j]; // 累加第i小分队所有人员的工资
   printf("第%d小分队平均工资=%f\n",i+1,sum/n);
}
```

pay[0][0]	pay[0][1]	pay[0][2]	pay[0][3]	pay[0][4]	pay[0][5]
pay[1][0]	pay[1][1]	pay[1][2]	pay[1][3]	pay[1][4]	pay[1][5]
pay[2][0]	pay[2][1]	pay[2][2]	pay[2][3]	pay[2][4]	pay[2][5]

二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6},{7,8,9} }

```
#define M 3 // 行数
#define N 3 // 列数
int a[M][N]={{1,2,3},{4,5,6},{7,8,9}}; // 定义二维数组，并初始化
int b[M][N]={{1,2},{4,5},{7,8}}; // 定义二维数组，并初始化
int c[M][N]={{1,2},{},{7,8}}; // 定义二维数组，并初始化

// 给未初始化的元素赋值
b[0][N-1]=3;  b[1][N-1]=6; b[2][N-1]=9;
c[0][N-1]=3;
c[1][0]=4;  c[1][1]=5; c[1][N-1]=6;
c[2][N-1]=9;
```

例 6.4 将二维数组行列互换, 存到另一个二维数组中。

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \Longrightarrow b = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
#define M 2
#define N 3
int a[M][N]={{1,2,3},{4,5,6}};
int b[N][M],i,j;
```

```
for(i=0;i<M;i++)
{
    for(j=0;j<N;j++)
    {
        printf("%5d",a[i][j]);
        b[j][i]=a[i][j];
    }
    printf("\n"); // 换行
}
```

```
for(i=0;i<N;i++)
{
    for(j=0;j<M;j++)
    {
        printf("%5d",b[i][j]);
    }
    printf("\n"); // 换行
}
```

例 6.4 将二维数组行列互换, 存到另一个二维数组中。

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \Longrightarrow b = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
#define M 2
#define N 3
int a[M][N]={{1,2,3},{4,5,6}};
int b[N][M],i,j;
```

```
for(i=0;i<M;i++)
{
    for(j=0;j<N;j++)
    {
        printf("%5d",a[i][j]);
        b[j][i]=a[i][j];
    }
    printf("\n"); // 换行
}
```

```
for(i=0;i<N;i++)
{
    for(j=0;j<M;j++)
    {
        printf("%5d",b[i][j]);
    }
    printf("\n"); // 换行
}
```

例 6.4 将二维数组行列互换, 存到另一个二维数组中。

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \Longrightarrow b = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```c
#define M 2
#define N 3
int a[M][N]={{1,2,3},{4,5,6}};
int b[N][M],i,j;

    for(i=0;i<M;i++)
    {
        for(j=0;j<N;j++)
        {
            printf("%5d",a[i][j]);
            b[j][i]=a[i][j];
        }
        printf("\n"); // 换行
    }
```

```c
for(i=0;i<N;i++)
{
    for(j=0;j<M;j++)
    {
        printf("%5d",b[i][j]);
    }
    printf("\n"); // 换行
}
```

例 6.4 将二维数组行列互换, 存到另一个二维数组中。

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \Longrightarrow b = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
#define M 2
#define N 3
int a[M][N]={{1,2,3},{4,5,6}};
int b[N][M],i,j;
```

```
for(i=0;i<M;i++)
{
    for(j=0;j<N;j++)
    {
        printf("%5d",a[i][j]);
        b[j][i]=a[i][j];
    }
    printf("\n"); // 换行
}
```

```
for(i=0;i<N;i++)
{
    for(j=0;j<M;j++)
    {
      printf("%5d",b[i][j]);
    }
    printf("\n"); // 换行
}
```

例 6.5 求二维数组中的最大元素及其行号和列号。

```
#define M 2
#define N 3
int a[M][N]={{1,2,3},{4,5,6}},i,j,max,row=0,col=0;
max=a[0][0]; // //先认为a[0][0]最大
for(i=0;i<M;i++)
{
    for(j=0;j<N;j++)
    {
      if(a[i][j]>max)
      {
         max=a[i][j]; row=i; col=j;
      }
    }
}
printf("max=%d,row=%d,col=%d\n",max,row,col);
```

例: 方阵 int a[10][10] 左右对角线

例: 使方阵 (行数 = 列数) 左对角线元素置 1, 其它元素为 0.

```
#define M 10
int a[M][M],i,j;
for(i=0;i<M;i++)
{
    for(j=0;j<M;j++)
    {
        //右对角线, if(i+j==M)
        if(i==j) a[i][j]=1;
        else a[i][j]=0;
    }
}
```

```
// 输出
for(i=0;i<M;i++) // 行
{
    for(j=0;j<M;j++) // 列
        printf("%5d", a[i][j])
    ;
    printf("\n"); // 换行
}
```

例: 方阵 int a[10][10] 上下三角阵

例: 使方阵 (行数 = 列数) 下三角阵元素置 1, 其它元素为 0.

```
#define M 10
int a[M][M],i,j;
for(i=0;i<M;i++) // 行
   for(j=0;j<M;j++) // 列
   {
     // 如果是上三角阵, if(j>=i
    )
     if(j<=i) a[i][j]=1;
     else a[i][j]=0;
   }
printf("下三角阵\n");
for(i=0;i<M;i++) // 行
{
   for(j=0;j<=i;j++) // 列
      printf("%5d", a[i][j])
    ;
   printf("\n"); // 换行
}
```

```
printf("上三角阵\n");
for(i=0;i<M;i++) // 行
{
    for(j=0;j<M;j++) // 列
       if(j>=i) printf("%5d", a[i][j]);
       else printf("%5c", '␣');
    printf("\n"); // 换行
}
```

```
下三角阵
    1
    1    1
    1    1    1
    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1    1
    1    1    1    1    1    1    1
    1    1    1    1    1    1    1    1
    1    1    1    1    1    1    1    1    1
    1    1    1    1    1    1    1    1    1    1
上三角阵
    1    0    0    0    0    0    0    0    0    0
         1    0    0    0    0    0    0    0    0
              1    0    0    0    0    0    0    0
                   1    0    0    0    0    0    0
                        1    0    0    0    0    0
                             1    0    0    0    0
                                  1    0    0    0
                                       1    0    0
                                            1    0
                                                 1
```

用字符数组表示字符串 char s[81]

字符数组是用来存放字符数据的数组是字符数组。在字符数组中的一个元素内存放一个字符。

两种方式初始化字符数组

```
// s[0]='a', s='b', s='c', s[3]='d', s[4]以后的字符未赋值
char s[81]={'a','b','c','d'};  // 4个有效字符

// str[0]='a', str[1]='b', str[2]='c', str[3]='d', str[4]='\0'
// 自动追加'\0', 表示字符串结束)
char str[]="abcd"; // 5个有效字符
```

使用 printf 函数输出字符串

```
// s[0]='a', s='b', s='c', s[3]='d', s[4]以后的字符未赋值
char s[81]={'a','b','c','d'}; // 4个有效字符
for(i=0;i<4;i++)
  printf("%c\t",s[i]); // a b c d

// str[0]='a', str[1]='b', str[2]='c', str[3]='d', str[4]='\0'
// 自动追加'\0', 表示字符串结束)
char str[]="abcd"; // 5个有效字符
for(i=0;a[i]!='\0';i++)
  printf("%c\t",str[i]); // a b c d

// 或使用格式描述符%s, 输出以'\0'结尾的字符串
printf("%s\n",str); // abcd, 输出'\0'以前的字符

printf("%s\n",s); // 错误, 由于s不是以'\0'结尾的字符串
s[4]='\0'; // 使字符数组以'\0'结尾, 就可以使用上一句正常输出了
```

使用 scanf 函数输入字符串

```
char a[81],b[81],c[81];

// 遇空格或回车结束，自动追加字符串结束字符'\0'
scanf("%s",a); // 注意字符数组前没有取地址符号'&'
// 例如输入: abc回车，则a[0]='a',a[1]='b',a[2]='c',a[3]='\0'
printf("%s\n",a); // abc

// 以空格隔开，输入3个字符串。自动追加字符串结束字符'\0'
scanf("%s%s%s",a,b,c);
// 例如输入: How are You回车
printf("%s,%s,%s\n",a,b,c); // How,are,You
```

推荐使用 gets 函数和 puts 函数输入输出字符串

```
char a[81];

// 可以接收带空格的字符串，遇回车结束，自动追加字符串结束字符'\0'
gets(a);
// 例如输入: ab cd回车

// 遇字符串结束字符'\0'，换行退出
puts(a); // ab cd
```

lecture-12 主要内容

字符串处理库函数及程序举例

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- *while*(表达式){···}
- *do*{···}*while*(表达式);
- *for*(表达式 1; 表达式 2; 表达式 3){···}
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6},{7,8,9}}
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
  - 连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)
  - 字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)
  - 获取字符串的长度 strlen(s)
  - 大小写转换 strlwr(s); strupr(s)
- 程序举例
  - 统计单词数
  - 求多个字符串中的较大者
  - 矩阵乘法
- 使用函数进行模块化程序设计
  - 函数三要素: 函数原型声明, 函数定义, 函数调用
  - 函数定义: int add(int a,int b){ }
- 实参和形参间的数据传递 (值传递和地址传递)
- 程序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 整数 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例续

字符串处理函数 #include<string.h>

- 使用字符串处理库函数, 必须包含头文件 #include<string.h>
- 要求字符串必须以 '\0' 结尾

```
#include<stdio.h>
#include<string.h>
int main()
{
   char s1[81]="ab"; // 自动追加s1[2]='\0'
   char s2[81]="cdef"; // 自动追加s2[4]='\0'

   // 连接两个字符串, 结果被放入s1中, s1数组要足够大, 能够容纳两个字符串连接后的长
    度+1, 多一个字符长度是留给'\0'使用
   strcat(s1,s2);

   puts(s1); // abcdef, 最后一个字符是'\0'
   puts(s2); // cdef, 保持不变, , 最后一个字符是'\0'
   return 0;
}
```

连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)

```
char s1[81]="ab"; // 自动追加s1[2]='\0'
char s2[81]="cdef"; // 自动追加s2[4]='\0'
// 连接两个字符串，结果被放入s1中，s1数组要足够大，能够容纳两个字符串连接后的长
 度+1，多一个字符长度是留给'\0'使用
strcat(s1,s2);
puts(s1); // abcdef，最后一个字符是'\0'
puts(s2); // cdef，保持不变，，最后一个字符是'\0'
// 把s2复制给s1，因此s1数组也要足够大
strcpy(s1,s2);  // s1=s2; 是错误的，因为数组名表示地址常量
puts(s1); // cdef，最后一个字符是'\0'
puts(s2); // cdef，保持不变，，最后一个字符是'\0'
strncpy(s1,"1234",2); // 复制s2的前n个字符给s1,覆盖s1相应位置的字符，n要小
 于s1的数组长度
puts(s1); // 12ef
```

字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)

```
char s1[81]="ab"; // 自动追加s1[2]='\0'
char s2[81]="cdef"; // 自动追加s2[4]='\0'
// 从左到右逐字符比较串s1和s2，大于返回1，小于返回-1，同时到达'\0'返回0表示相
  等
printf("%d",strcmp(s1,s2)); // -1
if(strcmp(s1,s2)==-1) printf("s1␣<␣s2\n");
printf("%d\n",strcmp("abad","ab1d")); // 1
printf("%d\n",strcmp("1234","1234")); // 0
printf("%d\n",strcmp("12","1234")); // -1
printf("%d\n",strcmp("1234","12")); // 1
// 仅比较前n个字符
printf("%d\n",strncmp("1234","12",2)); // 0
```

获取字符串的长度 strlen(s)

```
char s1[81]="ab"; // 自动追加s1[2]='\0'
char s2[81]="cdef"; // 自动追加s2[4]='\0'
int i;
// 获取字符串的长度，不包括字符串结尾字符'\0'
printf("%d,%d\n",strlen(s1), strlen(s2)); // 2,4
for(i=0;i<strlen(s1);i++) printf("%c␣",s[i]); // a b
// 相当于下面程序段计算出的字符串长度len
int len=0;
for(i=0; s1[i]!='\0'; i++) len++;
printf("\n字符串s1的长度=%d\n", len);
```

大小写转换 strlwr(s); strupr(s)

```
char s1[81]="abCD12"; // 自动追加s1[2]='\0'
char s2[81]="cD12ef"; // 自动追加s2[4]='\0'
int i;
strlwr(s1); // 大写转小写
puts(s1); // abcd12

strupr(s2); // 小写转大写
puts(s2); // CD12EF

// 小写转大写，等效于下面程序段
for(i=0;s2[i]!='\0';i++)
  if(s2[i]>='a' && s2[i]<='z') s2[i]-=32;  // s2[i]=s2[i]-32
```

例 6.8 输入一行字符, 统计其中有多少个单词, 单词之间用空格 (可能多个) 分隔开。

```
// 定义变量
char string[81]; // 用于存放字符串。
int i; // 计数器，用于遍历字符串中的每个字符。
int word=0; // 用于判断是否开始了一个新单词的标志。若word=0表示未出现新单词，如出
    现了新单词，就把word置成1。
int num=0; //用于统计单词数。
```

构思程序流程 $\Longrightarrow$

例 6.8 输入一行字符, 统计其中有多少个单词, 单词之间用空格 (可能多个) 分隔开。

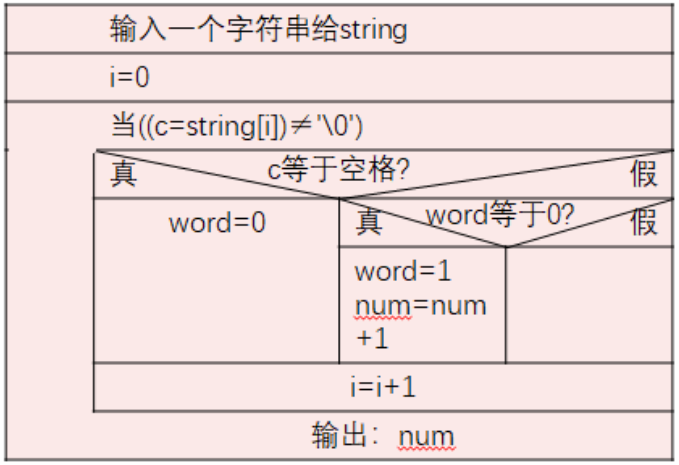
```
// 定义变量
char string[81]; // 用于存放字符串。
int i; // 计数器，用于遍历字符串中的每个字符。
int word=0; // 用于判断是否开始了一个新单词的标志。若word=0表示未出现新单词，如出
    现了新单词，就把word置成1。
int num=0; //用于统计单词数。
```

构思程序流程 $\Longrightarrow$

例 6.8 输入一行字符, 统计其中有多少个单词, 单词之间用空格 (可能多个) 分隔开。

```
// 定义变量
char string[81]; // 用于存放字符串。
int i; // 计数器, 用于遍历字符串中的每个字符。
int word=0; // 用于判断是否开始了一个新单词的标志。若word=0表示未出现新单词, 如出
    现了新单词, 就把word置成1。
int num=0; //用于统计单词数。
```

构思程序流程 $\Longrightarrow$



解: 例 6.8 输入一行字符, 统计其中有多少个单词, 单词之间用空格 (可能多个) 分隔开。

```
char string[81];
int i,num=0,word=0;
char c;
gets(string); //输入一个字符串给字符数组string
for(i=0;(c=string[i])!='\0';i++) //只要字符不是'\0'就循环
{
   if(c==' ') word=0; //若是空格字符，使word置0
   else if(word==0) //如果不是空格字符且word原值为0
   {
     word=1; //使word置1
     num++; //num累加1，表示增加一个单词
   }
}
printf("%d words in this line.\n",num); //输出单词数
```

例 6.9 有 3 个字符串, 要求找出其中 “最大” 者。

```
char str[3][20]; //定义二维字符数组，存放3个字符串。【重点学习】
char string[20]; //定义一维字符数组，作为交换字符串时的临时字符数组
int i;
for(i=0;i<3;i++)
   gets(str[i]); //读入3个字符串，分别给str[0],str[1],str[2]

if(strcmp(str[0],str[1])>0) //若str[0]大于str[1]
     strcpy(string,str[0]); //把str[0]的字符串赋给字符数组string
else //若str[0]小于等于str[1]
     strcpy(string,str[1]); //把str[1]的字符串赋给字符数组string
if(strcmp(str[2],string)>0) //若str[2]大于string
     strcpy(string,str[2]); //把str[2]的字符串赋给字符数组string
printf("\nthe␣largest␣string␣is:\n%s\n",string); //输出string
```

例 6.9 有 3 个字符串, 要求找出其中 “最大” 者。

```
char str[3][20]; //定义二维字符数组，存放3个字符串。【重点学习】
char string[20]; //定义一维字符数组，作为交换字符串时的临时字符数组
int i;
for(i=0;i<3;i++)
   gets(str[i]); //读入3个字符串，分别给str[0],str[1],str[2]

if(strcmp(str[0],str[1])>0) //若str[0]大于str[1]
     strcpy(string,str[0]); //把str[0]的字符串赋给字符数组string
else //若str[0]小于等于str[1]
     strcpy(string,str[1]); //把str[1]的字符串赋给字符数组string
if(strcmp(str[2],string)>0) //若str[2]大于string
     strcpy(string,str[2]); //把str[2]的字符串赋给字符数组string
printf("\nthe␣largest␣string␣is:\n%s\n",string); //输出string
```

另解: 例 6.9 有 3 个字符串, 要求找出其中 “最大” 者。

```
#include<stdio.h>
#include<string.h>
int main()
{
   char s[20],max[20];  int i;
   for(i=0;i<3;i++)
   {
      if(i==0)gets(max);
      else
      {
         gets(s);
         if(strcmp(s,max)>0) strcpy(max,s);
      }
   }
   printf("\nthe␣largest␣string␣is:\n%s\n",max);
   return 0;
}
```

例: 矩阵乘法

$$c[M][N], a[M][P], b[P][N];$$

$$c[i][j] = \sum_{k=0}^{P-1} a[i][k]b[k][j] = a[i][0]b[0][j] + a[i][1]b[1][j] + \cdots + a[i][P-1]b[P-1][j]$$

$$a = \begin{bmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \end{bmatrix} \quad b = \begin{bmatrix} b[0][0] & b[0][1] \\ b[1][0] & b[1][1] \\ b[2][0] & b[2][1] \end{bmatrix}$$

$$c = \begin{bmatrix} a[0][0]b[0][0] + a[0][1]b[1][0] + a[0][2]b[2][0] & a[0][0]b[0][1] + a[0][1]b[1][1] + a[0][2]b[2][1] \\ a[1][0]b[0][0] + a[1][1]b[1][0] + a[1][2]b[2][0] & a[1][0]b[0][1] + a[1][1]b[1][1] + a[1][2]b[2][1] \end{bmatrix}$$

c[i][j]=a 的第 i 行各列 (k) 和 b 的第 j 列各行 (k) 的乘积累加

例: 矩阵乘法

$$c[M][N], a[M][P], b[P][N];$$

$$c[i][j] = \sum_{k=0}^{P-1} a[i][k]b[k][j] = a[i][0]b[0][j] + a[i][1]b[1][j] + \cdots + a[i][P-1]b[P-1][j]$$

$$a = \begin{bmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \end{bmatrix} \quad b = \begin{bmatrix} b[0][0] & b[0][1] \\ b[1][0] & b[1][1] \\ b[2][0] & b[2][1] \end{bmatrix}$$

$$c = \begin{bmatrix} a[0][0]b[0][0] + a[0][1]b[1][0] + a[0][2]b[2][0] & a[0][0]b[0][1] + a[0][1]b[1][1] + a[0][2]b[2][1] \\ a[1][0]b[0][0] + a[1][1]b[1][0] + a[1][2]b[2][0] & a[1][0]b[0][1] + a[1][1]b[1][1] + a[1][2]b[2][1] \end{bmatrix}$$

c[i][j]=a 的第 i 行各列 (k) 和 b 的第 j 列各行 (k) 的乘积累加

解: 矩阵乘法

```
#define M 100 // 估计的最大值
#define P 50
#define N 100
int c[M][N],a[M][P]={{1,2,4}},b[P][N]={{2,1,3},{7,9,10},{5,7,1}}; // c
    ={36,47,27}
int m=4,n=2,p=3; // M,N,P实际大小
int i,j,k; //i,j,k分别是m,n,p的循环变量
for(i=0;i<m;i++)
{
   for(j=0;j<n;j++)
   {
     c[i][j]=0;
     for(k=0;k<p;k++)
        c[i][j]+=a[i][k]*b[k][j]; // a的第i行各列(k)和b的第j列各行(k)的乘积
    累加
     printf("c[%d][%d]=%d\n",i,j,c[i][j]); // 输出测试
   }
}
```

曾经的测试用例

```
//int c[M][N],a[M][P]={{1,2,4}},b[P][N]={{2,1,3},{7,9,10},{5,7,1}}; //
    c={36,47,27}
//int m=1,n=3,p=3; // M,N,P实际大小
//int c[M][N],a[M][P]={{2,1},{4,3}},b[P][N]={{1,2},{1,0}}; // c
    ={{3,4},{7,8}}
//int m=2,n=2,p=2; // M,N,P实际大小
//int c[M][N],a[M][P]={{1,0,2},{-1,3,1}},b[P][N]={{3,1},{2,1},{1,0}};
    // c={{5,1},{4,2}}
//int m=2,n=2,p=3; // M,N,P实际大小
int c[M][N],a[M][P]={{5,2,4},{3,8,2},{6,0,4},{0,1,6}},b[P][N
    ]={{2,4},{1,3},{3,2}}; // c={{24,34},{20,40},{24,32},{19,15}}
int m=4,n=2,p=3; // M,N,P实际大小
int i,j,k; //i,j,k分别是m,n,p的循环变量
// 添加输入语句
scanf("%d%d%d\n", &m,&n,&p);
for(i=0;i<m;i++)
  for(k=0;k<p;k++) scanf("%d",&a[i][k]);
for(k=0;k<p;k++)
  for(j=0;j<n;j++) scanf("%d",&b[k][j]);
...
```

lecture-12 主要内容

字符串处理库函数及程序举例

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- *while*(表达式){ · · · }
- *do*{ · · · }*while*(表达式);
- *for*(表达式 1; 表达式 2; 表达式 3){ · · · }
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6},{7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
  - 连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)
  - 字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)
  - 获取字符串的长度 strlen(s)
  - 大小写转换 strlwr(s); strupr(s)
- 程序举例
  - 统计单词数
  - 求多个字符串中的较大者
  - 矩阵乘法
- 使用函数进行模块化程序设计
  - 函数三要素: 函数原型声明, 函数定义, 函数调用
  - 函数定义: int add(int a,int b){ }
- 实参和形参间的数据传递 (值传递和地址传递)
- 程序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 整数 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例续

lecture-13 主要内容

用函数实现模块化程序设计

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- *while*(表达式){ · · · }
- *do*{ · · · }*while*(表达式);
- *for*(表达式 1; 表达式 2; 表达式 3){ · · · }
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6},{7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
  - 连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)
  - 字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)
  - 获取字符串的长度 strlen(s)
  - 大小写转换 strlwr(s); strupr(s)
- 程序举例
  - 统计单词数
  - 求多个字符串中的较大者
  - 矩阵乘法
- 使用函数进行模块化程序设计
  - 函数三要素: 函数原型声明, 函数定义, 函数调用
  - 函数定义: int add(int a,int b){ }
- 实参和形参间的数据传递 (值传递和地址传递)
- 程序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 整数 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例续

为什么使用函数 (1)

函数调用

```c
#include<stdio.h>
#include<math.h>
int main() // 主函数
{
    int a;
    // 库函数调用
    scanf("%d",&a);
    a=(int)fabs(a);
    printf("%d\n",a);
    return 0;
}
```

函数定义

```c
int scanf(char format[], args, ...)
{ ...;
  return 整型值;
}
int printf(char format[], args, ...)
{ ...;
  return 整型值;
}
double fabs(double a)
{ // 模拟代码，没有考虑精度
  if(a<0) return -a;
  else return a;
}
```

为什么使用函数 (2)

函数调用

```
#include<stdio.h>
#include<string.h>
int main() // 主函数
{
   int a;
   char s1[81],char s2[81]="1234";
   // 库函数调用
   strcpy(s1,s2);
   a=strcmp(s1,s2);
   printf("%d␣",a);
   printf("%d\n",strlen(s1));
   return 0;
}
```

函数定义

```
char[] strcpy(char s1[],char s2[])
{ ...; return s1;
}
int strcmp(char s1[],char s2[])
{  ...;  return 整型值; // 1,-1,0
}
int strlen(char s[])
{
  int len=0;
  while(s[len]!='\0') len++;
  return len;
}
```

功能分解是简化程序设计的有效手段

功能分解, 函数调用

```
#include<stdio.h>
#include<math.h>
int add(int a,int b); // 函数原型声明
void output(double a);
int main() // 主函数
{
  int a=10,b=20,c;
  //传递参数a,b的值, 计算结果赋值给变量c
  c=add(a,b);  // 函数调用
  output(10.5*c); // 函数调用
  return 0;
}
```

分解功能实现, 函数定义

```
// 通过参数a,b的值, 进行相关计算, 返回整
    型数据给调用者
int add(int a,int b)
{  return a+b; // 返回整型值 }
// 通过参数a的值, 进行相关计算, 不需要返
    回数据
void output(double a)
{
   printf("%lf\n", sqrt(a)); // 函数
    中可调用别的函数
}
```

使用函数进行程序设计的优点

1. 使用函数可使程序清晰、精炼、简单、灵活。
2. 函数就是功能。每一个函数用来实现一个特定的功能。函数名应反映其代表的功能。
3. 在设计较大程序时,往往把它分为若干个程序模块,每一个模块包括一个或多个函数,每个函数实现一个特定的功能。
4. 一个 C 程序可由一个主函数和若干个其他函数构成。由主函数调用其他函数,其他函数也可以互相调用。

函数三要素: 函数原型声明, 函数定义, 函数调用

函数原型声明, 函数调用

```
#include<stdio.h> // 库函数原型声明
// 函数原型声明, 使编译器认识这个函数, 如
    果函数定义在调用之前, 可省略声明
int add(int a,int b);
void output(double a);
int main() // 主函数
{
   int a=10,b=20,c;
   // 函数调用, 执行函数功能
   c=add(a,b);
   output(10.5*c);
   return 0;
}
```

函数定义, 定义函数功能

```
// 通过参数a,b的值, 进行相关计算, 返回整
    型数据给调用者
int add(int a,int b)
{  return a+b; // 返回整型值 }

// 通过参数a的值, 进行相关计算, 不需要返
    回数据
void output(double a)
{
   printf("%lf\n", sqrt(a)); // 函数
    中可调用别的函数
}
```

函数定义: int add(int a,int b){ }

函数定义: 返回类型 函数名 (参数类型 参数, · · ·) {· · · ; }

```
int add(int a, int b)
{   ...;
    return 整型值; // 必须含return语句, 函数执行结束, 并将返回值返回给调用者
}
double fun(void) // 无参函数, 等效 double fun()
{   ...;
    return 双精度值; // 必须含return语句, 函数执行结束, 并将返回值返回给调用者
}
void output(double a) // 无返回值函数
{   ...;
    return; // 可选return语句(注意没有表达式), 仅表示函数执行结束
}
```

函数调用时数据类型的隐式转换

```
int a=2,b=3,c;
// pow函数原型: double pow(double x,double y);
// 编译器自动把a,b"隐式"转换为double
// 计算a^b的结果是doublele类型, 赋值语句隐式转换为int, 但是会引起警告信息
c=pow(a,b);
c=(int)pow(a,b); // 将函数的返回值强制转换为int, 不会有警告信息
```

实参和形参间的数据传递 (值传递)

```c
#include<stdio.h> // 库函数原型声明
// 函数原型声明，使编译器认识这个函数
int max(int x,int y);
int main() // 主函数
{
    int a=10,b=20,c;
    scanf("%d%d",&a,&b);
    // 把此处的a,b值拷贝给函数形式参数x,y
    c=max(a,b); // 实际参数
    printf("较大者=%d\n",c);//与下一句
      等效
    printf("较大者=%d\n",max(a,b));
    return 0;
}
```

```c
// 定义函数求形参x,y中的较大者并返回给调
    用者
int max(int x,int y) // 形式参数
{
    int z;
    z=x>y ? x : y;
    return z;
}
```

在调用函数过程中发生的实参与形参间的数据传递称为“虚实结合”。

实参和形参间的数据传递 (地址传递)

例: 将数组 a 中 n 个整数按相反顺序存放。

```c
#include<stdio.h>
void inv(int x[],int n); // 要求x是地址传递
#define N 100
int main()
{
    int i, n, a[N]={1,2,3,4,5,6,7,8,9,10};
    scanf("%d",&n);
    for(i=0; i<n; i++) scanf("%d",&a[i]);
    // 把实参a数组的地址拷贝给形参x, n的值拷贝给形式参数n
    inv(a,n); // 实参a数组的内容被改变
    for(i=0; i<n; i++) printf("%d␣",a[i]);
    return 0;
}
```

```c
// 倒置数组x的内容, n是x的长度
// 要求x是地址传递
void inv(int x[],int n)
{
    int i,temp,m=(n-1)/2;
    //以中间元素为界, 前后元素交换
    for(i=0; i<=m; i++)
    {
        temp=x[i];
        x[i]=x[n-1-i];
        x[n-1-i]=temp;
    }
    return; //可选, 函数执行完毕
}
```

值传递与地址传递的不同点

- 值传递, 对形参值的改变不会引起实参值的改变。
- 地址传递虽然不能改变实参的地址, 但是对地址指向内容的改变会引起实参指向内容的改变。
- 数组名表示数组元素在内存中的首地址, 数组元素是连续存放的。
- 用数组名作为参数传递, 就是地址传递。在函数内部对数组元素的改变, 就是改变实参数组元素。

```
void fun(double a[],int n) // a是地址传递, n是值传递
{
    a[1]=20.5; // 改变地址a指向的内容, 就是改变实参数组的元素值
    n=30; // 此处的改变不会影响实参的值
}
int main()
{
   double a[2]={0.8,0.3};
   int n=2;
   fun(a,n); // 实参a的地址拷贝给形参a,实参n的值拷贝给形参n
   printf("%d,%lf,%lf\n",n,a[0],a[1]); // 2,0.8,20.5
}
```

数组名作为函数参数的注意事项

```c
#define M 100 // 估计的数组的第一维长度
#define N 100 // 估计的数组的第二维长度
// 函数定义时省略第一维大小
// a[]表示a接受地址传递，以区别于值传递
void fun(int a[],int n)
{
   int i;
   for(i=0;i<n;i++)
     printf("%d␣"a[i]);
   printf("\n");
}
// 函数调用
int x[M]={1, 2, 3, 4, 5};
fun(x,5);//调用时仅用数组名传递x的地址
```

```c
// 函数定义时省略第一维大小,第二维不能省略
void fun(int a[][N],int m,int n)
{
  int i;
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
      printf("%d␣"a[i][j]);
    printf("\n");
  }
}
// 函数调用
int x[M][N]={{1,2},{3,4},{5,6},
             {7,8},{9,10}};
fun(x,5,2);//调用时仅用数组名传递x的地址
```

例: PM2.5

给出一组 PM2.5 数据,按以下分级标准统计各级天气的天数,并计算出 PM2.5 平均值。

PM2.5 分级标准为:

一级优 $(0 \leq PM2.5 \leq 50)$

二级良 $(51 \leq PM2.5 \leq 100)$

三级轻度污染 $(101 \leq PM2.5 \leq 150)$

四级中度污染 $(151 \leq PM2.5 \leq 200)$

五级重度污染 $(201 \leq PM2.5 \leq 300)$

六级严重污染 $(PM2.5 > 300)$

输入说明, 输入分为两行,

第一行是一个整数 n 表示天数 $(1 < n \leq 100)$

第二行为 n 个非负整数 $Pi(0 \leq Pi \leq 1000)$ 表示每天的 PM2.5 值,整数之间用空格分隔。

输出说明, 输出两行数据,

第一行为 PM2.5 平均值, 结果保留 2 位小数;

第二行依次输出一级优, 二级良, 三级轻度污染, 四级中度污染, 五级重度污染, 六级严重污染的天数。

例: PM2.5(非模块化设计)

```
#include <stdio.h>
int main()
{
  int i =0,n,pm25,day[6] = {0,0,0,0,0,0},sum = 0;
  scanf("%d",&n);
  while(i < n)
  {
    scanf("%d",&pm25);
    sum += pm25;
    if(pm25 >= 0 && pm25 <= 50 ) day[0]++;
    else if(pm25 >= 51 && pm25 <= 100 ) day[1]++;
    else if(pm25 >= 101 && pm25 <= 150 ) day[2]++;
    else if(pm25 >= 151 && pm25 <= 200 ) day[3]++;
    else if(pm25 >= 201 && pm25 <= 300 ) day[4]++;
    else day[5]++;
    i++;
  }
  printf("%.2f\n",(float)sum/n);
  for(i = 0; i < 6; i++)
    if(i == 5) printf("%d\n",day[i]);
    else  printf("%d␣",day[i]);
  return 0;
}
```

例: PM2.5(模块化设计, 主程序)

```
#include <stdio.h>
float haze(int pm25[], int day[], int n, int m);// 函数声明
#define N 1000 // 估计的最大值
int main()
{
   int i =0,n,pm25[N],day[6];
   scanf("%d",&n);
   while(i < n)
   {
     scanf("%d",&pm25[i]);
     i++;
   }
   printf("%.2f\n",haze(pm25,day,n,6)); // 函数调用
   for(i = 0; i < 6; i++)
      if(i == 5) printf("%d\n",day[i]);
      else  printf("%d␣",day[i]);
   return 0;
}
```

例: PM2.5(模块化设计, 雾霾统计信息)

```
// pm25[]: PM2.5值, day[]: 不同标准对应的统计天数;
// n是pm25数组的长度, m是数组day的长度, 返回PM2.5平均值
float haze(int pm25[], int day[], int n, int m)
{
  int i=0, sum=0;
  // 初始化day数组
  for(i=0; i<m; i++) day[i]=0;
  while(i < n)
  {
    sum += pm25[i];
    if(pm25[i] >= 0 && pm25[i] <= 50 ) day[0]++;
    else if(pm25[i] >= 51 && pm25[i] <= 100 ) day[1]++;
    else if(pm25[i] >= 101 && pm25[i] <= 150 ) day[2]++;
    else if(pm25[i] >= 151 && pm25[i] <= 200 ) day[3]++;
    else if(pm25[i] >= 201 && pm25[i] <= 300 ) day[4]++;
    else day[5]++;
    i++;
  }
  return (float)sum/n;
}
```

用数组做参数, 可获得函数计算结果的多值传递, return 仅返回一个值。

例: 数字排序 (主程序)

给定 n 个整数,请计算每个整数各位数字和,按各位数字和从大到小的顺序输出。

```
#include <stdio.h>
#define N 1000 // 估计的数组大小
int bitsSum(int a); // 计算整数a的各位之和
void sort(int a[], int n); // 从大到小排序
void output(int a[], int n); // 输出
int main()
{
   int i,n; // n是实际数组长度
   int num[N],sum[N]; // num表示N个整数, sum存放对应整数的各位数字的和
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      scanf("%d",&num[i]);
      sum[i]=bitsSum(num[i]);
   }
   sort(sum,n);  // 从大到小排序
   output(sum,n); // 输出
   return 0;
}
```

例: 数字排序 (子函数: 计算整数的各位之和, 输出)

```
// 计算整数a的各位之和
int bitsSum(int a)
{
  int sum=0;
  while(a)
  {
    sum += a%10;
    a /= 10;
  }
  return sum;
}
// 输出
void output(int a[], int n)
{
    int i;
    for(i=0;i<n;i++) printf("%d␣",a[i]);
    printf("\n");
}
```

例: 数字排序 (排序子函数—方法 1: 冒泡排序)

```
// 从大到小排序, 冒泡
 void sort(int a[], int n)
{
   int i,j,flag,temp;
   for(j = 1; j <= n-1; j++) // 第j趟比较
   {
      flag=0;
      for(i = 0; i < n - j; i++) // 相邻两数比较
      {
        if (a[i] < a[i+1]) // 交换
        {
           temp = a[i]; a[i] = a[i+1]; a[i+1] = temp;
           flag=1;
        }
      }
      if(!flag) break;
   }
}
```

例: 数字排序 (排序子函数—方法 2: 选择法排序)

从大到小**选择法排序**: 先将 n 个数中最大的数与 $a[0]$ 对换;

再将 $a[1] \sim a[n-1]a[1]$ 对换, $\cdots$, 每比较一轮, 找出一个未经排序的数中最大的一个。共比较 $n-1$ 轮。

```
// 从大到小排序, 选择
void sort(int a[], int n)
{
   int i,j,k,temp;
   for(j = 0; j <= n-2; j++) // 第j趟比较
   {
     k=j; // 未经排序的数中, 最大元素的下标
     for(i = j+1; i < n; i++) // 未经排序的数从j+1开始
     {
         if (a[i] > a[k]) k=i;
     }
     if(k!=j)//未经排序数中的最大元素与a[j]交换, a[j]及其以前元素是已经排好的数据
     {
        temp = a[j]; a[j] = a[k]; a[k] = temp;
     }
   }
}
```

例: 数字排序 (联动输出: 修改主程序)

给定 n 个整数,请计算每个整数各位数字和,按各位数字和从大到小的顺序输出。

要求联动输出: 整数　该整数的各位之和

```
#include <stdio.h>
#define N 1000 // 估计的数组大小
int bitsSum(int a); // 计算整数a的各位之和
void sort(int a[], int b[], int n);  // a从大到小排序, b联动改变
void output(int a[], int b[], int n); // 输出a,b
int main()
{
   int i,n; // n是实际数组长度
   int num[N],sum[N]; // num表示N个整数, sum存放对应整数的各位数字的和
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      scanf("%d",&num[i]);
      sum[i]=bitsSum(num[i]);
   }
   sort(sum,num,n);  // sum从大到小排序,num联动改变
   output(num,sum,n); // 输出num,sum
   return 0;
}
```

例: 数字排序 (联动输出: 修改排序子函数)

```
// a从大到小排序, b联动改变
void sort(int a[], int b[],int n)
{
   int i,j,flag,temp;
   for(j = 1; j <= n-1; j++) // 第j趟比较
   {
       flag=0;
       for(i = 0; i < n - j; i++) // 相邻两数比较
       {
          if (a[i] < a[i+1]) // 同时交换a和b
          {
             temp = a[i]; a[i] = a[i+1]; a[i+1] = temp;
             temp = b[i]; b[i] = b[i+1]; b[i+1] = temp;
             flag=1;
          }
       }
       if(!flag) break;
    }
}
```

例: 数字排序 (联动输出: 修改输出子函数)

```
// 输出a,b
void output(int a[], int b[], int n)
{
    int i;
    for(i=0;i<n;i++) printf("%d␣%d",a[i],b[i]);
    printf("\n");
}
```

例: 数字排序 (联动输出: 修改主程序, 二维数组)

给定 n 个整数,请计算每个整数各位数字和,按各位数字和从大到小的顺序输出。

要求联动输出: 整数　该整数的各位之和

```
#include <stdio.h>
#define N 1000 // 估计的数组大小
int bitsSum(int a); // 计算整数a的各位之和
void sort(int a[][2], int n);  // 按a的第0列从大到小排序
void output(int a[][2], int n); // 输出a
int main()
{
   int i,n; // n是实际数组长度
   int num[N][2]; // 第0列表示整数, 第1列是该整数的各位数字的和
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
     scanf("%d",&num[i][0]);
     num[i][1]=bitsSum(num[i][0]);
   }
   sort(num,n);  // sum从大到小排序,num联动改变
   output(num,n); // 输出num
   return 0;
}
```

例: 数字排序 (联动输出: 修改排序子函数, 二维数组)

```
// 按a的第0列从大到小排序
void sort(int a[][2],int n)
{
   int i,j,flag,temp;
   for(j = 1; j <= n-1; j++) // 第j趟比较
   {
      flag=0;
      for(i = 0; i < n - j; i++) // 相邻两数比较
      {
          if (a[i][0] < a[i+1][0]) // 同时交换a的第0列和第1列
          {
            temp = a[i][0]; a[i][0] = a[i+1]; a[i+1][0] = temp;
            temp = a[i][1]; a[i][1] = a[i+1][1]; b[i+1][1] = temp;
            flag=1;
          }
      }
      if(!flag) break;
   }
}
```

例: 数字排序 (联动输出: 修改输出子函数, 二维数组)

```
// 输出a
void output(int a[][2], int n)
{
    int i;
    for(i=0;i<n;i++) printf("%d␣%d",a[i][0],a[i][1]);
    printf("\n");
}
```

模块化程序设计的优点

综上,程序功能的变化,仅修改相应子函数即可,逻辑清晰。

因此,功能分解是进行复杂程序设计的有效手段。

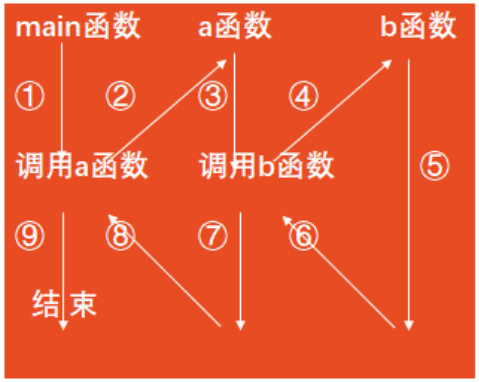
lecture-14 主要内容

用函数实现模块化程序设计 (嵌套与递归)

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- *while*(表达式){ · · · }
- *do*{ · · · }*while*(表达式);
- *for*(表达式 1; 表达式 2; 表达式 3){ · · · }
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6},{7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
  - 连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)
  - 字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)
  - 获取字符串的长度 strlen(s)
  - 大小写转换 strlwr(s); strupr(s)
- 程序举例
  - 统计单词数
  - 求多个字符串中的较大者
  - 矩阵乘法
- 使用函数进行模块化程序设计
  - 函数三要素: 函数原型声明, 函数定义, 函数调用
  - 函数定义: int add(int a,int b){ }
- 实参和形参间的数据传递 (值传递和地址传递)
- 程序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 整数 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例续

函数的嵌套调用

```c
#include<stdio.h>
int a(); int b(); // 函数声明
int main()
{
    int c; // 与a()中的c无关
    c=a(); // 函数调用
    return 0;
}
```




```c
int a()
{
    int c; // 与main()中的c无关
    ...;
    c=b();// 函数调用
    ...;
    return c;
}
int b()
{
    ...;
    return 10;
}
```

例: 求 4 个整数中的最大者。

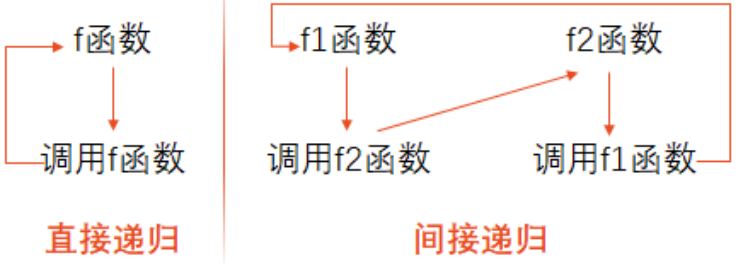
```c
#include<stdio.h>
int max2(int x,int y);
int max4(int a,int b,int c,int d);
int main() // 主函数
{
    int a=b,c,d;
    scanf("%d%d%d%d",&a,&b,&c,&d);
    printf("较大者=%d\n", max4(a,b,c,
     d));
    printf("较大者=%d\n", max2(max2(a
     ,b),max2(c,d))); // 等效
    return 0;
}
```

```c
// 定义函数
int max2(int x,int y) // 形式
     参数
{
    int z;
    z=x>y ? x : y;
    return z;
}
int max4(int a,int b,int c,
     int d)
{
   return max2(max2(a,b),max2
     (c,d));
}
```

函数的递归调用

在调用一个函数的过程中又出现直接或间接地调用该函数本身，称为函数的递归调用。

```
// 递推公式: f(0)=0,f(1)=1,
//      n>1: f(n)=n+f(n-1)
int f(int n)
{
    int sum;
    if(n==0||n==1) sum=n;
    else sum=n+f(n-1);
    return sum;
}
```

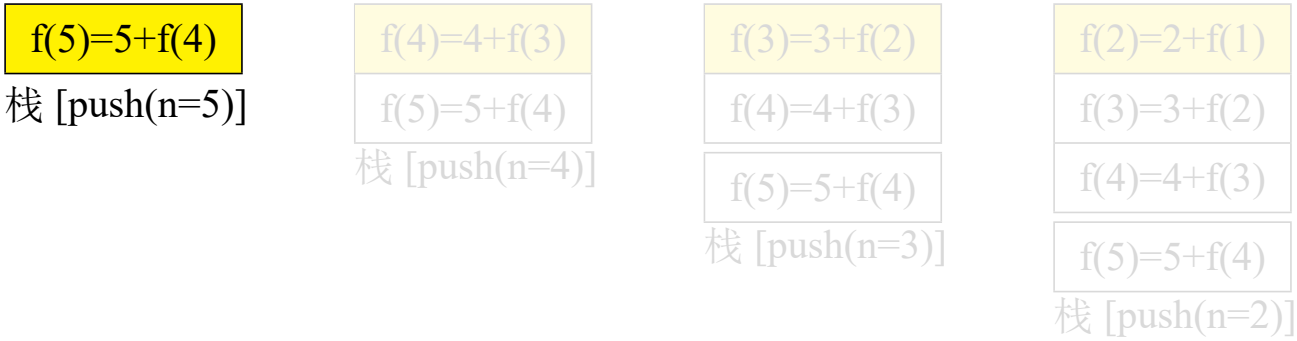


程序中不应出现无终止的递归调用，而只应出现**有限次数的，有终止的递归调用**，这可以用 if 语句来控制，只有在某一条件成立时才继续执行递归调用；否则就不再继续。

递归调用过程分析 (push)

```
// 递推公式: f(0)=0,f(1)=1, n>1: f(n)=n+f(n-1)
int f(int n)
{
  int sum;
  if(n==0||n==1) sum=n;
  else sum=n+f(n-1); // 未完成的计算用"栈"存储起来(push)
  return sum; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

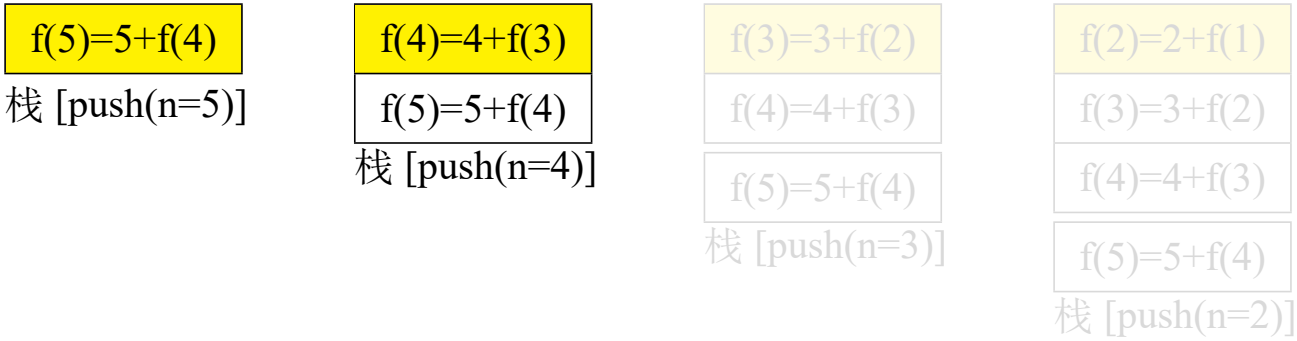
系统内部自动维护一个称作 “栈” 的存储数据的空间,栈是一种 “先进后出 (FILO)= 后进先出 (LIFO)” 的数据结构。向栈中存储数据操作称作 push, 取出栈顶数据操作称作 pop。第一个 push 的数据,最后一个被 pop.



递归调用过程分析 (push)

```
// 递推公式: f(0)=0,f(1)=1, n>1: f(n)=n+f(n-1)
int f(int n)
{
  int sum;
  if(n==0||n==1) sum=n;
  else sum=n+f(n-1); // 未完成的计算用"栈"存储起来(push)
  return sum; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

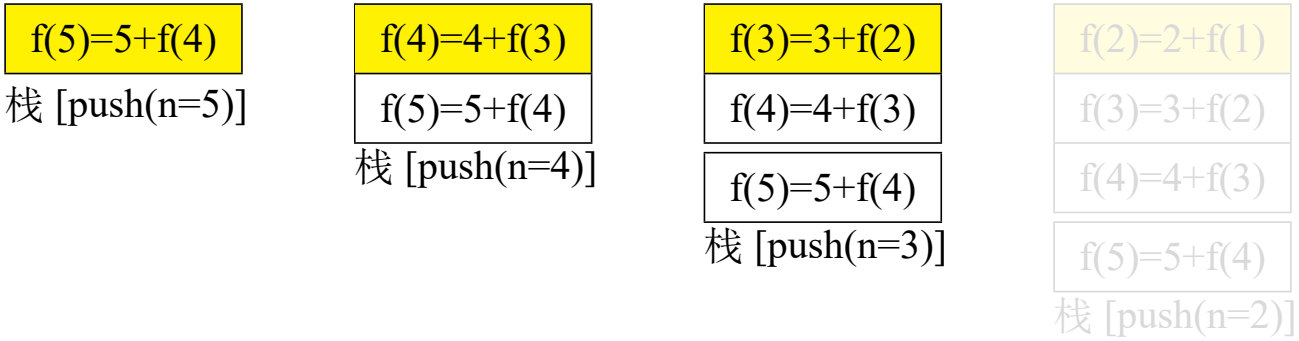
系统内部自动维护一个称作 “栈” 的存储数据的空间,栈是一种 “先进后出 (FILO)= 后进先出 (LIFO)” 的数据结构。向栈中存储数据操作称作 push, 取出栈顶数据操作称作 pop。第一个 push 的数据,最后一个被 pop.



递归调用过程分析 (push)

```
// 递推公式: f(0)=0,f(1)=1, n>1: f(n)=n+f(n-1)
int f(int n)
{
  int sum;
  if(n==0||n==1) sum=n;
  else sum=n+f(n-1); // 未完成的计算用"栈"存储起来(push)
  return sum; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

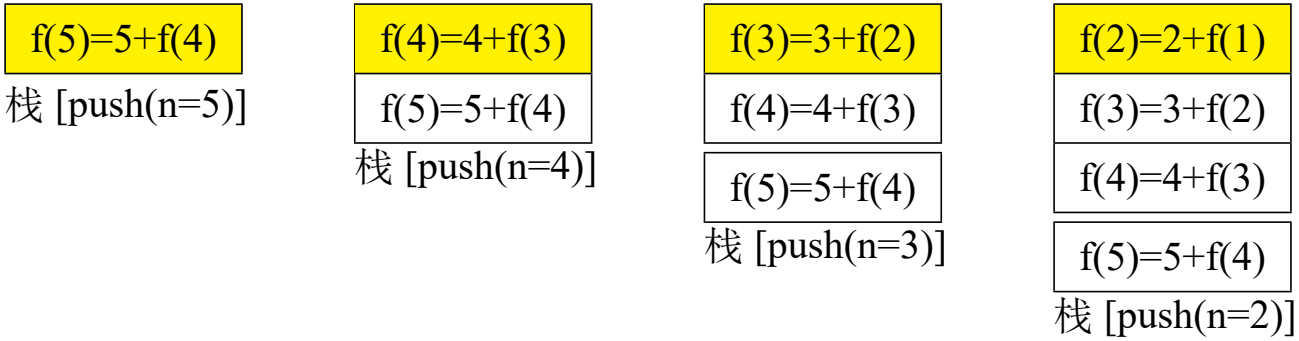
系统内部自动维护一个称作 “栈” 的存储数据的空间,栈是一种 “先进后出 (FILO)= 后进先出 (LIFO)” 的数据结构。向栈中存储数据操作称作 push, 取出栈顶数据操作称作 pop。第一个 push 的数据,最后一个被 pop.



递归调用过程分析 (push)

```
// 递推公式: f(0)=0,f(1)=1, n>1: f(n)=n+f(n-1)
int f(int n)
{
  int sum;
  if(n==0||n==1) sum=n;
  else sum=n+f(n-1); // 未完成的计算用"栈"存储起来(push)
  return sum; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

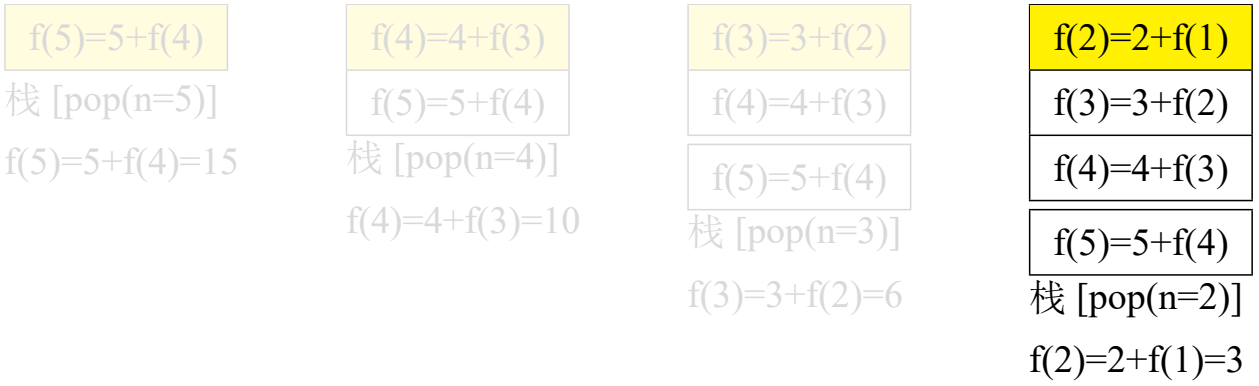
系统内部自动维护一个称作 “栈” 的存储数据的空间,栈是一种 “先进后出 (FILO)= 后进先出 (LIFO)” 的数据结构。向栈中存储数据操作称作 push, 取出栈顶数据操作称作 pop。第一个 push 的数据,最后一个被 pop.



递归调用过程分析 (pop)

```
// 递推公式: f(0)=0,f(1)=1, n>1: f(n)=n+f(n-1)
int f(int n)
{
  int sum;
  if(n==0||n==1) sum=n;
  else sum=n+f(n-1); // 未完成的计算用"栈"存储起来(push)
  return sum; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

系统内部自动维护一个称作 “栈” 的存储数据的空间,栈是一种 “先进后出 (FILO)= 后进先出 LIFO” 的数据结构。向栈中存储数据操作称作 push, 取出数据操作称作 pop。第一个 push 的数据,最后一个被 pop.



递归调用过程分析 (pop)

```
// 递推公式: f(0)=0,f(1)=1, n>1: f(n)=n+f(n-1)
int f(int n)
{
  int sum;
  if(n==0||n==1) sum=n;
  else sum=n+f(n-1); // 未完成的计算用"栈"存储起来(push)
  return sum; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

系统内部自动维护一个称作 “栈” 的存储数据的空间,栈是一种 “先进后出 (FILO)= 后进先出 LIFO” 的数据结构。向栈中存储数据操作称作 push, 取出数据操作称作 pop。第一个 push 的数据,最后一个被 pop.

f(5)=5+f(4)
栈 [pop(n=5)]
f(5)=5+f(4)=15

f(4)=4+f(3)
f(5)=5+f(4)
栈 [pop(n=4)]
f(4)=4+f(3)=10

f(3)=3+f(2)
f(4)=4+f(3)
f(5)=5+f(4)
栈 [pop(n=3)]
f(3)=3+f(2)=6

f(2)=2+f(1)
f(3)=3+f(2)
f(4)=4+f(3)
f(5)=5+f(4)
栈 [pop(n=2)]
f(2)=2+f(1)=3

递归调用过程分析 (pop)

```
// 递推公式: f(0)=0,f(1)=1, n>1: f(n)=n+f(n-1)
int f(int n)
{
  int sum;
  if(n==0||n==1) sum=n;
  else sum=n+f(n-1); // 未完成的计算用"栈"存储起来(push)
  return sum; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

系统内部自动维护一个称作 “栈” 的存储数据的空间,栈是一种 “先进后出 (FILO)= 后进先出 LIFO” 的数据结构。向栈中存储数据操作称作 push, 取出数据操作称作 pop。第一个 push 的数据,最后一个被 pop.

f(5)=5+f(4)
栈 [pop(n=5)]
f(5)=5+f(4)=15

f(4)=4+f(3)
f(5)=5+f(4)

栈 [pop(n=4)]
f(4)=4+f(3)=10

f(3)=3+f(2)
f(4)=4+f(3)
f(5)=5+f(4)

栈 [pop(n=3)]
f(3)=3+f(2)=6

f(2)=2+f(1)
f(3)=3+f(2)
f(4)=4+f(3)
f(5)=5+f(4)

栈 [pop(n=2)]
f(2)=2+f(1)=3

递归调用过程分析 (pop)

```
// 递推公式: f(0)=0,f(1)=1, n>1: f(n)=n+f(n-1)
int f(int n)
{
  int sum;
  if(n==0||n==1) sum=n;
  else sum=n+f(n-1); // 未完成的计算用"栈"存储起来(push)
  return sum; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

系统内部自动维护一个称作 “栈” 的存储数据的空间,栈是一种 “先进后出 (FILO)= 后进先出 LIFO” 的数据结构。向栈中存储数据操作称作 push, 取出数据操作称作 pop。第一个 push 的数据,最后一个被 pop.

f(5)=5+f(4)

栈 [pop(n=5)]

f(5)=5+f(4)=15

f(4)=4+f(3)
f(5)=5+f(4)

栈 [pop(n=4)]

f(4)=4+f(3)=10

f(3)=3+f(2)
f(4)=4+f(3)
f(5)=5+f(4)

栈 [pop(n=3)]

f(3)=3+f(2)=6

f(2)=2+f(1)
f(3)=3+f(2)
f(4)=4+f(3)
f(5)=5+f(4)

栈 [pop(n=2)]

f(2)=2+f(1)=3

例: age(n)

有 5 个学生坐在一起,问第 5 个学生多少岁,他说比第 4 个学生大 2 岁。问第 4 个学生岁数,他说比第 3 个学生大 2 岁。问第 3 个学生,又说比第 2 个学生大 2 岁。问第 2 个学生,说比第 1 个学生大 2 岁。最后问第 1 个学生,他说是 10 岁。请问第 5 个学生多大。

$$\text{第 n 个学生年龄}\begin{cases} age(n) = 10 & (n = 1) \\ age(n) = age(n-1) + 2 & (n > 1) \end{cases}$$

例: age(n)

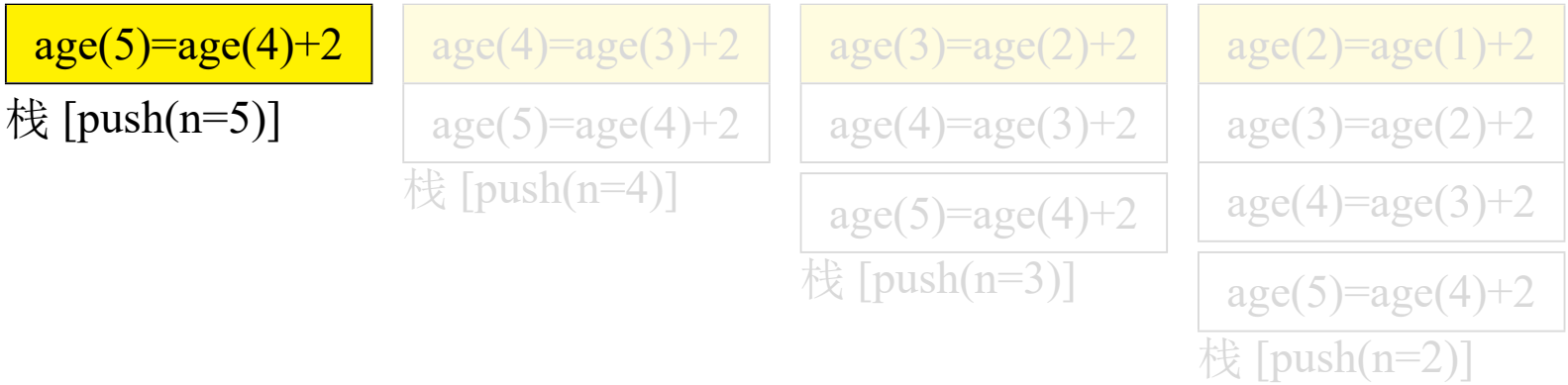
有 5 个学生坐在一起,问第 5 个学生多少岁,他说比第 4 个学生大 2 岁。问第 4 个学生岁数,他说比第 3 个学生大 2 岁。问第 3 个学生,又说比第 2 个学生大 2 岁。问第 2 个学生,说比第 1 个学生大 2 岁。最后问第 1 个学生,他说是 10 岁。请问第 5 个学生多大。

$$\text{第 n 个学生年龄}\begin{cases} age(n) = 10 & (n = 1) \\ age(n) = age(n-1) + 2 & (n > 1) \end{cases}$$

例: age(n), 递归调用过程分析 (push)

```
// 递推公式: age(1)=10; n>1: age(n)=age(n-1)
int age(int n)
{
  int y;
  if(n==1) y=10;
  else y=age(n-1)+2; // 未完成的计算用"栈"存储起来(push)
  return y; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

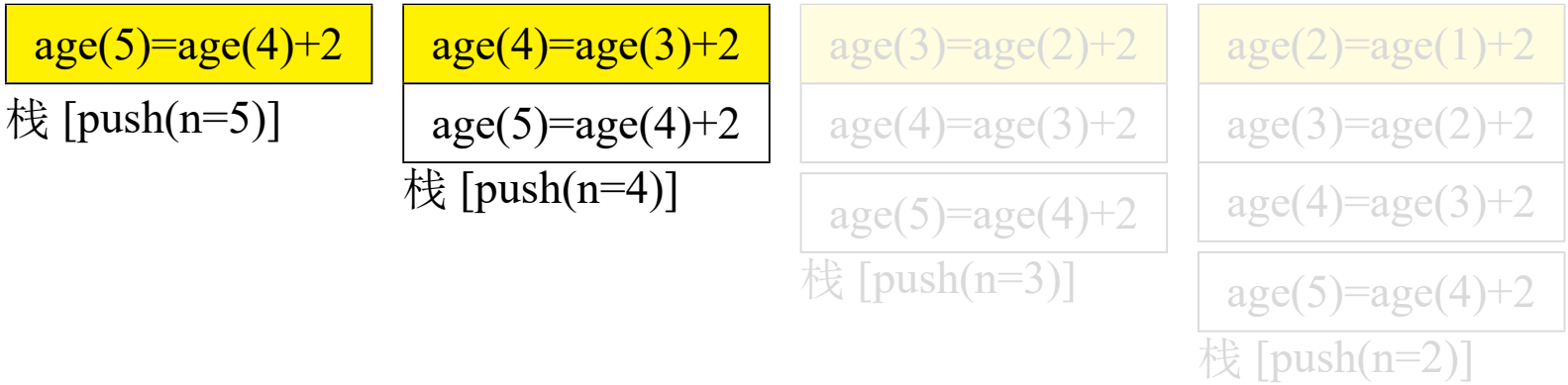
压栈: push



例: age(n), 递归调用过程分析 (push)

```
// 递推公式: age(1)=10; n>1: age(n)=age(n-1)
int age(int n)
{
  int y;
  if(n==1) y=10;
  else y=age(n-1)+2; // 未完成的计算用"栈"存储起来(push)
  return y; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

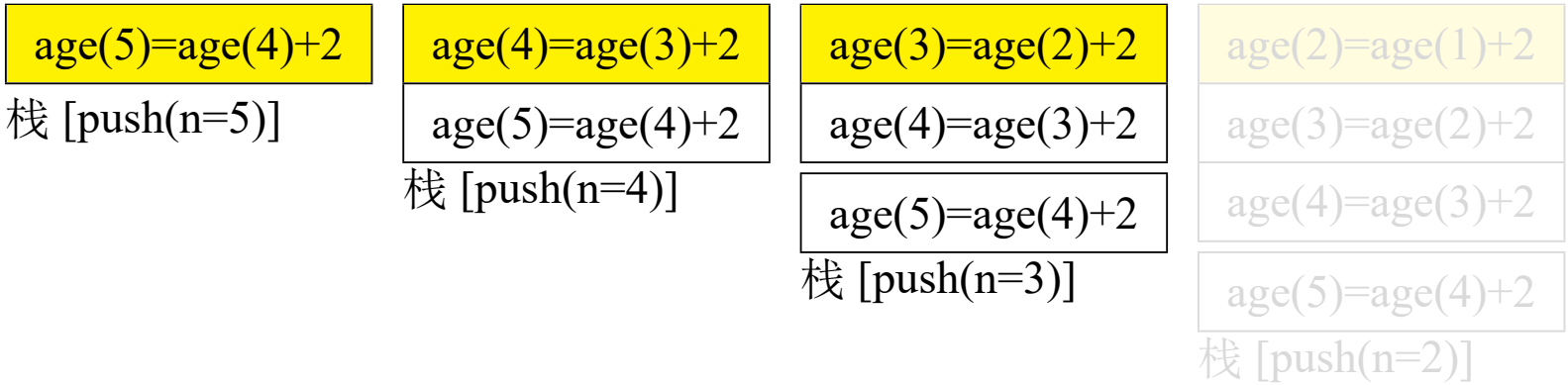
压栈: push



例: age(n), 递归调用过程分析 (push)

```
// 递推公式: age(1)=10; n>1: age(n)=age(n-1)
int age(int n)
{
  int y;
  if(n==1) y=10;
  else y=age(n-1)+2; // 未完成的计算用"栈"存储起来(push)
  return y; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

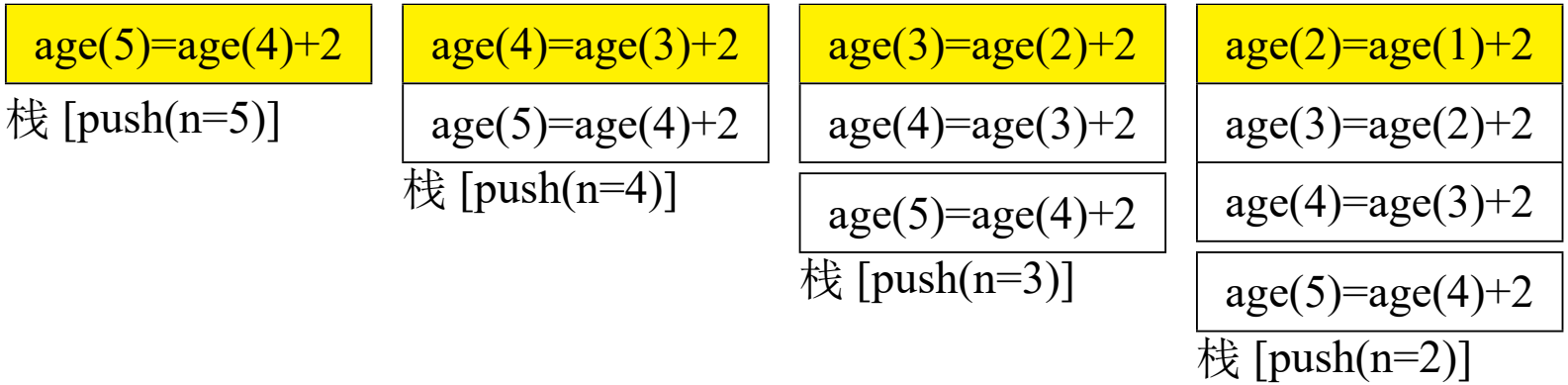
压栈: push



例: age(n), 递归调用过程分析 (push)

```
// 递推公式: age(1)=10; n>1: age(n)=age(n-1)
int age(int n)
{
  int y;
  if(n==1) y=10;
  else y=age(n-1)+2; // 未完成的计算用"栈"存储起来(push)
  return y; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

压栈: push



例: age(n), 递归调用过程分析 (pop)

```
// 递推公式: age(1)=10; n>1: age(n)=age(n-1)
int age(int n)
{
  int y;
  if(n==1) y=10;
  else y=age(n-1)+2; // 未完成的计算用"栈"存储起来(push)
  return y; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

弹出: pop

age(5)=age(4)+2
栈 [pop(n=5)]
age(5)=age(4)+2=18

age(4)=age(3)+2
age(5)=age(4)+2
栈 [pop(n=4)]
age(4)=age(3)+2=16

age(3)=age(2)+2
age(4)=age(3)+2
age(5)=age(4)+2
栈 [pop(n=3)]
age(3)=age(2)+2=14

age(2)=age(1)+2
age(3)=age(2)+2
age(4)=age(3)+2
age(5)=age(4)+2
栈 [pop(n=2)]
age(2)=age(1)+2=12

例: age(n), 递归调用过程分析 (pop)

```
// 递推公式: age(1)=10; n>1: age(n)=age(n-1)
int age(int n)
{
  int y;
  if(n==1) y=10;
  else y=age(n-1)+2; // 未完成的计算用"栈"存储起来(push)
  return y; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

弹出: pop

age(5)=age(4)+2

栈 [pop(n=5)]

age(5)=age(4)+2=18

age(4)=age(3)+2

age(5)=age(4)+2

栈 [pop(n=4)]

age(4)=age(3)+2=16

age(3)=age(2)+2

age(4)=age(3)+2

age(5)=age(4)+2

栈 [pop(n=3)]

age(3)=age(2)+2=14

age(2)=age(1)+2

age(3)=age(2)+2

age(4)=age(3)+2

age(5)=age(4)+2

栈 [pop(n=2)]

age(2)=age(1)+2=12

例: age(n), 递归调用过程分析 (pop)

```
// 递推公式: age(1)=10; n>1: age(n)=age(n-1)
int age(int n)
{
  int y;
  if(n==1) y=10;
  else y=age(n-1)+2; // 未完成的计算用"栈"存储起来(push)
  return y; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

弹出: pop

age(5)=age(4)+2

栈 [pop(n=5)]

age(5)=age(4)+2=18

age(4)=age(3)+2
age(5)=age(4)+2

栈 [pop(n=4)]

age(4)=age(3)+2=16

age(3)=age(2)+2
age(4)=age(3)+2
age(5)=age(4)+2

栈 [pop(n=3)]

age(3)=age(2)+2=14

age(2)=age(1)+2
age(3)=age(2)+2
age(4)=age(3)+2
age(5)=age(4)+2

栈 [pop(n=2)]

age(2)=age(1)+2=12

例: age(n), 递归调用过程分析 (pop)

```
// 递推公式: age(1)=10; n>1: age(n)=age(n-1)
int age(int n)
{
  int y;
  if(n==1) y=10;
  else y=age(n-1)+2; // 未完成的计算用"栈"存储起来(push)
  return y; // 函数return前, 从"栈"顶取数据,计算,直到"栈"空
}
```

弹出: pop

age(5)=age(4)+2

栈 [pop(n=5)]

age(5)=age(4)+2=18

age(4)=age(3)+2
age(5)=age(4)+2

栈 [pop(n=4)]

age(4)=age(3)+2=16

age(3)=age(2)+2
age(4)=age(3)+2
age(5)=age(4)+2

栈 [pop(n=3)]

age(3)=age(2)+2=14

age(2)=age(1)+2
age(3)=age(2)+2
age(4)=age(3)+2
age(5)=age(4)+2

栈 [pop(n=2)]

age(2)=age(1)+2=12

例: $n!$

$$n! = \begin{cases} 1 & (n = 0, 1) \\ n(n-1) & (n > 1) \end{cases}$$

```
double fac(int n) // n!较大, 因此函数返回类型设置为double
{
   if(n==0 || n==1) return 1;
   else return n*fac(n-1);
}
int main()
{
   printf("%.0lf\n",fac(2)); // 2
   printf("%.0lf\n",fac(3)); // 6
   printf("%.0lf\n",fac(4)); // 24
   printf("%.0lf\n",fac(20));// 2432902008176640000
   return 0;
}
```

例: $n!$

$$n! = \begin{cases} 1 & (n = 0, 1) \\ n(n-1) & (n > 1) \end{cases}$$

```
double fac(int n) // n!较大，因此函数返回类型设置为double
{
   if(n==0 || n==1) return 1;
   else return n*fac(n-1);
}
int main()
{
   printf("%.0lf\n",fac(2)); // 2
   printf("%.0lf\n",fac(3)); // 6
   printf("%.0lf\n",fac(4)); // 24
   printf("%.0lf\n",fac(20));// 2432902008176640000
   return 0;
}
```

例: 斐波那契数列

$$\begin{cases} F_n = 1 & (n = 0, 1) \\ F_n = F_{n-1} + F_{n-2} & (n > 1) \end{cases}$$

```
double F(int n) // Fn较大，因此函数返回类型设置为double
{
   if(n==0 || n==1) return 1;
   else return F(n-1)+F(n-2);
}
int main()
{
   int i;
   for(i=0;i<20;i++)
   {
      printf("%d\t",(int)F(i));
      if((i+1)%4==0) printf("\n");
   }
   return 0;
}
```

例: 斐波那契数列

$$\begin{cases} F_n = 1 & (n = 0, 1) \\ F_n = F_{n-1} + F_{n-2} & (n > 1) \end{cases}$$

```
double F(int n) // Fn较大，因此函数返回类型设置为double
{
   if(n==0 || n==1) return 1;
   else return F(n-1)+F(n-2);
}
int main()
{
   int i;
   for(i=0;i<20;i++)
   {
      printf("%d\t",(int)F(i));
      if((i+1)%4==0) printf("\n");
   }
   return 0;
}
```

求整数 a, b 的最大公约数, 欧几里得算法

古希腊数学家欧几里德在其著作 «The Elements» 中最早描述了这种算法。

定理: 两个整数的最大公约数等于其中较小的那个数和两数相除余数的最大公约数。

伪代码分析

```
a,b的最大公约数, [则: a=mb+r, m=a/b; r=a%b]
循环, b作被除数, 分母是余数r,
[则: n=b/r; r=nb; a=mb+nb=(m+n)b; 如果一个整数能整除b, 必整除a]
直到r=0, 本轮循环的a(上轮循环的b)就是最大公约数。
while(1)
{
  r = a%b; // 注意b为0时, 不能计算余数, a就是最大公约数
  if(r==0) { gcd=a; break; } // 本轮循环的a(上轮循环的b)就是最大公约数
  a=b; b=r; // 准备下一轮迭代
}
```

求整数 a, b 的最大公约数, 非递归实现

```
int a,b,r,t;
scanf("%d%d",&a,&b); // 机试系统不要想当然给提示语句, 除非题目要求
if(a<b) { t=a; a=b; b=t; } // 交换a,b,使a是较大者
while(1)
{
    if(b==0) { t=a; break; } // 分母为0时, a就是最大公约数
    r = a%b;
    if(r==0) {t=b; break;} // 本轮循环的a(上轮循环的b)就是最大公约数
    a=b; b=r; // 准备下一轮迭代
}
printf("%d\n",t);// 输出最大公约数
```

求整数 a, b 的最大公约数, 递归实现

```
// 求a,b的最大公约数(约定a是分子,b是分母)
int gcd(int a, int b)
{
   int result;
   if(b==0) result=a;
   else result=gcd(b,a%b); // b是分子, a%b成为分母
   return result;
}

int main()
{
  int a,b,t;
  scanf("%d%d",&a,&b);
  if(a<b) // 确保 a > b
  {
    t=a; a=b; b=t;
  }
  printf("%d\n",gcd(a,b));
  return 0;
}
```

例: 数字处理

编写一个程序,从键盘输入一个非零整数 n(0 < n <= 1000000000),对整数 n 进行如下处理:

将整数的各位数字取出来相加,如果结果是一位数则输出该数,否则重复上述过程,直到得到的结果为一位数,并输出该结果。

例如:n=456,变换过程如下

4+5+6=15

1+5=6

输出结果为 6

例: 数字处理—非递归实现

```c
#include<stdio.h>
int bitsSum(int a);
int main()
{
   int n,sum;
   scanf("%d",&n);
   while(1)
   {
      sum=bitsSum(n);
      if(sum<=9) break;//1位数字
      else n=sum; // 继续下一轮迭代
    }
    printf("%d\n",sum);
    return 0;
}
```

```c
// 整数a的各位数字之和
int bitsSum(int a)
{
   int sum=0;
   while(a)
   {
      sum += a%10;
      a /= 10;
   }
   return sum;
}
```

例: 数字处理—递归实现

```c
#include<stdio.h>
int bitsSum(int a);
int bits1(int n);

int main()
{
   int n,sum=0;
   scanf("%d",&n);
   printf("%d\n",bits1(n));
   return 0;
}
```

```c
// 整数a的各位数字之和
int bitsSum(int a)
{
   int sum;
   if(a==0) sum=0;
   else sum=bitsSum(a/10)+a%10;
   return sum;
}
// 确保最后是1位数字
int bits1(int n)
{
   int result;
   result=bitsSum(n);
   if(result>9) result=bits1(result);//递归
   return result;
}
```

例: 二进制输出 (正序)

```
void to_binary(unsigned long n);
int main()
{
  // 无符号长整型
  unsigned long x=0XD2; //11010010
  scanf("%x",&x); //十六进制输入
  //scanf("%uld",&x); //等效10进制无符号长整型
  to_binary(x);
  putchar('\n');
  return 0;
}

void to_binary(unsigned long n)
{
  int r;
  r=n%2;//01001011
  // push(先计算的二进制位r)
  if(n>=2) to_binary(n/2);
  // pop(后进先出FILO=先进后出FILO)
  putchar('0'+r);//11010010
}
```

以x=0XD2为例,分析递归过程。

`r=n%2;`

计算 $2^0, 2^1, \ldots, 2^7$

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
0	1	0	0	1	0	1	1

`to_binary(n/2);`

push(先计算的 r), 栈顶 (top, 最左端), 栈底 (bottom, 最右端)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	0	1	0

`putchar('0'+r);`

pop(后进先出), 即栈顶元素先出, 栈底元素最后出

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	0	1	0

例: 二进制输出 (倒序)

```
void to_binary(unsigned long n);
int main()
{
  // 无符号长整型
  unsigned long x=0XD2; //11010010
  scanf("%x",&x); //十六进制输入
  //scanf("%uld",&x); //等效10进制无符号长整型
  to_binary(x);
  putchar('\n');
  return 0;
}

void to_binary(unsigned long n)
{
  int r;
  r=n%2;//01001011
  putchar('0'+r);//01001011
  // push(先计算的二进制位r)
  if(n>=2) to_binary(n/2);
}
```

以x=XD2为例,分析递归过程。

`r=n%2;`

计算 $2^0, 2^1, \ldots, 2^7$

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
0	1	0	0	1	0	1	1

`to_binary(n/2);`

push(先计算的 r), 栈顶 (top, 最左端), 栈底 (bottom, 最右端)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	0	1	0

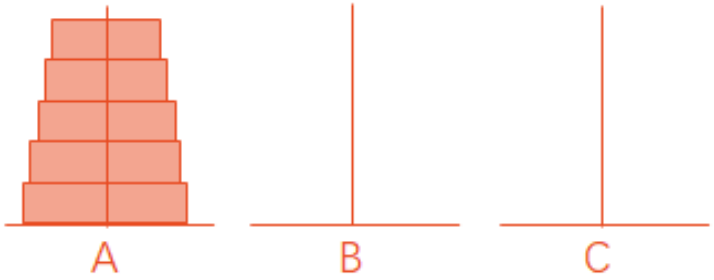
pop(后进先出), 即栈顶元素先出, 栈底元素最后出

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	0	1	0

例: Hanoi(汉诺) 塔问题

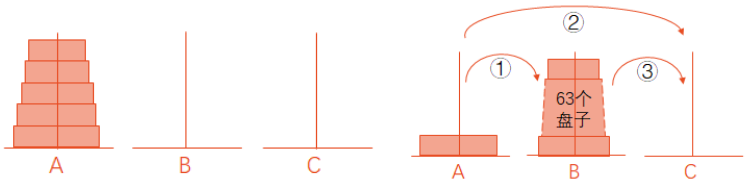
古代有一个梵塔,塔内有 3 个座 A,B,C。开始时 A 座上有 64 个盘子,盘子大小不等,大的在下,小的在上。有一个老和尚想把这 64 个盘子从 A 座移到 C 座,但规定每次只允许移动一个盘,且在移动过程中在 3 个座上都始终保持大盘在下,小盘在上。在移动过程中可以利用 B 座。

要求编程序输出移动盘子的步骤。



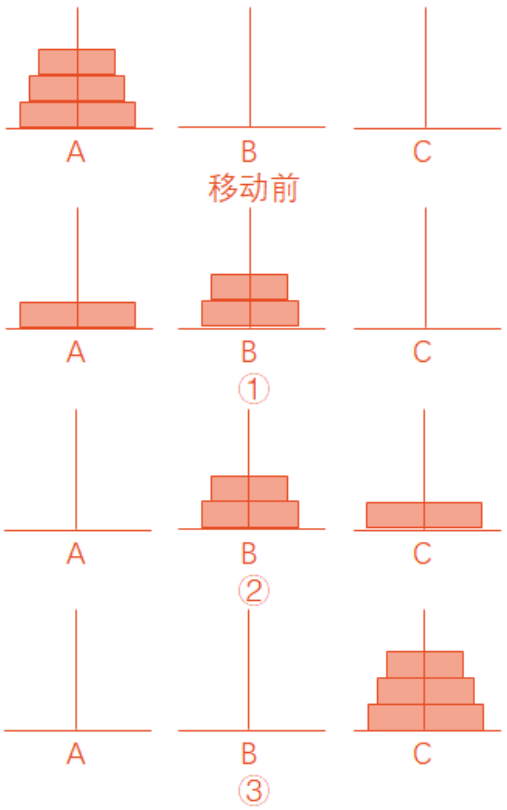
例: Hanoi(汉诺) 塔问题—解题思路

- 老和尚想: 假如有第 2 个和尚将上面 63 个盘子从 A 座移到 B 座。我就能将 64 个盘子从 A 座移到 C 座。。
  1. 命令第 2 个和尚将 63 个盘子从 A 座移到 B 座;
  2. 自己将 1 个盘子 (最底下的、最大的盘子) 从 A 座移到 C 座;
  3. 再命令第 2 个和尚将 63 个盘子从 B 座移到 C 座。
- 第 2 个和尚想: 假如有第 3 个和尚能将上面 62 个盘子从 A 座移到 C 座,我就能将 63 个盘子从 A 座移到 B 座。
  1. 命令第 3 个和尚将 62 个盘子从 A 座移到 C 座;
  2. 自己将 1 个盘子从 A 座移到 B 座;
  3. 再命令第 3 个和尚将 62 个盘子从 C 座移到 B 座。
- ...



例: Hanoi(汉诺) 塔问题—解题思路 (3 层)

- 将 A 座的 3 个盘子移到 C 座, 分 3 步:
  1. 将 A 上 2 个盘子移到 B**(借助 C 座)**。
  2. 将 A 上 1 个盘子移到 C**(直接实现)**。
  3. 将 B 上 2 个盘子移到 C**(借助 A 座)**。
- 分解第 1 步: 将 A 上的 2 个盘子移到 B:
  1. 将 A 上 1 个盘子移到 C**(借助 C 座)**。
  2. 将 A 上 1 个盘子移到 B。
  3. 将 C 上 1 个盘子移到 B。
- 分解第 3 步: 将 B 座的 2 个盘子移到 C 座:
  1. 将 B 上 1 个盘子移到 A**(借助 A 座)**。
  2. 将 B 上 1 个盘子移到 C。
  3. 将 A 上 1 个盘子移到 C。



例: Hanoi(汉诺) 塔问题—解

```c
void hanoi(int n,char one,
    char two,char three);
void move(char one,char two)
    ;
int main()
{
   int n=3;
   scanf("%d",&n);
   //n个盘由'A'移到'C',借助'B'
   hanoi(n,'A','B','C');
   return 0;
}
void move(char one,char two)
{
 printf("%c->%c\n",one,two);
}
```

```c
//将n个盘从one移到three,借助two
void hanoi(int n,char one,char two,char
    three)
{
   if(n==1) move(one,three);
   else
   {
      //n-1盘从one移到two,借three
      hanoi(n-1,one,three,two);
      //将1个盘从one移到three
      move(one,three);
      //n-1盘从two移到three,借one
      hanoi(n-1,two,one,three);
   }
}
```

- 将 A 座的 3 个盘子移到 C 座，分 3 步:
  1. 将 A 上 2 个盘子移到 B**(借 C)**。
  2. 将 A 上 1 个盘子移到 C**(直接)**。
  3. 将 B 上 2 个盘子移到 C**(借 A)**。
- 分解第 1 步: 将 A 上的 2 个盘子移到 B:
  1. 将 A 上 1 个盘子移到 C**(借 C)**。
  2. 将 A 上 1 个盘子移到 B。
  3. 将 C 上 1 个盘子移到 B。
- 分解第 3 步: 将 B 座的 2 个盘子移到 C 座:
  1. 将 B 上 1 个盘子移到 A**(借 A)**。
  2. 将 B 上 1 个盘子移到 C。
  3. 将 A 上 1 个盘子移到 C。

```
//将n个盘从one移到three,借助two
void hanoi(int n,char one,char two,char
    three)
{
  if(n==1) move(one,three); // 递归终止条件,
    出栈pop(2,...,n-1,n), 回归, 回溯
  else
  { // 压栈push(n,n-1,n-2,...2)
    //n-1盘从one移到two,借three
    hanoi(n-1,one,three,two);
    //将1个盘从one移到three
    move(one,three);
    //n-1盘从two移到three,借one
    hanoi(n-1,two,one,three);
  }
}
```

$A \to C, A \to B, C \to B, A \to C, B \to A, B \to C, A \to C$

递归小结

- 递归函数必须满足两个条件:
  1. 在每一次调用自己时,必须是 (在某种意义上) 更接近于解;
  2. 必须有一个终止处理的语句。
- 递归能够解决的问题:
  1. 数据的定义是按递归定义的。如 Fibonacci 函数。
  2. 问题解法按递归算法实现。如 Hanoi 问题。
  3. 数据的结构形式是按递归定义的。如二叉树、广义表等。
- 使用递归的关键在于将问题分解为小部分,递归不能永远进行下去,因为它总是以最小可能性问题结束,
- 递归函数的优点是定义简单,逻辑清晰,但是代码难于理解。
- 理论上,所有的递归函数都可以写成循环的方式,但循环逻辑不如递归清晰。
- 层次越深, 调用栈 (内存) 越大, 效率越低, 并且可能会溢出。因此, 能用循环解决的问题, 尽量不要用递归。

局部变量及其作用域

定义在函数内部或复合语句中的变量称为**局部变量**。

局部变量的**作用域**是从定义语句开始至函数或复合语句结束,不会影响作用域以外的同类型的同名变量值。

```
// 各函数中的局部变量, 不会相互影响。
float f1(folat a)
{
   float c; // 本函数局部变量
   ....;
}
float f2(float a)
{
   float c; // 本函数局部变量
   ....;
}
int main()
{
   float c=10,d; // 本函数局部变量
   d=f1(c);
   d=f2(d);
}
```

```
// 各复合语句中的局部量, 不会相互影响。
float f1(folat a)
{
   float c; // 本函数局部变量
   for(i=0;i<10;i++)
   {
     int t; // 复合语句中的局部变量
     ....
   }
   if(c>10)
   {
     int t; // 复合语句中的局部变量
     ....
   }
   ...
}
```

形式参数被当作本函数的局部变量

形式参数被当作本函数的局部变量,因此它不会影响实际参数的值。

```
float f1(folat a)
{
  float c; // 本函数内部的局部变量不得与
    形式参数同名
  a=30; // 改变形式参数的值不会影响实际
    参数的值
  ....;
}
int main()
{
   float c=10,d; // 本函数局部变量
   d=f1(c); // f1函数对形参的改变不会影
    响实参c的值
   float x[2]={0.1,0.2};
   f2(x); // 函数内部可以改变数组元素的
    值
   printf("%f,%f,%f\n",c,x[0],x[1])
    ; // 10,10.1,10.2
}
```

```
// 函数内部可以改变数组元素的值,但是对地址
    的改变不会影响实参的地址
float f2(folat a[])
{
  a[0]=10.1; // 对数组元素的改变, 就是
    对实参数组元素的改变
  a[1]=10.2;
  a=a+1; // 对数组名(地址)的改变, 不会
    影响实参的地址。
  ...
}
```

全局变量及其作用域

定义在函数外部的变量称为**全局变量**。与模块化函数封装思想冲突,不推荐使用。全局变量的作用域是从定义语句开始至该文件结束,会影响作用域内的同类型的同名变量值。

```
int g=0; // 全局变量
float f1(folat a)
{
  g=10; // 改变全局变量的值
  ....;
}
float f2(float a)
{
  g=20; // 改变全局变量的值
  ....;
}
int main()
{
   g=30; // 改变全局变量的值
   ....;
}
```

lecture-15 主要内容

善于利用指针

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- *while*(表达式){ · · · }
- *do*{ · · · }*while*(表达式);
- *for*(表达式 1; 表达式 2; 表达式 3){ · · · }
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6},{7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
  - 连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)
  - 字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)
  - 获取字符串的长度 strlen(s)
  - 大小写转换 strlwr(s); strupr(s)
- 程序举例
  - 统计单词数
  - 求多个字符串中的较大者
  - 矩阵乘法
- 使用函数进行模块化程序设计
  - 函数三要素: 函数原型声明, 函数定义, 函数调用
  - 函数定义: int add(int a,int b){ }
- 实参和形参间的数据传递 (值传递和地址传递)
- 程序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 整数 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例续

指针就是指向变量存储单元 (首) 地址的变量

- 在程序中定义的各个变量, 在对程序进行编译时, 系统就会给每个变量分配一个存储单元。
- 基本存储单元是一个字节 (byte)。
- 根据变量类型, 系统分配相应长度的存储单元, 其长度是字节的整数倍。
- 不同的编译器, 给变量分配的存储空间可能会有所不同。sizeof(类型), 可获得该编译器给相应类型的变量分配的存储空间大小。
- 指针就是指向目标变量存储单元 (首) 地址的变量。

```
char c;//分配1个存储单元, 即1个字节
int  a;//分配2个存储单元, 即2个字节(大多数是4个字节)
float f;//分配4个存储单元, 即4个字节
double d; //分配8个存储单元, 即8个字节
printf("%d,%d,%d,%d\n",sizeof(char),sizeof(int),
    sizeof(float),sizeof(double));
// 1,4,4,8
```

变量名	地址	内容
c	2000	'A'
	...	
a	3000	20
	3001	
	...	
f	4000	10.2
	4001	
	4002	
	4003	
	...	
d	5000	20.8
	5001	
	5002	
	5003	
	...	

定义指针变量: int *p;

指针就是指向变量存储单元 (首) 地址的变量。

```
数据类型 *指针变量名; //数据类型表示指针指向目标变量类型,*表示其后的变量是指针变量

int *pa; // 定义指向整数类型变量的指针变量pa, *表示其后的变量是指针变量
char *pc; // 定义指向字符类型变量的指针变量pc
float *pf; // 定义指向单精度浮点类型变量的指针变量pf
double *pd; // 定义指向双精度浮点类型变量的指针变量pd
```

特别注意

以上定义了各个指针变量,但是还没有设置指针指向的目标变量。未设置目标变量前的指针变量,是一个“无用”的变量,此时定义的指针变量仅说明了将来的目标变量的数据类型。

设置指针变量指向的目标变量

地址运算符 &, 可获得变量在内存中的 (首) 地址。

```
// 定义指针变量时，初始化它指向的目标变量

// 定义指针变量pa，并初始化pa指向变量a
int a=20,*pa=&a;
// 定义指针变量pc，并初始化pc指向变量c
char c='A',*pc=&c;
// 定义指针变量pf，并初始化pf指向变量f
float f=10.2,*pf=&f;
// 定义指针变量pd，并初始化pd指向变量d
double d=20.8,*pd=&d;

int b=30,*p;

// 根据需要，可以随时设置指针指向的目标变量

// p前无'*'，此时p已经是指针变量了，仅是设置它的指向
p=&b;
// 改变pa的指向
pa=p; // 或
pa=&b;
```

变量名	指针	地址	内容
c	pc →	2000	'A'
		...	
a	pa →	3000	20
		3001	
		...	
f	pf →	4000	10.2
		4001	
		4002	
		4003	
		...	
d	pd →	5000	20.8
		5001	
		5002	
		5003	
		...	

变量的直接访问

- **直接访问:** 通过变量名访问 (读写) 存储单元的内容 (变量的值)
- 此前的程序, 均采用直接访问的形式, 读写变量的值。

```
char c; // 分配1个存储单元, 即1个字节
int  a; // 分配2个存储单元, 即2个字节(大多数是4个字节)
float f; // 分配4个存储单元, 即4个字节
double d; // 分配8个存储单元, 即8个字节
// 直接访问
// "写" 变量的值
c='A';
a=20;
f=10.2;
d=20.8;
// "读"变量的值
printf("%c,%d,%f,%lf\n",c,a,f,d);
```

变量名	地址	内容
c	2000	'A'
	...	
a	3000	20
	3001	
	...	
f	4000	10.2
	4001	
	4002	
	4003	
	...	
d	5000	20.8
	5001	
	5002	
	5003	
	...	

变量的间接访问

- **间接访问:** 通过变量的 (首) 地址访问 (读写) 存储单元的内容 (变量的值)
- * 指针变量, **间接访问运算符 ***, 表示读写指针的目标变量值

```
// 定义指针变量，并初始化， 指向特定的变量地址
char c, *pc=&c; //pc是指向char类型变量的指针变量
int  a, *pa=&a; //pa是指向char类型变量的指针变量
float f, *pf=&f; //pf是指向char类型变量的指针变量
double d, *pd=&d; //pd是指向char类型变量的指针变量
// 间接访问
// "写" 变量的值
*pc='A'; // 不要忘记前面的*, 否则成为改变指针指向的语句。
*pa=20;
*pf=10.2;
*pd=20.8;
// "读"变量的值
printf("%c,%d,%f,%lf\n",*pc,*pa,*pf,*pd);
```

变量名	指针	地址	内容
c	pc →	2000	'A'
		...	
a	pa →	3000	20
		3001	
		...	
f	pf →	4000	10.2
		4001	
		4002	
		4003	
		...	
d	pd →	5000	20.8
		5001	
		5002	
		5003	
		...	

指针变量使用 (间接访问) 前,必须有指向。

```
// 注意: 指针变量使用前, 必须有指向
float *p;
*p=10.2; // 错!!! p没有设置指向的目标变量, 即p未赋值, 间接访问是错误的。

float f;
p=&f;
*p=10.2; // 合法

int a, *pa;
scanf("%d",pa); // 错!!! pa没有设置指向的目标变量
pa=&a; // 设置pa指向a
scanf("%d",pa); // pa有指向了, 正确
scanf("%d",&a); // 等效
printf("%d,%d\n",a,*pa); // 输出相同的值
```

例: 交换两个指针变量的值 (地址, 指向)

```
int a=10, *pa=&a, b=20, *pb=&b;
int *tmp;
if(a<b) // 不交换a,b的值, 交换pa,pb的指向
{
  tmp=pa;
  pa=pb;
  pb=tmp;
}
printf("%d,%d,%d,%d\n",a,b,*pa,*pb); // 10,20,20,10

// 或者, 不使用中间变量
if(a<b) // 不交换a,b的值, 交换pa,pb的指向
{
  pa=&b;
  pb=&a;
}
printf("%d,%d,%d,%d\n",a,b,*pa,*pb); // 10,20,20,10
```

指针变量作为函数参数 (地址传递, 形参和实参指向同一存储单元)

```
void swap(int *p1, int *p2);
int main()
{
    int a=10, *pa=&a, b=20, *pb=&b;
    if(a<b) swap(pa,pb); //地址传递
    //if(a<b) swap(&a,&b); //等效, 地址传递
    printf("%d,%d,%d,%d\n",a,b,*pa,*pb);
    // 20,10,20,10
    return 0;
}
// 交换p1,p2指向的内容, 注意不是交换p1,p2本身
void swap(int *p1, int *p2)
{
   int tmp;
   tmp=*p1;
   *p1=*p2;
   *p2=tmp;
}
```

main 函数: 调用 swap 前

实参指针	地址	内容
pa=&a →	3000	10
	3001	
pb=&b →	4000	20
	4001	

swap 函数: p1,p2 指向内容交换前

形参指针	地址	内容
p1 →	3000	10
	3001	
p2 →	4000	20
	4001	

形参实参地址相同。

指针变量作为函数参数 (地址传递, 形参和实参指向同一存储单元)

```
void swap(int *p1, int *p2);
int main()
{
  int a=10, *pa=&a, b=20, *pb=&b;
  if(a<b) swap(pa,pb); //地址传递
  //if(a<b) swap(&a,&b); //等效, 地址传递
  printf("%d,%d,%d,%d\n",a,b,*pa,*pb);
  // 20,10,20,10
  return 0;
}
// 交换p1,p2指向的内容, 注意不是交换p1,p2本身
void swap(int *p1, int *p2)
{
  int tmp;
  tmp=*p1;
  *p1=*p2;
  *p2=tmp;
}
```

main 函数: 调用 swap 后

实参指针	地址	内容
pa=&a →	3000	10 → 20
	3001	
pb=&b →	4000	20 → 10
	4001	

swap 函数: p1,p2 指向内容交换后

形参指针	地址	内容
p1 →	3000	10 → 20
	3001	
p2 →	4000	20 → 10
	4001	

形参实参地址相同。函数内对指针目标变量的改变, 就是改变实参的内容。

普通变量作为函数参数 (值传递, 形参和实参处于不同存储单元)

```
void swap(int p1, int p2);
int main()
{
  int a=10, *pa=&a, b=20, *pb=&b;
  if(a<b) swap(a,b);//值传递, 实参的值不会改变
  printf("%d,%d,%d,%d\n",a,b,*pa,*pb);
  // 10,20,10,20
  return 0;
}
// 对形参的改变不会引起实参的改变
void swap(int p1, int p2)
{
  int tmp;
  tmp=p1;
  p1=p2;
  p2=tmp;
}
```

main 函数: 调用 swap 前后

实参	地址	内容
a	3000	10
	3001	
b	4000	20
	4001	

swap 函数: 形参是局部变量, 不会影响实参的值

形参	地址	内容
p1	5000	10 → 20
	5001	
p2	6000	20 → 10
	6001	

形参实参地址 "不同"。

地址传递与值传递小结

- **地址传递:** 指针变量作为函数参数, 形参和实参指向同一存储单元。函数内对形参指向内容的改变**会**引起实参指向内容的改变。
- **值传递:** 普通变量作为函数参数, 形参和实参处于不同存储单元。对形参的改变**不会**引起实参的改变。

```c
// 地址传递
void swap(int *p1, int *p2)
{
  int tmp;
  tmp=*p1;
  *p1=*p2;
  *p2=tmp;
}
```

```c
// 值传递
void swap(int p1, int p2)
{
  int tmp;
  tmp=p1;
  p1=p2;
  p2=tmp;
}
```

常见错误

```c
//对p1,p2值交换而不是交换p1,p2指向的内容
//p1,p2是局部变量,不会引起实参值的改变
void swap(int *p1, int *p2)
{
  int *tmp;
  tmp=p1;
  p1=p2;
  p2=tmp;
}
```

```c
// tmp使用前，要有指向
void swap(int *p1, int *p2)
{
  int *tmp;//没有指向，即未赋值
  *tmp=*p1;//错！ tmp没有指向就间接访问
  *p1=*p2;
  *p2=*tmp;
}
```

借用地址传递, 使函数 “返回” 多个值

```c
int fun(int *p1, int *p2, int *p3);
int main()
{
  float a=10,b=20,c=30;
  int r=fun(&a,&b,&c);
  printf("%d,%f,%f,%f\n",r,a,b,c);
  // 4,1,2,3
  return 0;
}
```

```c
//对指针指向内容的改变, "返回"给调用者
int fun(int *p1, int *p2, int *p3)
{
  *p1=1;
  *p2=2;
  *p3=3;
  return 4;
}
```

不能混用不同类型的指针变量

```
int a=10, *p1=&a;
float b=20,*p2=&b;
p1=&b; // 错误,虽然编译器不报错
p2=&a; // 错误,虽然编译器不报错
p2=p1; // 错误,虽然编译器不报错
```

不同数据类型存错空间长度不同,因此,不能混用。

不同类型的指针变量	地址	内容
p1	5000	10
	5001	
p2	6000	20
	6001	
	6002	
	6003	

数组与指针

- `int a[10],*p=a;//p指向数组元素的首地址`
- `p=a; //等效p=&a[0];`
- `p+i; //p+i指向数组的第i个元素,*(p+i)=a[i]`
- `p++; //p指向数组的下一个元素,注意越界`
- `p--; //p指向数组的上一个元素,注意越界`
- `a=a+1; //错误, a是地址常量`
- `p=p+1; //正确, p是地址变量`

p=a	a+0	→	a[0]
p+1	a+1	→	a[1]
p+2	a+2	→	a[2]
...	...	...	...
p+i	a+i	→	a[i]
...	...	...	...
p+n-1	a+n-1	→	a[n-1]

用数组名作函数参数 (地址传递, 数组名是数组首地址)

```c
//编译器认为a是指针变量
void fun(int a[],int n)
// void fun(int *a,int n) //自动转化为此形式
{
  a[0]=10;
  a[1]=20;
  a[3]=30;
  // 因为a被当作指针变量, 因此下面的语句是合法的
  a++; // 使a指向a[1]
  *a=20; // a[1]=20
}
```

```c
// 指针与数组通用
void fun(int *a,int n)
{
    *a=10; // a[0]=10
    a++;
    *a=20; // a[1]=20
    a++;
    *a=30; // a[3]=30
}
```

注

用数组名作函数参数时, 编译器会把数组名自动转化为指针变量处理。函数内既可以按指针变量间接访问数组元素, 也可以按数组常规访问数组元素。

例: 将数组 a 中 n 个整数按相反顺序存放.

```c
#define N 100
void inv(int *x,int n);
int main()
{
   int i,n=6,a[N]={1,2,3,4,5,0};
   inv(a,n);
   for(i=0;i<n;i++)
     printf("%d␣",a[i]);
     // 0,5,4,3,2,1
   printf("\n");
   return 0;
}
```

```c
// n是数组长度
void inv(int *x,int n)
{
   int *p1,*p2,tmp;
   p1=x; // 指向首元素
   p2=x+n-1; // 指向最末元素
   //while(p1!=p2)//不适于n为偶数
   while(p1<p2)
   { //互换两个指针的目标变量值
      tmp=*p1;
      *p1=*p2;
      *p2=tmp;
      p1++; // 右移一元素
      p2--; // 左移一元素
   }
}
```

字符串常量

```
char s1[]="abcd"; // 数组长度为4(不包含末尾的'\0'), s1[4]='\0'
char s2[10]="abcd"; // 数组长度为10, s2[4]='\0', s2[5]到s2[9]未赋值

// 指针就是字符串常量的首地址
char *s3="abcd";  // 数组长度为4(不包含末尾的'\0'), s3[4]='\0'
// s3与s1等效, 但是与s2不等效。

s1="1234"; // 错误, 数组名是地址常量
strcpy(s1,"1234"); // 合法, 但是s1要大于等于"1234"的长度+1
s3="1234"; // 正确, 指针是变量
```

注意:

数组有长度概念,指针没有长度概念。系统维护一个存储空间,存储字符串常量,因此每个字符串常量具有唯一地址。

例: 实现自己的字符串拷贝函数

```c
void copy(char to[], char from[]);
int main()
{
   char a[81],b[81];
   gets(a); // 输入字符串
   copy_string(b,a);
   puts(b);
}
// 数组版本
void copy(char to[], char from[])
{
   int i=0;
   while(from[i]!='\0')
   {
      to[i]=from[i];
      i++;
   }
   to[i]='\0'; // 使字符串以'\0'结束
}
```

```c
// 指针版本
void copy(char *to, char *from)
{
   // 等效while(*from!='\0')
   // 或while(*from!=0)
   while(*from)
   {
      *to=*from;
      from++;
      to++;
   }
   *to='\0';// 不要忘记此语句
}
```

字符串数组

```
#define N 100
char name[N][81]; // N个字符串
int n=10,i; // n是实际字符串个数

for(i=0;i<n;i++) // 输入n个字符串
{
   gets(name[i]);
}

for(i=0;i<n;i++) // 输出
{
   puts(name[i]);
}
```

例: 选择法排序字符串数组

```c
#define N 100
void sort(char name[][81],int n);
int main()
{
  char name[N][81]; // N个字符串
  int n=10,i; // n是实际大小

  for(i=0;i<n;i++) // 输入n个字符串
  {
    gets(name[i]);
  }
  sort(name,n); // 排序
  for(i=0;i<n;i++) // 输出
  {
    puts(name[i]);
  }
}
```

```c
// 用选择法排序(从小到大)
void sort(char name[][81],int n)
{
  char tmp[81];
  int i,j,k;
  for(i=0;i<n-1;i++)
  {
    k=i;// 未排序部分最小的字符串
    for(j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0)
        k=j;
    if(k!=i) // 交换字符串
    {
      strcpy(tmp,name[i]);
      strcpy(name[i],name[k]);
      strcpy(name[k],tmp);
    }
  }
}
```

lecture-15 主要内容

善于利用指针

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- *while*(表达式){ · · · }
- *do*{ · · · }*while*(表达式);
- *for*(表达式 1; 表达式 2; 表达式 3){ · · · }
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6},{7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
  - 连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)
  - 字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)
  - 获取字符串的长度 strlen(s)
  - 大小写转换 strlwr(s); strupr(s)
- 程序举例
  - 统计单词数
  - 求多个字符串中的较大者
  - 矩阵乘法
- 使用函数进行模块化程序设计
  - 函数三要素: 函数原型声明, 函数定义, 函数调用
  - 函数定义: int add(int a,int b){ }
- 实参和形参间的数据传递 (值传递和地址传递)
- 程序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 整数 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例续

lecture-16 主要内容

指针综合应用举例, 期中考试总结

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- *while*(表达式){ · · · }
- *do*{ · · · }*while*(表达式);
- *for*(表达式 1; 表达式 2; 表达式 3){ · · · }
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5};
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={{1,2,3},{4,5,6},{7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
  - 连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)
  - 字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)
  - 获取字符串的长度 strlen(s)
  - 大小写转换 strlwr(s); strupr(s)
- 程序举例
  - 统计单词数
  - 求多个字符串中的较大者
  - 矩阵乘法
- 使用函数进行模块化程序设计
  - 函数三要素: 函数原型声明, 函数定义, 函数调用
  - 函数定义: int add(int a,int b){ }
- 实参和形参间的数据传递 (值传递和地址传递)
- 程序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 整数 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例续

例: 表达式求值

表达式由两个非负整数 x, y 和一个运算符 op 构成, 求表达式的值。

这两个整数和运算符的顺序是随机的, 可能是 “x op y”, “op x y” 或者 “x y op”, 例如, “25 + 3” 表示 25 加 3, “5 30 *” 表示 5 乘以 30, “/ 600 15” 表示 600 除以 15。

输入说明

输入为一个表达式, 表达式由两个非负整数 x, y 和一个运算符 op 构成, x,y 和 op 之间以空格分隔, 但顺序不确定。x 和 y 均不大于 10000000, op 可以是 +, -, *, /, % 中的任意一种, 分表表示加法, 减法, 乘法, 除法和求余。除法按整数除法求值, 输入数据保证除法和求余运算的 y 值不为 0。

输出说明

输出表达式的值。

输入样例	输出样例
样例 1 输入	样例 1 输出
5 20 *	100
样例 2 输入	样例 2 输出
4 + 8	12
样例 3 输入	样例 3 输出
/ 8 4	2

例: 表达式求值—解题思路

- gets函数读取字符串,遍历字符串,根据 op 字符是" 非数字字符" 的特点,判断表达式的三种形式。
- 设计各个子函数, 进行模块化程序设计。
- 编写函数 parse, 解析输入字符串, 生成 3 个子串, 分别代表 op,x,y
- 编写函数 strToInt, 将数字字符串转为整数。
- 编写函数 compute, 根据 op,x,y 计算表达式的值。
- 编写其它辅助函数。
- 主函数, 调用上述函数, 完成程序功能。

注:

```
int s1[81],s2[81],s3[81]; scanf("%s%s%s",s1,s2,s3);// 遇空格结束特点(忽略前缀后缀空格), 自动获取3个子串, 无需分解。
```

例: 表达式求值—计算函数

```
// 根据参数，计算表达式的值
int compute(char op,int x,int y)
{
    int result = -1;
    switch(op)
    {
      case '+': result = x+y; break;
      case '-': result = x-y; break;
      case '*': result = x*y; break;
      case '/': if(y != 0) result = x/y; break;
      case '%': if(y != 0) result = x%y; break;
    }
    return result;
}
```

例: 表达式求值—将数字字符串转为整数函数

```
// 数字字符串s转为int, 要求s以'\0'结尾
int strToInt(char *s) //等效: int toInt(char s[])
{
  int result=0;
  while(*s) //等效 *s != '\0'
  {
     result = result*10 + (*s-'0');
     s++; //移至下一字符
  }
  return result;
}
```

注: 简单修改此函数, 可将由 01 组成的二进制字符串转为整数。

例: 表达式求值—提取子串函数

```
/*****************************************************
 提取子串函数, 忽略s中空格前缀, 复制s中的字符串到subs中, 遇空格或'\0'结束
 返回subs不含空格。 返回复制后s指针当前指向(地址), 要求s和subs以'\0'结尾。
*****************************************************/
char* getSubs(char *s, char *subs)
{
  int start=0; // 是否开始提取标志
  while(*s)
  {
    if(*s==' ')
    {
      if(start==0) s++; // 未开始复制, 忽略s的前缀空格
      else break; // 是有效字符串后的一个空格
    }
    else
    {
      start=1; // 标记开始复制
      *subs=*s;  s++;  subs++;
    }
  }
  *subs='\0'; // 不要忘记结尾符
  return s; // 返回复制后s指针当前指向(地址)
}
```

注: 简单修改此函数, 可以任何分隔符提取子串函数

例: 表达式求值—解析字符串函数

```
// 解析s, 以空格为分隔符, 分解s为3个字符串, 通过参数result返回
void parse(char *s,char result[][N])
{
  char *p;
  p=getSubs(s,result[0]);
  p=getSubs(p,result[1]);
  getSubs(p,result[2]);
}
```

注: 通过此函数, 学习字符串数组的使用技巧。

例: 表达式求值—是否运算符函数

```
// 如果s是操作符，返回1，参数op返回该操作符
// 否则，返回0
int isOp(char *s, char *op)
{
  if(*s >= '0' && *s <= '9')
    return 0;
  else
  {
    *op=*s;
    return 1;
  }
}
```

注: 通过此函数, 学习地址传递技巧。

例: 表达式求值—测试主函数

```
int main1() // 测试主函数, 使能改为main(), 原main()改名
{
   char *s="123";
   printf("%d\n",strToInt(s)); // 123
   char s1[N]="␣␣␣␣abcd␣␣␣456␣␣+", s2[50], *s3;
   s3= getSubs(s1,s2);
   printf("%s,%d\n",s2,*s3); // abcd,32(空格的ASCII码)
   s3= getSubs(s3,s2);
   printf("%s,%d\n",s2,*s3); // 456,32(空格的ASCII码)
   s3= getSubs(s3,s2);
   printf("%s,%d\n",s2,*s3); // +,0('\0'的ASCII码)
   char result[3][N];
   parse(s1,result);
   puts(result[0]); // abcd
   puts(result[1]); // 456
   puts(result[2]); // +
   return 0;
}
```

注: main() 和 main1() 相互换名, 调试程序。是实用调试程序技巧。

例: 表达式求值—主程序

```
#include <stdio.h>
// 自定义函数在调用之前定义，略去函数声明。
#define N 81 // 估计字符串最大长度，存储有效字符(N-1)个，预留最后一个字符'\0'
int main()
{
  char s[N], op, char s3[3][N];
  int x,y;
  gets(s); // 不能使用scanf("%s",s); 空格将会终止
  parse(s,s3); // s被分解为3个字符串
  if(isOp(s3[0],&op)) // op x y
  {
    x=strToInt(s3[1]);  y=strToInt(s3[2]);
  }
  else if(isOp(s3[1],&op)) // x op y
  {
     x=strToInt(s3[0]); y=strToInt(s3[2]);
  }
  else if(isOp(s3[2],&op)) // x y op
  {
    x=strToInt(s3[0]); y=strToInt(s3[1]);
  }
  printf("%d\n",compute(op,x,y));
  return 0;
}
```

例: 表达式求值—主程序 (另解)

```
#include <stdio.h>
// 自定义函数在调用之前定义，略去函数声明。
#define N 81 // 估计字符串最大长度，存储有效字符(N-1)个，预留最后一个字符'\0'
int main()
{
  char s[N], op, char s3[3][N];
  int x,y;
  scanf("%s%s%s",s3[0],s3[1],s3[2]); // 利用"%s"读字符串遇空格结束特点，直接
    读取3个字符串。
  if(isOp(s3[0],&op)) // op x y
  {
    x=strToInt(s3[1]);  y=strToInt(s3[2]);
  }
  else if(isOp(s3[1],&op)) // x op y
  {
    x=strToInt(s3[0]); y=strToInt(s3[2]);
  }
  else if(isOp(s3[2],&op)) // x y op
  {
    x=strToInt(s3[0]); y=strToInt(s3[1]);
  }
  printf("%d\n",compute(op,x,y));
  return 0;
}
```

期中考试总结

1. 评分规则
   - 将 5 个题的得分按从高到低排序, 记为 S1, S2, S3, S4, S5
   - 总分 =S1*0.30+S2*0.25+S3*0.20+S4*0.15+S5*0.1
2. 第 1 题: 自由落体, 基本计算, 仅有 1 人没得满分。
3. 第 2 题: 运费: 5 人未得满分, 3 人 0 分

   与 lecture-6.pdf 中的例题基本一致。考查 switch 语句或 if else 选择语句。
4. 第 3 题: 二进制字符转换, 考查一重循环, 本题平均成绩最低 44。

   lecture-8.pdf p15-16, 循环语句中接收字符的常见技巧。

   lecture-5.pdf p8, 数字 =ASCII 编码值-'0'

   lecture-14.pdf p20, 以递归形式输出一个整数的二进制位。

   课件及上机练习中 (如 ISBN) 涉及各种数字分解程序设计技巧。
5. 第 4 题, 考查一重循环, 平均成绩 86。

   lecture-9.pdf p3 原题, 迭代计算是基本编程要领, 必须掌握。
6. 质数求和, 考查二重循环, 平均成绩 51。

   lecture-8.pdf p8—p11, 100~200 间的素数。
7. 本班题目简单, 其他班有类似机试练习中选 QQ 号题。期末考试会复杂些。

第 3 题—二进制字符转换, 简单实现

10101d $\Longrightarrow$ 21 101,1 $\Longrightarrow$ 5

```
#include<stdio.h>
int main()
{
    char ch;
    int sum=0;
    while(1)
    {
        ch=getchar(); //或 scanf("%c",&ch);
        if(ch!='0' && ch !='1') break;
        sum=sum*2+ch-'0';
    }
    printf("%d\n",sum);
    return 0;
}
```

第 3 题—二进制字符转换, 字符串数组实现

```
#define N 31 // 估计最大数组大小, 预留'\0'
char ch[N];
int sum=0,i;
gets(ch);
for(i=0;;i++)
{
    if(ch[i]!='0' && ch[i] !='1') break;
    sum=sum*2+ch[i]-'0';
}
printf("%d\n",sum);
```

第 3 题—二进制字符转换, 指针实现

```
#define N 31 // 估计最大数组大小, 预留'\0'
char ch[N], *p=ch;
int sum=0;
gets(ch);
for(;;)
{
  if(*p != '0' && *p !='1') break;
  sum=sum*2+ (*p - '0');
  p++; // 指向下一个字符
}
printf("%d\n",sum);
```

整数 pow 函数问题

`double pow(double x,double y);` 数学库函数真对双精度浮点数设计。用它计算整数 x^y 会有复杂的精度问题, 因此整数运算尽量不要用此函数。

```
#include<stdio.h>
#include<math.h>   // 数学库函数
int main()
{
  int x=2,y=3;

  // 有些编译器会输出99,124, 因为转换前是99.999, 124.999
  printf("%d,%d\n",(int)pow(10,x),(int)pow(5,y));

  // 奇怪的是下列调用输出结果与上面有可能会输出不一致的结果
  printf("%d,%d\n",(int)pow(10,2),(int)pow(5,3));

  return 0;
}
```

自定义整数 pow 函数

```
#include<stdio.h>
// 自定义整数pow函数
int intpow(int x, int y)
{
  int i,p=1;
  if(x==0) return 0;
  for(i=0;i<y;i++) p=p*x;
  return p;
}

int main()
{
  int x=2,y=3;

  printf("%d,%d\n",intpow(10,x),intpow(5,y)); // 100,125
  printf("%d,%d\n",intpow(10,2),intpow(5,3)); // 100,125
  printf("%d,%d\n",intpow(0,2),intpow(0,0)); // 0,1

  return 0;
}
```

lecture-17 主要内容

建立自己的数据类型

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- *while*(表达式){···}
- *do*{···}*while*(表达式);
- *for*(表达式 1; 表达式 2; 表达式 3){···}
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5} ;
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={ {1,2,3} ,{4,5,6} ,{7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
  - 连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)
  - 字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)
  - 获取字符串的长度 strlen(s)
  - 大小写转换 strlwr(s); strupr(s)
- 程序举例
  - 统计单词数
  - 求多个字符串中的较大者
  - 矩阵乘法
- 使用函数进行模块化程序设计
  - 函数三要素: 函数原型声明, 函数定义, 函数调用
  - 函数定义: int add(int a,int b){ }
- 实参和形参间的数据传递 (值传递和地址传递)
- 程序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 禁数 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例续

训练编程逻辑思维方式:

- 领会结构化 (模块化) 编程思想。
- 用结构体管理相互关联的数据, 使其成为结构化的整体数据, 便于统一处理。
- 大问题分解为小问题, 设计函数解决小问题, 各个子函数彼此之间相互独立 (便于调试, 不易出错), 又可通过参数和返回值传递数据。
- 主程序调用各个子函数, 解决 “大” 问题。

定义结构体数据类型: struct 结构体类型名 {...};

定义结构体数据类型

```
struct 结构体类型名
{
    数据类型 成员名1;
    数据类型 成员名2;
    ... // 定义其它成员
};
```

例: 定义结构体数据类型, 描述如下信息

num	name	sex	age	score	addr
10010	Li Fang	M	18	87.5	Xian

```
struct Student // 定义结构体数据类型struct Student
{
    int num;          // 学号为整型
    char name[20];    // 姓名为字符串
    char sex;         // 性别为字符型
    int age;          // 年龄为整型
    float score;      // 成绩为实型
    char addr[30];    // 地址为字符串
};                    // 注意最后有一个分号
```

定义结构体数据类型: struct 结构体类型名 {...};

例: 定义结构体数据类型, 描述如下信息

num	name	sex	age	score	addr
10010	Li Fang	M	18	87.5	Xian

定义结构体数据类型

```
struct 结构体类型名
{
    数据类型 成员名1;
    数据类型 成员名2;
    ... // 定义其它成员
};
```

```
struct Student // 定义结构体数据类型struct Student
{
    int num;          // 学号为整型
    char name[20];    // 姓名为字符串
    char sex;         // 性别为字符型
    int age;          // 年龄为整型
    float score;      // 成绩为实型
    char addr[30];    // 地址为字符串
};                    // 注意最后有一个分号
```

嵌套定义结构体: 结构体的成员, 也可以是另外一个结构体变量。

```
struct Date // 定义结构体类型 struct Date
{
   int month; // 月
   int day;   // 日
   int year;  // 年
};
struct Student // 定义结构体类型 struct Student
{
   int num;
   char name[20];
   char sex;
   int age;
   struct Date birthday; //成员birthday是struct Date类型变量
   char addr[30];
};
```

num	name	sex	age	birthday			addr
10010	Li Fang	M	18	month	day	year	Xian

定义结构体类型变量: struct 结构体类型名 变量名;

定义结构体类型后, 可以当作内置类型 (如, int) 一样进行变量定义和使用。

```
#include<stdio.h>
....
// 定义结构体数据类型struct Student
struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};
```

```
void fun1()
{
    // 定义结构体类型变量
    struct Student stu1, stu2;
    ....
}

// 结构体类型变量可作为函数的形式参数
void fun2(struct Student stu)
{
    ....
}
```

定义结构体类型变量: struct 结构体类型名 变量名;

定义结构体类型后, 可以当作内置类型 (如, int) 一样进行变量定义和使用。

```
#include<stdio.h>
....
// 定义结构体数据类型struct Student
struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};
```

```
void fun1()
{
    // 定义结构体类型变量
    struct Student stu1, stu2;
    ....
}

// 结构体类型变量可作为函数的形式参数
void fun2(struct Student stu)
{
    ....
}
```

结构体类型变量的初始化

```c
#include<stdio.h>
....
// 定义结构体数据类型struct Student
struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};
```

```c
void fun()
{
    // 定义结构体变量时，初始化各成员变量
    struct Student stu1={10101,"
     LiLin","M",18,90,"Xidian"},stu2
     ;
    ....
}
```

引用结构体变量的数据成员: 结构体变量名. 成员名

定义结构体类型和结构体变量后, 通过“.” 成员运算符, 引用其数据成员。

```
#include<stdio.h>
....
// 定义结构体数据类型struct Student
struct Student
{
  int num;
  char name[20];
  char sex;
  int age;
  float score;
  char addr[30];
};
```

```
void fun()
{
  // 定义结构体变量时, 初始化各成员变量
  struct Student stu1={10101,"LiLin
    ",'M',18,90,"Xidian"},stu2;
  printf("%d,%s,%c,%d,%.2f,%s",stu1
    .num,stu1.name,stu1.sex,stu1.
    age,stu1.score,stu1.addr);
  stu2.num=10102;
  // 注: 字符串用数组表示, 不能直接赋值。
  strcpy(stu2.name,"Wanghong");
}
```

对结构体变量的成员可以像普通变量一样进行各种运算

```
struct Date
{
  int month;
  int day;
  int year;
};
struct Student
{
  int num;
  char name[20];
  char sex;
  int age;
  struct Date birthday;
  char addr[30];
};
```

```
struct Student stu1,stu2;
stu1.num=10010;
stu1.sex='F';
stu1.birthday.month=6;// 逐级引用
stu1.age=18;
stu2.age=stu1.age+1;
stu2.age++;
stu2=stu1; // stu1中的各数据成员赋值给stu2
scanf("%d",&stu2.num);
gets(stu2.name); // 接受键盘输入的字符串
stu2.sex=stu1.sex; // 同性别
```

“.” 成员运算符在所有运算符中优先级最高，stu2.num 可以当作一个整体看待。

结构体指针变量

```c
// 定义结构体
struct Student
{
  int num;
  char name[20];
  char sex;
  int age;
  char addr[30];
};
```

```c
// stu2是结构体指针变量
struct Student stu1,*stu2;
stu1.num=10010;
stu1.sex='F';
// 引用结构体指针变量指向的数据成员
(*stu2).num=10011; // 一定要有括号
stu2->num=10011; // 或
scanf("%d",&(*stu2).num));
scanf("%d",&(stu2->num)); // 或
gets((*stu2).name);
gets(stu2->name); // 或
(*stu2).sex=stu1.sex;
stu2->sex=stu1.sex; // 或
```

结构体数组

```c
// 定义结构体
struct Student
{
  int num;
  char name[20];
  char sex;
  int age;
  char addr[30];
};
```

```c
// 结构体数组
struct Student stu[100];
scanf("%d",&stu[0].num);
gets(stu[0].name);
scanf("%c",&stu[0].sex);
stu[0].sex=getchar(); // 或
strcpy(stu[0].addr,"Xian");
```

例 1: 输入学生信息并输出统计数据

```c
struct Student
{
    int num;
    char name[20];
    // 5门课成绩
    float score[5];
};
```

```c
int main()
{
  struct Student stu;
  int i; float sum=0;
  scanf("%d%s",&stu.num,stu.name);
  for(i=0;i<5;i++)
     scanf("%f",&stu.score[i]);
  // 计算与输出
  printf("%d␣%s",stu.num,stu.name);
  for(i=0;i<5;i++)
  {
     printf("␣%.2f␣",stu.score[i]);
     sum=sum+stu.score[i];
  }
  printf("%.2f\n",sum/5.0);//平均成绩
  return 0;
}
```

例 2: 输入学生信息并输出统计数据 (函数实现)

```c
struct Student
{
  int num;
  char name[20];
  // 5门课成绩
  float score[5];
};
// 地址传递
void input(struct Student *stu)
{
  int i;
  scanf("%d%s",&(stu->num),stu->name);
  for(i=0;i<5;i++)
    scanf("%f",&(stu->score[i]));
    //scanf("%f",&((*stu).score[i]));
}
```

```c
void print(struct Student stu)
{
  int i; float sum=0;
  printf("%d␣%s",stu.num,stu.name);
  for(i=0;i<5;i++)
  {
     printf("␣%.2f␣",stu.score[i]);
     sum=sum+stu.score[i];
  }
  printf("%.2f\n",sum/5.0);
}
int main()
{
  struct Student stu;
  input(&stu); print(stu);
  return 0;
}
```

例 3: 输入学生信息并输出统计数据 (结构体数组), 输入

```c
// 定义结构体
struct Student
{
  int num;
  char name[20];
  // 5门课成绩
  float score[5];
};
```

```c
// 输入n个学生信息
//void input(struct Student stu[],int n)
void input(struct Student *stu, int n)
{
  int i,j;
  for(i=0;i<n;i++)
  {
     scanf("%d%s",&stu[i].num,stu[i].name);
     for(j=0;j<5;j++)
       scanf("%f",&stu[i].score[j]);
  }
}
```

例 3: 输入学生信息并输出统计数据 (结构体数组), 输出

```
//void print(struct Student stu[],int n)
void print(struct Student *stu,int n)
{
  int i,j; float sum;
  for(i=0;i<n;i++)
  {
    printf("%d␣%s",stu[i].num,stu[i].name);
    sum=0;//注意在此处置0
    for(j=0;j<5;j++)
    {
      printf("␣%.0f␣",stu[i].score[j]);
      sum=sum+stu[i].score[j];
    }
    printf("%.2f\n",sum/5.0);
  }
}
```

```
int main()
{
  struct Student stu[100];
  input(stu,2); // 输入2个学生
  print(stu,2); // 输出2个学生
  return 0;
}
```

例如, 输入两个学生的数据

1 zhang3 10 20 30 40 50

2 wang5 100 90 80 70 60

1 zhang3 10 20 30 40 50 30.00

2 wang5 100 90 80 70 60 80.00

例 4: 输入学生信息并输出统计数据 (结构体数组), 输入

```c
// 含平均成绩的结构体
struct Student
{
  int num;
  char name[20];
  // 5门课成绩
  float score[5];
  // 平均成绩
  float aver;
};
```

```c
// 输入并计算平均成绩，返回平均成绩最高者(假定唯一)
struct Student input(struct Student *stu, int n)
{
  int i,j,maxIndex=0; float sum;
  for(i=0;i<n;i++)
  {
    sum=0;
    scanf("%d%s",&stu[i].num,stu[i].name);
    for(j=0;j<5;j++)
    {
      scanf("%f",&stu[i].score[j]);
      sum=sum+stu[i].score[j];
    }
    stu[i].aver=sum/5.0; //填充平均成绩
    //非唯一最大者，这里添加条件。复杂条件可以设计独立函数
    if(stu[i].aver > stu[maxIndex].aver) maxIndex=i;
  }
  return stu[maxIndex];
}
```

例 4: 输入学生信息并输出统计数据 (结构体数组), 输出

```c
void print(struct Student *stu,int n)
{
  int i,j;
  for(i=0;i<n;i++)
  {
    printf("%d␣%s␣",stu[i].num,
                    stu[i].name);
    for(j=0;j<5;j++)
      printf("%.2f␣",stu[i].score[j]);
    printf("%.2f\n",stu[i].aver);
  }
}
```

```c
int main()
{
  struct Student stu[100],maxStu;
  maxStu=input(stu,3); // 输入3个学生
  print(stu,3); // 输出3个学生
  // 可复用print函数
  print(&maxStu,1); // 平均成绩最大者
  return 0;
}
```

例 5: 输入学生信息并输出统计数据 (结构体数组), 交换函数

```
// 交换两个结构体指针的内容，地址传递
// 两个结构体的各数据成员互相交换
void swap(struct Student *stu1, struct Student *stu2)
{
    struct Student tmp;
    tmp = *stu1; *stu1 = *stu2; *stu2 = tmp;
}
```

例 5: 输入学生信息并输出统计数据 (结构体数组), 排序

```
// 定义排序函数(选择法，降序)：按平均值排序，如果平均值相同，按照前2门课比较单科成绩
void sort(struct Student stu[],int n)
{
  int i,j,k;
  for(i = 0; i < n-1; i++)
  {
    k = i; // 未经排序较大者
    for(j = i + 1; j < n; j++)
    {
       if(stu[j].aver > stu[k].aver) k=j;
       else if(stu[j].aver == stu[k].aver)
       {
         if(stu[j].score[0] > stu[k].score[0] ||
            stu[j].score[1] > stu[k].score[1])
            k = j;
       }
    }
    if(k != i) swap(&stu[i],&stu[k]); // 交换
  }
}
```

lecture-17 主要内容

建立自己的数据类型

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- *while*(表达式){···}
- *do*{···}*while*(表达式);
- *for*(表达式 1; 表达式 2; 表达式 3){···}
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5} ;
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={ {1,2,3} ,{4,5,6} ,{7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
  - 连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)
  - 字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)
  - 获取字符串的长度 strlen(s)
  - 大小写转换 strlwr(s); strupr(s)
- 程序举例
  - 统计单词数
  - 求多个字符串中的较大者
  - 矩阵乘法
- 使用函数进行模块化程序设计
  - 函数三要素: 函数原型声明, 函数定义, 函数调用
  - 函数定义: int add(int a,int b){ }
- 实参和形参间的数据传递 (值传递和地址传递)
- 程序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 整数 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例续

训练编程逻辑思维方式:

- 领会结构化 (模块化) 编程思想。
- 用结构体管理相互关联的数据, 使其成为结构化的整体数据, 便于统一处理。
- 大问题分解为小问题, 设计函数解决小问题, 各个子函数彼此之间相互独立 (便于调试, 不易出错), 又可通过参数和返回值传递数据。
- 主程序调用各个子函数, 解决 “大” 问题。

lecture-18 主要内容

结构体编程举例续

- 课程介绍
- 导论简介
- C 语言程序设计
- 开发工具
- Algorithm + Data Structures = Programs
- 初识 C 语言程序 (作业)
- 初识 C 语言程序 (讲解作业)
- 数据的输入输出
- 数据输入再讨论
- 数据类型
- 运算符和表达式
- 数学库函数
- 顺序程序设计举例
- 选择结构和条件判断
- if 语句的一般形式
- 关系运算符及其优先次序
- 逻辑运算符
- 条件运算符和条件表达式
- 数学表达式与 C 语言表达式的不同
- 用 switch 语句实现多分支选择结构
- *while*(表达式){···}
- *do*{···}*while*(表达式);
- *for*(表达式 1; 表达式 2; 表达式 3){···}
- 循环的嵌套
- break,continue 改变循环执行的状态
- 循环结构程序设计举例
- 循环结构程序设计举例 (续)
- 定义数组: int a[10];
- 引用数组: int i=0; a[i]
- 初始化数组: int a[5]={1,2,3,4,5} ;
- 冒泡排序
- 定义和引用二维数组: int a[10][20];
- 引用数组: int i=0,j=0; a[i][j]
- 二维数组的初始化: int a[2][3]={ {1,2,3} ,{4,5,6} ,{7,8,9} }
- 用字符数组表示字符串 char s[81]
- 字符串处理函数 #include<string.h>
  - 连接和复制字符串 strcat(s1,s2); strcpy(s1,s2); strncpy(s1,s2,n)
  - 字符串比较 strcmp(s1,s2); strncmp(s1,s2,n)
  - 获取字符串的长度 strlen(s)
  - 大小写转换 strlwr(s); strupr(s)
- 程序举例
  - 统计单词数
  - 求多个字符串中的较大者
  - 矩阵乘法
- 使用函数进行模块化程序设计
  - 函数三要素: 函数原型声明, 函数定义, 函数调用
  - 函数定义: int add(int a,int b){ }
- 实参和形参间的数据传递 (值传递和地址传递)
- 程序举例
- 函数的嵌套调用
- 函数的递归调用
- 局部变量和全局变量
- 指针就是指向变量存储单元 (首) 地址的变量
- 定义指针变量: int *p;
- 变量的直接访问与间接访问
- 地址传递与值传递
- 数组与指针
- 指针与字符串
- 字符串数组
- 指针综合应用举例
- 期中考试总结
- 幂数 pow 函数问题
- 定义结构体数据类型: struct 结构体类型名 {...};
- 定义结构体类型变量: struct 结构体类型名 变量名;
- 引用结构体变量的数据成员: 结构体变量名. 成员名
- 结构体指针变量和结构体数组
- 结构体编程举例续

训练编程逻辑思维方式:

- 领会结构化 (模块化) 编程思想。
- 用结构体管理相互关联的数据, 使其成为结构化的整体数据, 便于统一处理。
- 大问题分解为小问题, 设计函数解决小问题, 各个子函数彼此之间相互独立 (便于调试, 不易出错), 又可通过参数和返回值传递数据。
- 主程序调用各个子函数, 解决 “大” 问题。

例 5: 上机题, 复试赛选

考研初试成绩公布后需要对 m 个学生的成绩进行排序,筛选出可以进入复试的前 n 名学生。排序规则为首先按照总分排序,总分相同则按英语单科成绩排序,总分和英语成绩也相同时考号小者排在前面。

现给出这 m 个学生的考研初试成绩,请筛选出可以进入复试的 n 名学生并按照排名从高到低的顺序依次输出。

输入说明输入为 m+1 行,第一行为两个整数 m 和 n,分别表示总人数和可以进入复试人数,m 和 n 之间用空格分隔,0<n<m<200。接下来为 m 行数据,每行包括三项信息,分别表示一个学生的考号 (长度不超过 20 的字符串)、总成绩(小于 500 的整数)和英语单科成绩(小于 100 的整数),这三项之间用空格分隔。

输出说明

按排名从高到低的顺序输出进入复试的这 n 名学生的信息。

输入样例	输出样例
5 3	XD20160005 360 75
XD20160001 330 65	XD20160003 340 60
XD20160002 330 70	XD20160002 330 70
XD20160003 340 60	
XD20160004 310 80	
XD20160005 360 75	

例 5: 复试赛选, 定义结构体

思路:

1. 定义考生结构体, 各考生组成结构体数组;
2. 按要求对结构体数组进行排序。

```
#include<stdio.h>
// 字符串比较函数strcmp(s1,s2);
#include<string.h>
#define N 100 // 估计的考生数
// 定义结构体
struct Student
{
    char no[20]; // 考号
    int  total;  // 总成绩
    int  english;// 英语成绩
};
```

例 5: 复试赛选 (结构体数组), 输入输出

```
// 输入m个考生信息
void input(struct Student *stus, int m)
{
  int i;
  for(i=0;i<m;i++)
    scanf("%s%d%d",
      stus[i].no,&stus[i].total,&stus[i].english);
}
// 输入n个考生信息
void print(struct Student *stus, int n)
{
  int i;
  for(i=0;i<n;i++)
    printf("%s␣%d␣%d\n",
      stus[i].no,stus[i].total,stus[i].english);
}
```

例 5: 复试赛选 (结构体数组), 交换函数

```
// 交换两个结构体指针的内容，地址传递
// 两个结构体的各数据成员互相交换
void swap(struct Student *stu1, struct Student *stu2)
{
    struct Student tmp;
    tmp = *stu1;
    *stu1 = *stu2;
    *stu2 = tmp;
}
```

例 5: 复试赛选 (结构体数组), 排序

```
// 定义排序函数(选择法, 降序): 按总成绩和英语单科成绩
void sort(struct Student stu[],int n)
{
  int i,j,k;
  for(i = 0; i < n-1; i++)
  {
    k = i; // 未经排序较大者
    for(j = i + 1; j < n; j++)
    {
       // 条件判断语句: 善用&&、||运算, 简化if else结构
       if (a[j].total>a[k].total ||
         (a[j].total==a[k].total &&  a[j].english>a[k].english) ||
         (a[j].total==a[k].total &&  a[j].english==a[k].english &&
    strcmp(a[j].no,a[k].no)<0))
         k = j;
    }
    if(k != i) swap(&stu[i],&stu[k]); // 交换
  }
}
```

例 5: 复试赛选 (结构体数组), 主函数

```
// 模块化程序设计，便于调试。
int main()
{
    struct Student stus[N];
    int m,n,i;
    scanf("%d%d",&m,&n);
    input(stus,m);
    // print(stus,m); // 查看输入数据是否正确
    sort(stus,m);
    // print(stus,m); // 查看排序结果是否正确
    print(stus,n);
    return 0;
}
```

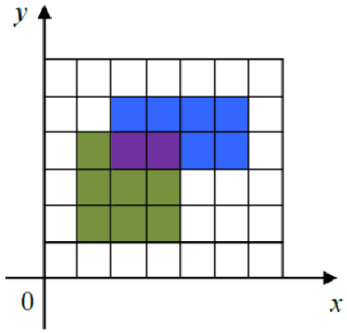
例 6: 上机题, 画图

在一个定义了直角坐标系的纸上,画一个 (x1,y1) 到 (x2,y2) 的矩形,指将横坐标范围从 x1 到 x2,纵坐标范围从 y1 到 y2 之间的区域涂上颜色。

下图给出了一个画了两个矩形的例子。第一个矩形是 (1,1) 到 (4, 4),用绿色和紫色表示。第二个矩形是 (2, 3) 到 (6, 5),用蓝色和紫色表示。图中,一共有 15 个单位的面积被涂上颜色,其中紫色部分被涂了两次,但在计算面积时只计算一次。在实际的涂色过程中,所有的矩形都涂成统一的颜色,图中显示不同颜色仅为说明方便。给出所有要画的矩形,请问总共有多少个单位的面积被涂上颜色。

输入说明: 输入的第一行包含一个整数 n,表示要画的矩形的个数,1<=n<=100,接下来 n 行,每行 4 个非负整数,分别表示要画的矩形的左下角的横坐标与纵坐标,以及右上角的横坐标与纵坐标。0<= 横坐标、纵坐标 <=100。

输出说明: 输出一个整数,表示有多少个单位的面积被涂上颜色。



输入样例

2

1 1 4 4

2 3 6 5

输出样例

15

例 6: 画图 (结构体数组), 定义结构体

思路:

1. 二维数组表示网格;
2. 定义矩形结构体, 各矩形组成结构体数组;
3. 计算最大矩形, 其覆盖的单元格初始化为 0;
4. 每个矩形覆盖的单元格标记为 1,
5. 最大矩形被标记的单元格即是被涂色的单位面积。

```
#define N 100
//定义矩形结构体
struct Rec
{
    int leftBottomX;
    int leftBottomY;
    int rightTopX;
    int rightTopY;
};
```

例 6: 画图 (结构体数组), 输入输出

```
// 输入n个矩形
void input(struct Rec *recs, int n)
{
  int i;
  for(i=0;i<n;i++)
     scanf("%d%d%d%d",&recs[i].leftBottomX, &recs[i].leftBottomY, &recs
    [i].rightTopX,&recs[i].rightTopY);
}
// 输出n个矩形
void print(struct Rec *recs, int n)
{
  int i;
  for(i=0;i<n;i++)
    printf("%d,%d,%d,%d\n",recs[i].leftBottomX,recs[i].leftBottomY,recs
    [i].rightTopX,recs[i].rightTopY);
}
```

例 6: 画图 (结构体数组), 计算最大矩形

```
// 返回最大矩形
struct Rec Largest(struct Rec res[], int n)
{
   struct Rec largest=res[0];
   int i;
   for(i=1;i<n;i++)
   {
      if(res[i].leftBottomX < largest.leftBottomX)
          largest.leftBottomX=res[i].leftBottomX;
      if(res[i].leftBottomY < largest.leftBottomY)
          largest.leftBottomY=res[i].leftBottomY;
      if(res[i].rightTopX > largest.rightTopX)
          largest.rightTopX=res[i].rightTopX;
      if(res[i].rightTopY > largest.rightTopY)
          largest.rightTopY=res[i].rightTopY;
   }
   return largest;
}
```

例 6: 画图 (结构体数组), 网格 grid[N][N] 初始化与标记

```
// rec区域, grid数组元素置1(标记)
void grid1(struct Rec rec, int grid[][N])
{
   int i,j;
   for(i=rec.leftBottomY;i<rec.rightTopY;i++)
   {
       for(j=rec.leftBottomX;j<rec.rightTopX;j++)
           grid[i][j]=1;
   }
}
// largest区域, grid数组元素初始化为0
void grid0(struct Rec largest, int grid[][N])
{
   int i,j;
   for(i=largest.leftBottomY;i<largest.rightTopY;i++)
   {
      for(j=largest.leftBottomX;j<largest.rightTopX;j++)
           grid[i][j]=0;
   }
}
```

例 6: 画图 (结构体数组), 计算被标记的网格

```
// largest区域, 返回grid数组元素为1的单元数量, 即被覆盖的单位面积数
int gridNum(struct Rec largest, int grid[][N])
{
    int i,j,num=0;
    for(i=largest.leftBottomY;i<largest.rightTopY;i++)
    {
        for(j=largest.leftBottomX;j<largest.rightTopX;j++)
            if(grid[i][j]) num++;
    }
    return num;
}
```

例 6: 画图 (结构体数组), 主函数

```
// 模块化程序设计，便于调试。
int main()
{
    int grid[N][N],n,i,num;
    struct Rec recs[N],largest;
    scanf("%d",&n);
    input(recs,n);
    // print(recs,n); // 检查输入
    largest=Largest(recs,n); // 计算最大矩形
    // print(&largest,1); // 查看最大矩形是否正确
    grid0(largest,grid); // 初始化
    for(i=0;i<n;i++)
       grid1(recs[i],grid); // 标记网格，置1
    printf("%d\n",gridNum(largest,grid)); // 统计输出
    return 0;
}
```

例 7: 世界杯排名

给出世界杯一个小组的小组赛成绩,请找出最终进入淘汰赛的球队。

小组赛积分规则:胜一场积 3 分、平一场积 1 分、负一场积 0 分。

小组赛排名规则如下:

1. 首先按照积分排名,积分越高,排名越高
2. 积分相同的队按照净胜球 (进球数减去失球数) 排名,净胜球越多,排名越高

输入说明: 输入数据第一行为两个整数 N 和 M,N 表示小组中队伍数量 (N<=32),M 表示出线名额 (M<=16)。接下来是 N 行数据,每一行表示一只队伍的小组赛比赛信息,依次为队名 (长度不超过 20 的字符串)、胜场数、平局数、负场数、进球数、失球数。输入数据保证没有两个球队的积分和净胜球完全相同。输出说明: 根据题目给出的积分排名规则,按照排名从高到低顺序依次输出出线球队的排名(从 1 开始排名)、队名、积分和净胜球,各个字段之间用空格分隔。

输入样例	输出样例
4 2	1 Germany 7 5
Germany 2 1 0 7 2	2 America 4 0
Portuguesa 1 1 1 4 7	
America 1 1 1 4 4	
Ghana 0 1 2 4 6	

例 7: 世界杯排名, 定义结构体

思路:

1. 定义结构体存储球队相关信息;
2. 各球队组成结构体数组;
3. 按要求对结构体数组进行排序。

```
// 估计最大数
#define N 32
#define M 16
```

```
// 定义结构体类型，存储球队相关信息
struct Team
{
  char name[20];
  int win,draw,lose; // 胜，平，负场数
  // 进球数，失球数
  int goalsNum,losesNum;
  int grades; // 积分
  int goals;  // 净胜球
};
```

例 7: 世界杯排名, 输入

```
// 输入n个球队信息, 填充结构体成员
void input(struct Team *teams, int n)
{
    for(i = 0; i < n; i++)
    {
        scanf("%s%d%d%d%d%d",teams[i].name,&teams[i].win,&teams[i].draw,&
     teams[i].lose,&teams[i].goalsNum,&teams[i].losesNum);
        // 填充其它数据成员
        teams[i].grades =  teams[i].win*3 + teams[i].draw*1;
        teams[i].goals = teams[i].goalsNum - teams[i].losesNum;
    }
}
```

例 7: 世界杯排名, 交换函数

```
// 交换两个结构体指针的内容, 地址传递
// 两个结构体的各数据成员互相交换
void swap(struct Team *team1, struct Team *team2)
{
    struct Team tmp;
    tmp = *team1;
    *team1 = *team2;
    *team2 = tmp;
}
```

例 7: 世界杯排名, 排序

```
// 定义排序函数(选择法, 降序): 按总成绩和英语单科成绩
void sorts(struct Team a[], int n)
{
   int i,j,k;
   for(i = 0; i < n - 1; i++)
   {
      k = i; // 未经排序较大者
      for (j = i+1; j < n; j++)
      {
         // 条件判断语句: 善用&&、||运算, 简化if else结构
        if (a[j].grades > a[k].grades ||
           (a[j].grades == a[k].grades &&  a[j].goals > a[k].goals))
              k = j;
      }
      if (k != i) swap(&a[i],&a[k]); // 交换
   }
}
```

例 7: 世界杯排名, 主函数

```
// 模块化程序设计，便于调试。
int main()
{
    struct Team teams[N]; // 以估计最大数，定义结构体数组
    int n,m,i;
    scanf("%d%d",&n,&m);
    input(teams,n);
    sorts(stus,n); // 排序
    // 输出前m个排序后的球队
    for(i = 0; i < m; i++)
        printf("%d␣%s␣%d␣%d\n",i+1,teams[i].name,teams[i].grades,teams[i
     ].goals);
    return 0;
}
```

欢迎批评指正！