

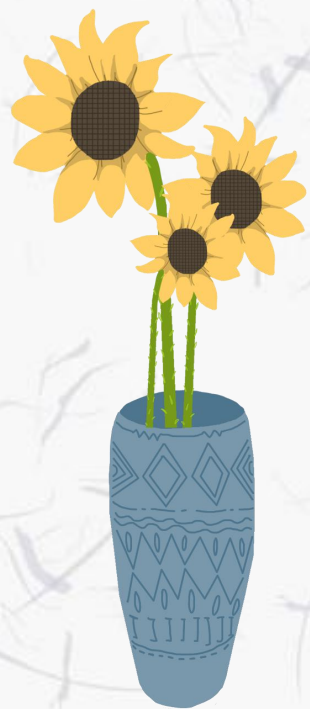
数据结构与算法

C语言的基础知识

王 昭

北京大学信息学院软件研究所

wangzhao@pku.edu.cn





程序猿的信仰： 用“代码”推动“未来”
重视细节犹如生命



主要内容



- C语言简介
- 数据类型与声明
- 运算符与表达式
- 数组
- 函数
- 程序流程
- 指针
- 结构体



C 语言

- C语言诞生于70年代初期，流行于80年代末期。它既有高级语言的面向用户的特点，又有低级语言的管理硬件的功能，因而得到广泛的应用。
- 许多经典的软件系统（例如，计算机辅助设计软件 **AUTOCAD**，数学软件系统 **Mathematica** 等），以及许多语言的编译系统本身就是用 C 语言开发的。众所周知的**UNIX**就是用C语言编写的。
- C语言开发环境有很多种，如：**DEV C++**、**CodeBlock** 、**Visual C++ 6.0**、**Visual C++ 2017/2012/2010/2008**、**Eclipse IDE for C/C++ Developers** 等。
- 特点：
 - 语言简洁、紧凑。
 - 语法灵活。
 - 运算符丰富。
 - 允许直接访问物理地址。
 - 目标程序效率高、可移植性好



```
#include <stdio.h>
int main( )
{
    printf("hello,world! \n");
    return 0;
}
```

一切伟大的思想和行动
都有一个微不足道的开始



C 程序的基本框架

① 预编译声明（宏定义、头文件）

② 自定义函数的声明

③

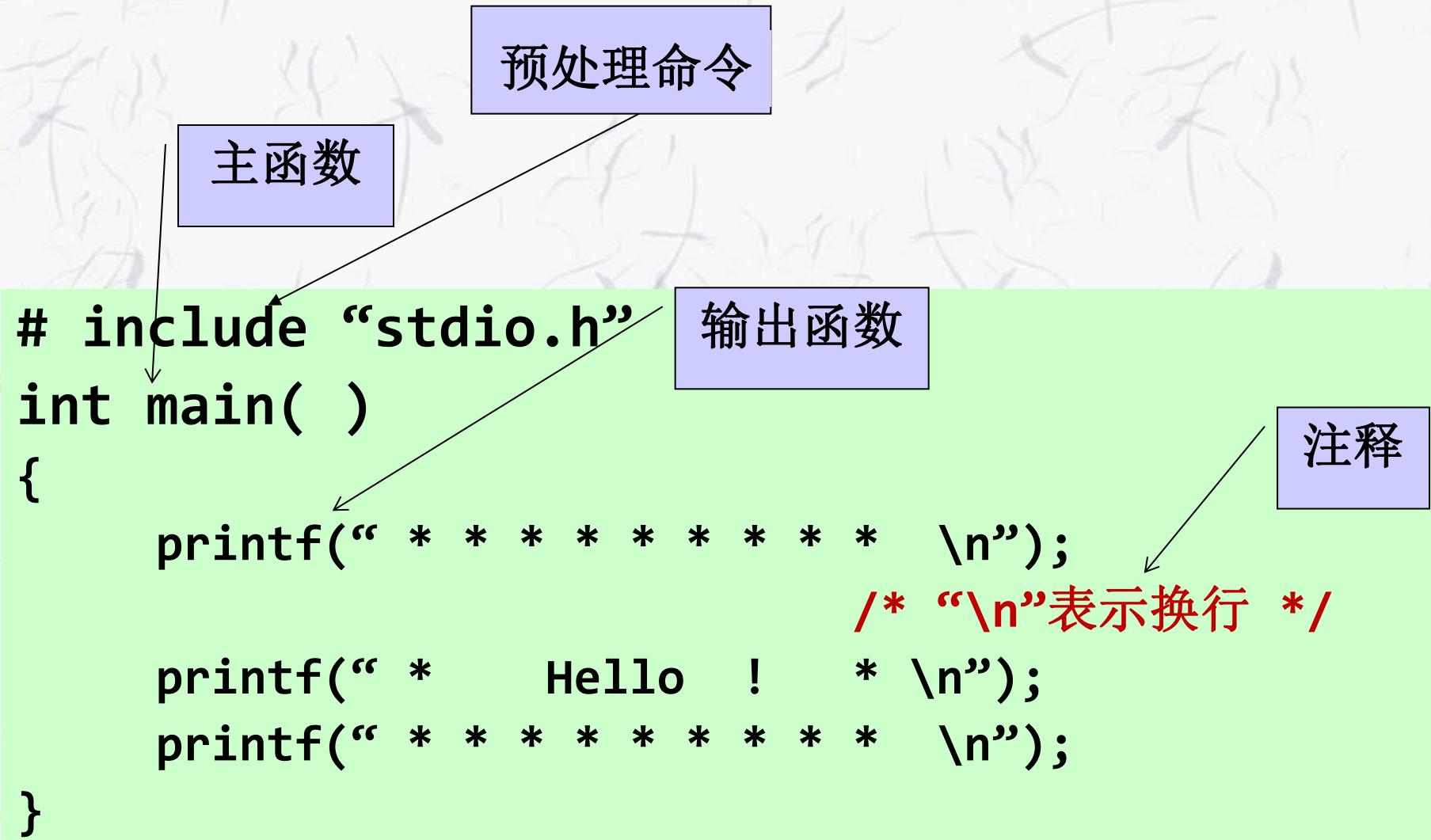
```
int main( )  
{  
    ... ..  
    return 0;  
}
```

主函数的定义

④ 自定义函数的定义



简单的C语言源程序



C程序的构成

- 一个C语言的源程序通常由一个或若干个函数构成，函数是C程序的基本单位，这些函数可以包含在一个或者多个源文件内。
- 一个C程序中必须有一个main函数，它可以放在程序的任何地方。程序总是从main函数开始执行。
- 函数可以是系统提供的库函数，也可以是用户根据自己需要编制的自定义函数。

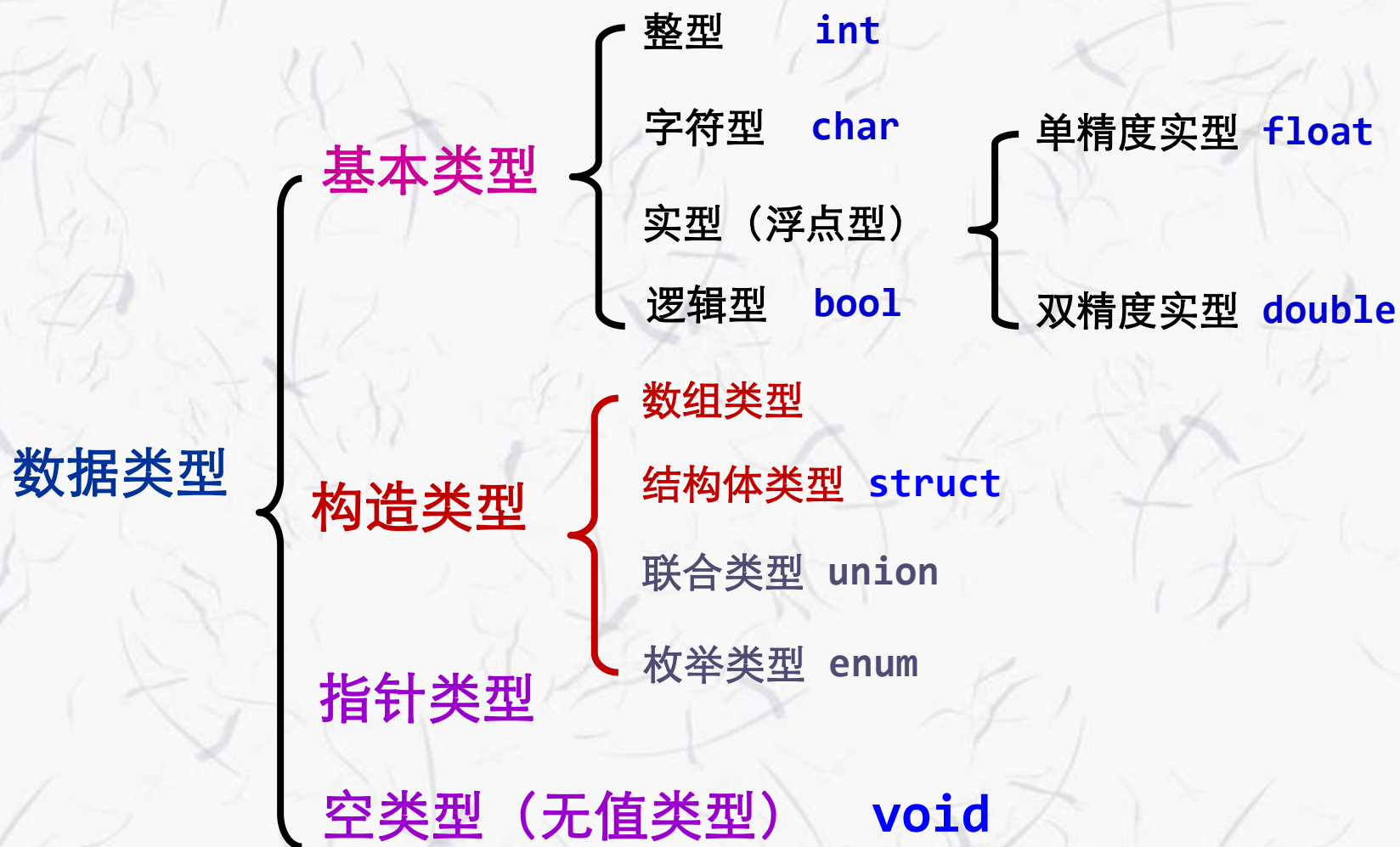


主要内容

- C语言简介
- 数据类型与声明
- 运算符与表达式
- 数组
- 函数
- 程序流程
- 指针
- 结构体



数据类型与声明



变量

- 在程序运行过程中其值可以改变的**量**叫做**变量**。
- 变量要先定义（即使用**类型标识符作说明**）再使用。
- `int num;`
- `char name;`



常量

- 在程序运行过程中其值不能改变的量叫做**常量**。
 - 常量分为整型常量（如果是长整型，要在数字后面加L）、实型常量、字符型常量等。
 - `const float PI=3.1415926;`
- **符号常量**:
 - 值在本文件内不能改变，也不能再赋值。
 - `#define MAXSIZE 16`
 - C语言规定标识符只能由字母或下划线打头，后面跟字母、数字或下划线，大小写不通用。（在Turbo C中标识符的长度可以超过8个字符）
 - 一般符号常量名一般用**大写**，变量名用**小写**，以示区别。
- **const**常变量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查，而对后者只进行字符替换，没有类型安全检查，并且在字符替换时可能会产生意料不到的错误。



整型数据

- C语言中的整型常量可以用十进制、八进制和十六进制表示。八进制数以0开头，十六进制数以0x开头。（0为数字0）
- C语言中的整型变量分为基本型、短整型、长整型、无符号型。
- 一般短整型不大于基本型，长整型不小于基本型。



数据类型

● 基本数据类型 (Primary Data Types)

数据类型说明	数据类型标识符	所占位数	其他
字符类型	char	8位 (1字节)	还可以用 unsigned 来限定 char 类型和所有整数类型。unsigned 限定的数总是正数或零。
整数类型	int	与机器相关, 现在通常为32位(4字节)	
短整数类型	short	int的一半	
长整数类型	long	int的2倍, 但现在与int一样, 为32位	
单精度浮点类型	float	32位 (4字节)	
双精度浮点类型	double	64位 (8字节)	
无类型	void		函数无返回值

- 一个数据类型所占的具体字节数, 可以通过 **sizeof** 运算符来确定 **sizeof(int)**

- **long long**, 有符号 64位整数, 所占8个字节(Byte), 输出格式**lld%**
- **-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807**



整型数据的分类

- 按照数据的表示范围分类
 - ▣ 基本型：以**int**表示
 - ▣ 短整型：以**short int**或**short**表示
 - ▣ 长整型：以**long int** 或**long** 表示；以**long long int** 或**long long**表示；
- 按照数据有无符号分类
 - ▣ 有符号型：表示某个范围内的整数
[signed] int;[signed] short;[signed] long;[signed] long long
 - ▣ 无符号型：表示某个范围内的正整数
unsigned int;unsigned short;unsigned long;unsigned long long



例如

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

```
int main ( )
```

```
{   int a,b;
```

```
    a=022;
```

```
    b=0x12;
```

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

```
    return 0;
```

```
}
```

格式说明符

格式说明符

(022为八进制数22，0x12为十六进制12)

E:\wz2\course\compt2020s\cpp\ch07-08\printcl1.exe

18, 18

Process exited after 0.1603 s
请按任意键继续. . .

实型数据

- 实型又称为浮点型，如3.1415926、5.0等。
- 它可以使用指数形式表示，例如：3e5、-3.5e7、5e-3等。
- 实型数据默认的显示格式是小数点后6位。

类型	类型标识符	占用字节数	能表示数值的有效数字	数值范围
单精度实型	float	4	7位	$\pm 10^{\pm 38}$
双精度实型	double	8	15~16位	$\pm 10^{\pm 308}$



例如： 将一个double型变量的值输出到控制台

```
#include <stdio.h>

int main( ){
    double a;

    a = 1.987654321;
    printf("%f\n", a);
    return 0;
}
```

E:\wz2\course\compt2020s\cpp\ch07-08\printad0.exe

a=1.987654

Process exited after 0.1089 seconds
请按任意键继续. . .



C程序中的 常量

- 用文字串表示的，它区分为不同的类型：
 - 整型常量：123, -123
 - 长整型常量：123L, -123L
 - 无符号整型常量：10000U
 - 16进制常量：0x12, -0x12
 - 8进制常量：022, -022
 - 单精度浮点常量：1.23f, -1.23e12f
 - 双精度浮点常量：1.23, -1.23e12
- 在C中，除了直接写出常量的值之外，还可以通过预编译命令“#define”把一个标识符定义为常量，其定义格式为：**#define PI 3.14159**



字符型数据

- C语言中的字符型数据有：
 - 单字符常量
 - 转义字符常量
 - 字符串常量
 - 字符变量



单字符常量

- 单字符常量用单引号括起来
- 类型标识符为**char**，输出时用**%c** 表示。
- 按**ASCII**码存储，每个字符占一个字节。
- 给字符变量赋值时可以赋予字符的**ASCII**码。
- 例如：**char a=65**的结果是**a**变量中存放一个字母**A** 。

```
int main ( )  
{  
    char a='a',b='b',c='c',d=68;  
    a=a+1;b=b+2;c=c+3;d=d+4;  
    printf(“%c%c%c%c\n”,a,b,c,d);  
}
```

运行结果： bdfH



转义字符常量

- 是以“\”开头的字符序列。例\ n

字符形式	功 能
\n	换行
\t	横向跳格（即跳到下一个输出区）
\v	纵向跳格
\b	退格
\r	光标移到当前行开头
\f	走纸换页
\\	反斜杠字符“\”
\'	单引号字符
\“	双引号字符
\ddd	1到3位八进制字符。如\ 123表示八进制数123，即十进制数83
\xhh	1到2位十六进制字符。如\x21，表示十六进制数21，即十进制数33



字符串常量

- 字符串常量用双引号括起来 “CHINA”

C	H	I	N	A	\0
---	---	---	---	---	----

- ✓ `char c; c='a';`
 - ✗ `char c; c="a";`
- 用数组来存放，数组长度大于字符串长度。
 - `char c[10]="China";`
- 输入输出 `printf("%s",c); scanf("%s",c);`
- 字符串处理函数：

`puts(c); gets(c); strcat(str1,str2); strcpy(str1,str2);`



变量

- 可以在定义变量的同时为变量赋值。

- 例如: `int a=5,b=5,c=6;`

- 可以使用赋值语句为变量赋值。

- 例如:

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

```
a=b=5;
```

```
c=6;
```

- 不能在定义变量语句时使用连续的赋值运算。如:

```
int a=b=c=6;是错误的。
```



例如

```
int main()  
{  int a=5,b=6;  
    a=a+b;  
    b=a+b;  
    printf("a=%d,b=%d\n",a,b);  
    a=a+1;  
    b=b-2;  
    printf("a=%d,b=%d\n",a,b);  
    return 0;  
}
```

结果显示:

a=11,b=17

a=12,b=15



局部变量和全局变量-1

- 从作用域的角度分，有局部变量和全局变量。
- 局部变量：在一个函数内部定义的变量是内部变量，只在本函数内有效。

○ `int main()`

```
{ int m,n;  
  ...}
```

○ `float f1(int a)`

```
{ int b,c;  
  ...}
```



局部变量和全局变量-2

- 程序的编译单位是源程序文件，一个源文件包含多个函数，在函数外定义的变量称为外部变量，也称为全局变量。
- 它的有效范围为从定义变量的位置开始到本源文件结束。

```
int P=1,Q=5;
float f1(int a)
{  int b,c;
    ...}
int main()
{  int m,n;
}
```

- 全局变量名的第一个字母用大写表示。



变量的存储类别-1

- 从变量值**存在的时间**（即生存期）角度来分，可以分为**静态存储方式**和**动态存储方式**。
 - **静态存储方式**：在程序运行期间分配固定的存储空间的方式。
 - **动态存储方式**：在程序运行期间根据需要进行动态的分配存储空间的方式。
- 内存空间分为**程序区**、**静态存储区**、**动态存储区**。
- 变量的**存储类别**指的是数据在内存中的存储方式，具体包括：
 - 自动的(auto)
 - 静态的(static)
 - 寄存器的(register)
 - 外部的(extern)



变量的存储类别-2

- 局部变量
 - 自动变量 **auto**，即动态局部变量（离开函数，值就消失）
`auto int b;` 等价于 `int b;`
 - 静态局部变量（离开函数，值仍保留）
 - 寄存器变量（离开函数，值就消失）
- 形参可以定义为自动变量和寄存器变量
- 全局变量
 - 静态外部变量（只限**本文件**使用）
 - 外部变量（允许**其他文件**引用）

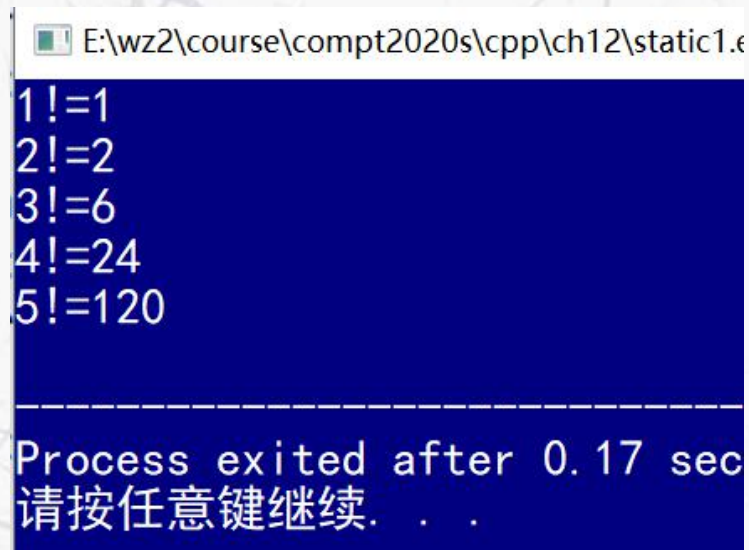


Example-static

```
int fac(int n)
{   static int f=1;
    f=f*n;
    return(f);
}

int main()
{   int i;
    for (i=1;i<=5;i++)
        printf("%d!=%d\n",i,fac(i));
    return 0;
}
```

- 使用场合：
 - ① 需要保留函数上一次调用结束时的值。
 - ② 初始化后，变量只被引用而不改变其值。



E:\wz2\course\compt2020s\cpp\ch12\static1.e

```
1!=1
2!=2
3!=6
4!=24
5!=120

-----
Process exited after 0.17 sec
请按任意键继续. . .
```



Example-register

```
int fac(int n)
{   register int i,f=1;
    for (i=1;i<=n;i++)
        f=f*i;
    return(f);
}

int main()
{   int i;
    for (i=1;i<=5;i++)
        printf("%d!=%d\n",i,fac(i));
    return 0;
}
```

- 说明:
- 频繁引用的变量存在寄存器中的存取速度高
- 只有局部自动变量和形式参数可以作为寄存器变量。
- 寄存器数目有限，不能定义任意多个寄存器变量。
- 局部静态变量不能定义为寄存器变量。
- 了解即可



Example -extern

- 外部变量作用范围从定义处到文件结束，如果在此前用到就需要声明。
- 了解即可

在一个文件内声明外部变量

```
int max(int x,int y)    /* 定义max函数 */
{   int z;
    z=x>y?x:y;
    return(z);
}
```

```
int main()
{   extern int A,B;    /* 外部变量声明 */
    printf("%d",max(A,B));
    return 0;
}
```

```
int A=13,B=-8;        /* 定义外部变量 */
```

运行结果如下： 13

Example-extern

- 在**多文件**的程序中声明外部变量，将外部变量的作用域扩展到其他文件。
- 给定b的值，输入A和m，求 $A \times b$ 和 a^m 的值。

文件file2.c中的内容为:

```
extern A;
```

```
power(int n)
```

```
{  int i,y=1;
```

```
    for (i=1;i<=n;i++)
```

```
        y*=A;
```

```
    return(y);
```

```
}
```

文件file1.c中的内容为:

```
int A;
```

```
int main()
```

```
{  int b=3,c,d,m;
```

```
    printf(" enter the number A  
and its power m:\n");
```

```
    scanf ("%d %d",&A,&m);
```

```
    c=A*b;
```

```
    printf("%d * %d=%d\n",A,b,c);
```

```
    d=power(m);
```

```
    printf("%d **  
%d=%d\n",A,m,d);
```

```
}
```

了解即可



函数运行时内存布局



堆：堆是程序运行过程中动态分配的数据块，是不连续的内存区域。（`malloc\calloc`）

栈：存储程序运行过程中动态分配的数据块，在Windows下，栈是向低地址扩展的数据结构，是一块连续的内存的区域。用于存储函数调用所传递的参数、函数的返回地址、函数的局部变量等。



```
int a = 0; //全局初始化区
```

```
int main( ){
```

```
    int b; //栈
```

```
    char s[] = "abc"; // 栈
```

```
    char *p1; //栈
```

```
    char *p2; //栈
```

```
    char *p3 = s; //栈
```

```
        static int c =0; //全局（静态）初始化区
```

```
    b=3;
```

```
    p1 = (char *)malloc(30);
```

```
    p2 = (char *)malloc(20);
```

```
        //分配得来30和20字节的区域就在堆区。
```

```
    strcpy(p1, "1234");
```

```
    return 0;
```

```
}
```



堆栈的申请方式

- **stack**

由系统自动分配。例如，声明在函数中一个局部变量 `int b;` 系统自动在栈中为**b**开辟空间。只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。

- **heap**

需要程序员自己申请，并指明大小，在c中**malloc**函数如**p1 = (char *)malloc(30);** 操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序。



```

#include "stdio.h"
int g1=0, g2=0, g3=0;
int main()
{ //打印出各个变量的内存地址
    static int s1=0, s2=0, s3=0;
    int v1=0, v2=0, v3=0;
    printf("0x%08x\n",&v1); //打印各本地变量的内存地址
    printf("0x%08x\n",&v2);
    printf("0x%08x\n\n",&v3);
    printf("0x%08x\n",&g1); //打印各全局变量的内存地址
    printf("0x%08x\n",&g2);
    printf("0x%08x\n\n",&g3);
    printf("0x%08x\n",&s1); //打印各静态变量的内存地址
    printf("0x%08x\n",&s2);
    printf("0x%08x\n\n",&s3);
    return 0;
}

```

```

0x0012ff44
0x0012ff40
0x0012ff3c
0x00427c48
0x00427c4c
0x00427c50
0x00427c54
0x00427c58
0x00427c5c

```



```

#include "stdio.h"
int g1=0, g2=0, g3=0;
int main()
{ //打印出各个变量的内存地址
    static int s1=0, s2=0, s3=0;
    int v1=0, v2=0, v3=0;
    printf("0x%08x\n",&v1); //打印各本地变量的内存地址
    printf("0x%08x\n",&v2);
    printf("0x%08x\n\n",&v3);
    printf("0x%08x\n",&g1); //打印各全局变量的内存地址
    printf("0x%08x\n",&g2);
    printf("0x%08x\n\n",&g3);
    printf("0x%08x\n",&s1); //打印各静态变量的内存地址
    printf("0x%08x\n",&s2);
    printf("0x%08x\n\n",&s3);
    return 0;
}

```

```

0x0062fe1c
0x0062fe18
0x0062fe14
0x00407030
0x00407034
0x00407038
0x0040703c
0x00407040
0x00407044

```



主要内容

- C语言简介
- 数据类型与声明
- 运算符与表达式
- 数组
- 函数
- 程序流程
- 指针
- 结构体



运算符与表达式

- C语言的运算符：
 - 算术运算符、关系运算符、逻辑、位、赋值、条件、逗号、指针、求字节数、强制类型换算、分量、下标、自加自减、函数调用等。
 - 本章介绍算术运算、赋值运算和逗号运算。
- 优先级
当表达式中包含有不同类型的运算符时的运算顺序。
- 结合性
表达式中有若干个同一类型的运算符时的运算顺序。



算术运算符及表达式

＋、－、*、/、% （取余）

例如：7%4 得 3

注意：float型数据不能做%运算

优先级：先*、/、%，后＋、－

结合性：自左至右



自增和自减运算

++ 自增、-- 自减

例如：++a 、 a ++ 相当于 $a = a + 1$

-- a 、 a -- 相当于 $a = a - 1$

但是：++a 是先自增再使用， a++是先使用再自增。

注意：float型数据不能做++或--运算。

结合性：自右至左



例1

```
int main()
{   int a=5,b=6;
    printf("a=%d,b=%d\n",a,b);
    printf("%d,%d\n",++a,--b);
    printf("a=%d,b=%d\n",a,b);
    printf("%d,%d\n",a++,b--);
    printf("a=%d,b=%d\n",a,b);
    return 0;
}
```

结果显示:

a=5,b=6

6,5

a=6,b=5

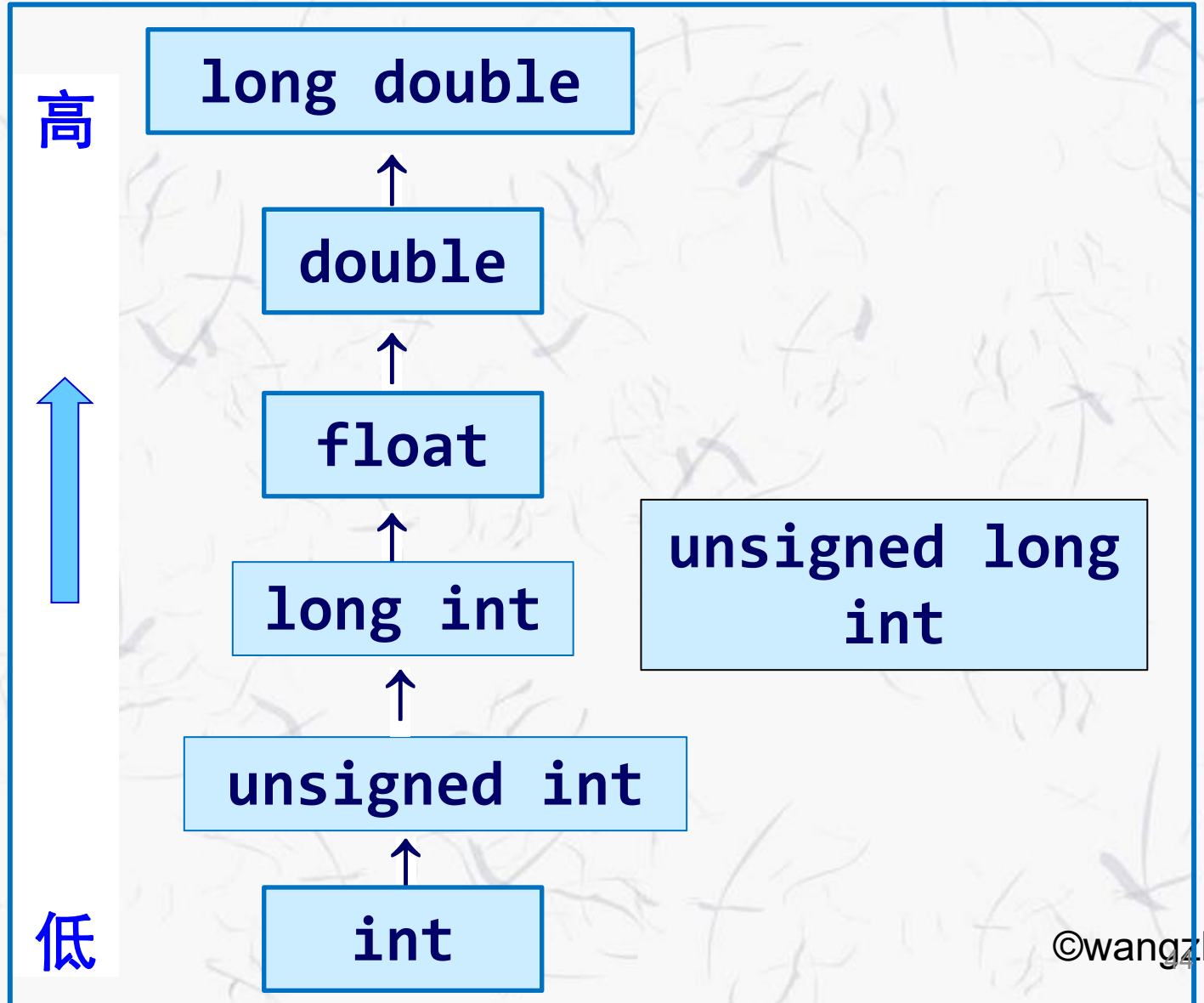
6,5

a=7,b=4



自动类型转换

- 一般来说，二元运算符的两个操作数类型不同，自动转换是把“**较窄的**”操作数转换为“**较宽的**”操作数。



例如

```
int main()
{
    unsigned a;
    int b;
    float i=3.16;
    //a=65535; /*216-1*/
    a=4294967295; /* 232-1 */
    b=a;
    printf("%d",b);
    b=i;
    printf("%d",b);
    return 0;
}
```

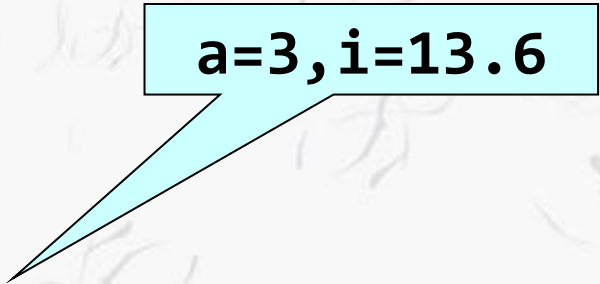
unsigned->int: 不符号
扩展 b=-1

实数->整数: 舍弃小数部
分; b=3

强制类型转换

- 必要时可以在表达式中对数据类型进行强制转换。注意转换的只是表达式的值，而变量的数据类型仍然不变。
- 例如：

```
int main( )  
{  
    int a;float i;  
    i=13.6;  
    a=(int)i%5;  
    printf("a=%d,i=%f\n",a,i);  
}
```



a=3,i=13.6



赋值运算符及表达式

- $=, +=, -=, *=, /=, \% =$, ...
- 对于数值型数据或字符型数据，当赋值运算符两边的数据类型不一致时，系统会自动进行转换。
 - ① 当数值赋给字符型变量时，按照ASCII码进行转换。
 - ② 当字符型数据赋给整型变量时，将其ASCII码存入双字节的低8位中，高8位补0。
 - ③ 当实型数据赋给整型变量时，舍弃小数部分。
 - ④ 当整型数据赋给实型变量时，数值不变，添加小数位数。
 - ⑤ 当有符号数赋给长度相同的无符号数时，连原有的符号也作为数值一起传送。



复合的赋值运算符

- $a+=3$ 等价于 $a=a+3$
- $a\%=3$ 等价于 $a=a\%3$
- $a*=3+2$ 等价于 $a=a*(3+2)$



逗号表达式

- 将几个表达式用逗号分开，分别计算。整个表达式的值是最后一个表达式的值。
- 例如： $a=3*5, a*4$
结果： $a=15$ 表达式的值是： 60
- 例如： $b=(a=3*5, a*4), a+5$
结果： $a=15, b=60$ 表达式的值是 20
- 逗号运算符是所有运算符中级别最低的。
- 三元运算符： 表达式1? 表达式2:表达式3
结合方向：从右向左， $a>b?a:c>d?c:d$
相当于 $a>b?a: (c>d?c:d)$



主要内容

- C语言简介
- 数据类型与声明
- 运算符与表达式
- 数组
- 函数
- 程序流程
- 指针
- 结构体



数组

- 数组的定义方式

- 一维数组的定义方式为:

类型说明符 数组名[常量表达式]

`int a[5];`

5个元素是: `a[0]`、`a[1]`、`a[2]`、`a[3]`、`a[4]` 。

- 二维数组的定义方式为:

类型说明符 数组名[常量表达式] [常量表达式]

`float a[3][2]`, `a`数组有三行两列共6个元素, 分别是: `a[0][0]`、`a[0][1]`、`a[1][0]`、`a[1][1]`、`a[2][0]`、`a[2][1]`;



关于数组的说明

- 数组名的命名规则和变量名相同，下标从0开始，方括号内是元素个数。
- 方括号内的常量表达式的值必须是整型数，不能是变量。
- 当定义数组语句中不同时给变量赋值时，方括号内不得为空。
- 二维数组在内存中是按行存放的。
- 数组名代表的是该数组的起始地址。
- C语言允许使用多维数组。



例如

```
int main ( )  
{  
    int i,a[4];  
    for(i=0;i<=3;i++)  
        a[i]=i;  
    for(i=0;i<=3;i++)  
        printf("%d,",a[i]);  
    printf("\n");  
    return 0;  
}
```

运行结果为: 0,1,2,3,



数组的赋值

- 在定义数组后数组中的各个变量自动取随机数。
 - `int b[10]; b[1]=3;`
 - 将3赋给**b[1]**，其他元素为随机数 。
- 可以在定义数组时对数组中的全部变量或部分变量赋值。也可以在以后的语句中为变量赋值。
 - `int a[5]={3,4,5,6,7};` 将五个数依次赋给a数组
- 在定义数组时为部分变量赋值后，其他元素为0 。
- `int b[10]={0}`
- `int b[10]={3,2,1,0};`
- 将四个数依次赋给**b[0]~b[3]**，其他元素为0
- 在给全部元素赋初值时，可以不用给出长度。
 - `int a[]={1,2,3,4,5}`



字符数组

- 字符数组的定义:

- `char c[9];`

- `c[0]='I'; c[1]=' '; c[2]='a'; c[3]='m'; c[4]=' '; c[5]='f'; c[6]='i'; c[7]='n'; c[8]='e';`

I		a	m		f	i	n	e
---	--	---	---	--	---	---	---	---

- 字符数组的初始化:

- `Char c[10]={ 'I', ' ', 'a', 'm', ' ', 'f', 'i', 'n', 'e' };`

- `Char c[]={ 'I', ' ', 'a', 'm', ' ', 'f', 'i', 'n', 'e' };`

I		a	m		f	i	n	e
---	--	---	---	--	---	---	---	---



字符串

- C语言中,把字符串作为字符数组来处理.
- 为了测定字符串的实际长度,C语言规定了一个字符串结束标志,以‘\0’代表。
- 系统自动为字符串常量自动加一个 ‘\0’作为结束符。

- `char c[]={" I am fine"};`

- `char c[]="I am fine";`

- `char c[]={'I',' ','a','m',' ','f','i','n','e','\0'};`

I		a	m		f	i	n	e	\0
---	--	---	---	--	---	---	---	---	----

- `char c[10]="China";`

C	h	i	n	a	\0	\0	\0	\0	\0
---	---	---	---	---	----	----	----	----	----

- `char c[5]={ 'C', 'h' , 'i' , 'n' , 'a' }`

- `char c[6]={ 'C', 'h' , 'i' , 'n' , 'a', '\0' }`



字符串处理函数-1

- **puts(字符数组)**
 - 将一个字符串输出到终端
- **gets(字符数组)**
 - 从终端输入一个字符串到字符数组,并且得到一个函数值,该函数值是字符数组的起始地址.
- **strcat(字符数组1,字符数组2)**
 - 连接两个字符数组中的字符串,把字符串2接到字符串1的后面
- **strcpy(字符数组1,字符串2)**
 - 将字符串2复制到字符数组1中去.



字符串处理函数-2

- **strcmp(字符串1, 字符串2)**
 - 比较字符串1和字符串2
- **strlen(字符数组)**
 - 测试字符串长度
- **strlwr(字符串)**
 - 将字符串中的大写字母换成小写字母
- **strupr(字符串)**
 - 将字符串中的小写字母换成大写字母

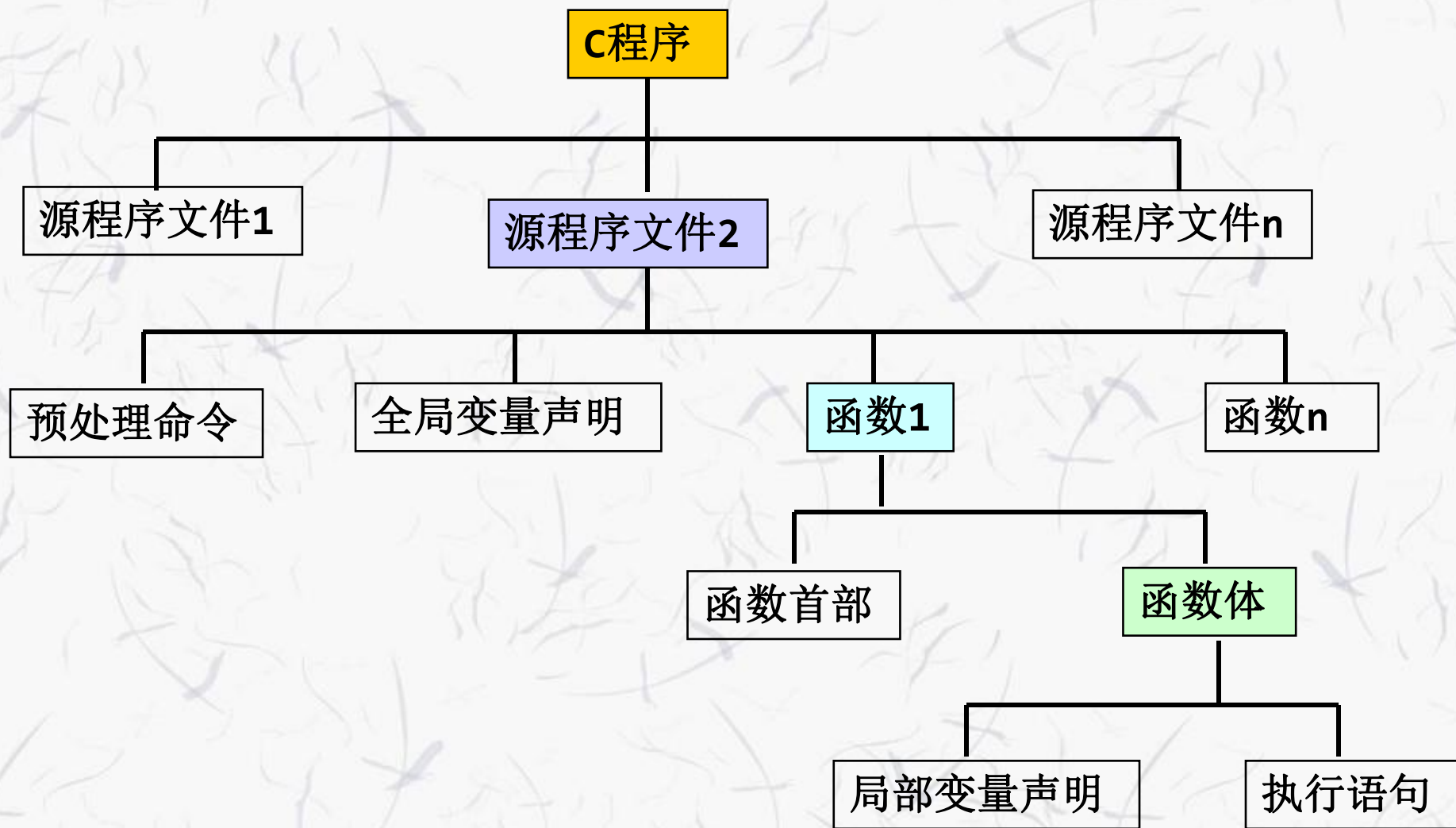


主要内容

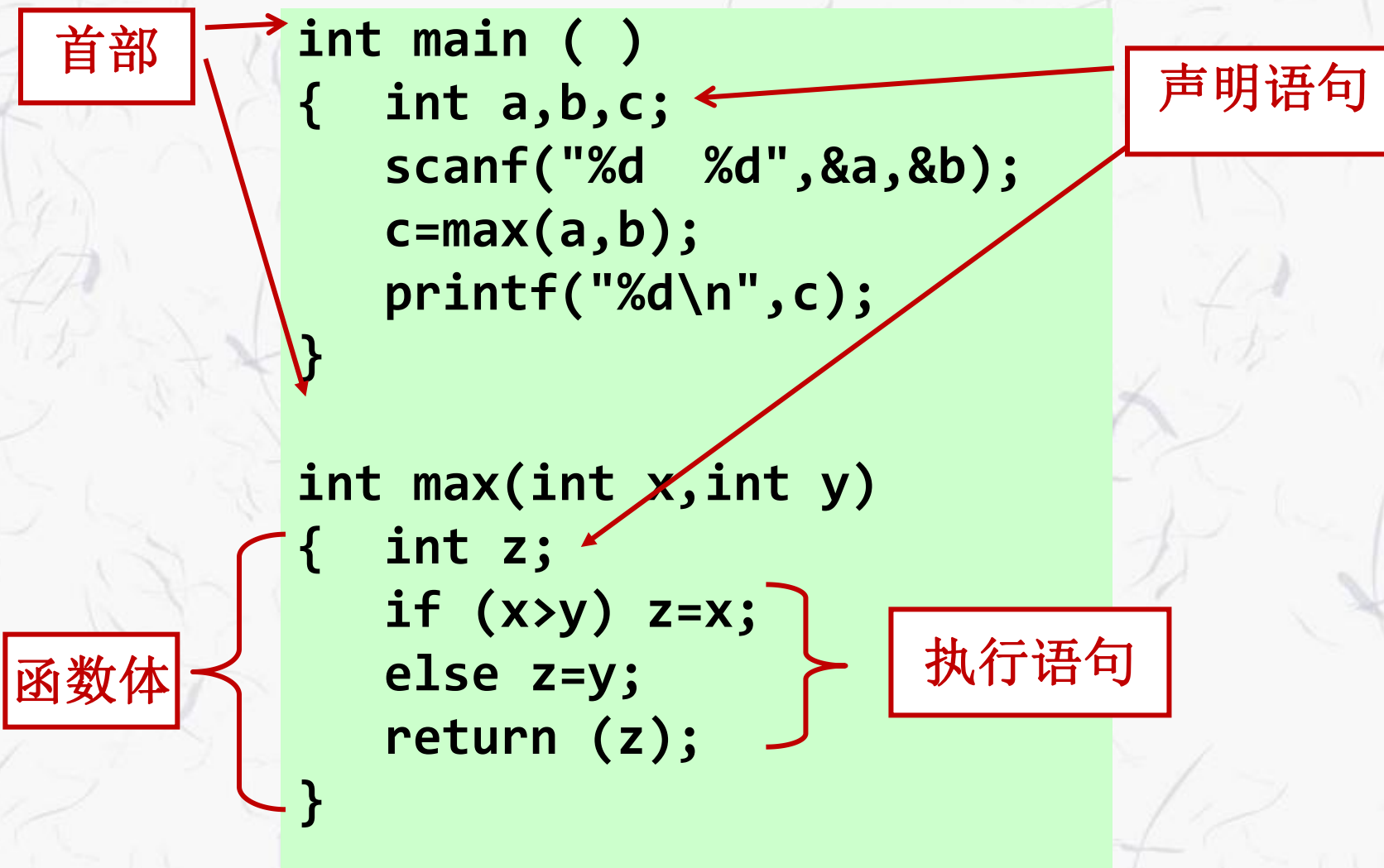
- C语言简介
- 数据类型与声明
- 运算符与表达式
- 数组
- 函数
- 程序流程
- 指针
- 结构体



C程序



函数



规定

- 函数分为**首部**和**函数体**。
- 在首部中定义函数，函数名后面必须是一个圆括号，括号内写函数参数，有些函数没有参数。
- 函数体写在花括号内，函数体一般包括数据**声明部分**和**执行语句**部分。
- 一条语句可以分写在几行上，一行也可以写几条语句。
- 每条语句后面必须有一个分号。
- `/*.....*/`是注释部分，是不执行的部分。



函数的调用

- 形式参数：有参函数在定义时，函数名后面括弧中的变量名称为“**形式参数**”，简称**形参**；
- 实际参数：在调用函数时，函数名后面括弧中的表达式称为“**实际参数**”，简称**实参**。
- **形参只能是变量**。在被定义的函数中，必须指定形参的类型。
- **实参可以是常量、变量或表达式**。
- 实参与形参的**个数应一样，类型应一致**（字符型和整型可以互相通用）。

```
#include <stdio.h>
double area(double r){
    double s;
    s = 3.1415926 * r * r;
    return s;
}
int main(){
    double bj, mj;
    scanf("%lf", &bj);
    mj = area(bj);
    printf("%lf\n", mj);
    return 0;
}
```



形式参数和实际参数

- 实参对形参的数据传递是**单向传递**，只由实参传递给形参。形参变量在未出现函数调用时，并不占用内存中的存储单元。在调用结束后，形参占用的内存单元被释放，实参单元仍保留并维持原值。因此，在执行一个被调用函数时，形参的值如果发生改变，并不会改变主调函数的实参的值。
- 在调用函数时，将实参的值赋给形参，如果实参是数组名，则传递的是数组首地址而不是变量的值。



函数的返回值

- 函数的返回值是通过函数中的**return**语句获得的，**return**语句的一般形式是：
 - **return** 表达式;
 - 表达式可以用括号括起来 **return (z)**
- 函数值的类型
 - **int max(float x,float y);**
 - **char letter(char c1,char c2);**
 - **double min(int x,int y);**
 - 凡不加类型说明的函数，一律自动按整型处理
- 如果没有**return**语句，只是不带回有用的值。
- 明确表示不带回值，可以定义**void**类型。



函数调用的方式

- 按位置可分为：
 - 函数语句
 - 不要求函数带返回值，只要求函数完成一定的操作
 - 函数表达式
 - 函数出现在一个表达式中
 - `c=2*max(a,b);`
 - 函数参数
 - 函数作为一个函数的实参
 - `m=max(a,max(b,c));`



被调用的函数必须具备的条件

- 被调用的函数必须是已经存在的函数
- 若调用库函数，应该在文件开头用**#include**命令将所需的信息包含到本文件中来。如：**#include <stdio.h>** 或 **#include "math.h"**
- 若使用自定义函数，该函数与主调函数一般应该在同一个文件中。
- 一般应该在文件开头或主调函数中对被调函数的类型进行说明。但是有两个例外：
 - 整型或字符型可以不说明
 - 被调函数写在主调函数之前可以不说明



求两个数的最大公约数和最小公倍数

解题方法：

- 最大公约数：
用大数整除小数，得到余数1；
再用小数整除余数1，得到余数2；
再用余数1整除余数2，
直到余数为0，该除数就是最大公约数。
- 最小公倍数
两数相乘再除以最大公约数。



求两个数的最大公约数程序

```
#include "stdio.h"
int gcd(int x,int y)
{
    int z;
    while (y){
        z=x%y;
        x=y;
        y=z;
        printf("%d,%d\n",x,y);
    }
    return(x);
}
//被调函数写在主调函数之前可以不说明
```

- $\text{gcd}(x,y)=\text{gcd}(x\%y,y)$
- $\text{gcd}(15,12)=\text{gcd}(12,3)=3$
- $\text{gcd}(45,6)=\text{gcd}(6,3)=3$

```
int main()
{
    int a,b,c;
    scanf("%d,%d",&a,&b);
    c=gcd(a,b);
    printf("%d\n",c);
    return 0 ;
}
```



求两个数的最小公倍数程序

```
#include <stdio.h>
int gcd(int x,int y)
{
    int z;
    while (y){
        z=x%y;
        x=y;
        y=z;
    }
    return(x);
}
```

/*被调函数写在主调函数之前可以不说明 */

```
int main()
{
    int a,b,c;
    scanf("%d,%d",&a,&b);
    c=a*b/gcd(a,b);
    printf("%d\n",c);
    return 0 ;
}
```



数组作为函数参数

- 数组元素可以作为函数实参
 - `large(a[i],b[i]);`
- 数组名可作函数参数
 - `average(score);`应在主调函数和被调用函数分别定义数组，实参数组与形参数组类型应一致，C编译只是将实参数组的首地址传给形参数组。
 - 形参数组可以不指定大小。
 - 用数组名做函数实参时，不是把数组元素的值传递给形参，而把**实参的起始地址**传递给**形参数组**，这样两个数组就共同占用一段内存单元。



例1-数组参数

```
# include <stdio.h>
void func(int b[]){
    int j;
    for(j=0;j<4;j++)
        b[j]=j;
}
int main(){
    int a[4],i;
    func(a);
    for(i=0;i<4;i++)
        printf("%d ",a[i]);
    return 0;
}
```

E:\wz2\course\compt2020s\cpp\ch12\shuzhu1.exe

0 1 2 3

Process exited after 0.1046 seconds
请按任意键继续. . .



例2-求10个数的平均值

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

```
float average(float *array){  
    int i;  
    float aver,sum=0;  
    for(i=0;i<10;i++)  
        sum=sum+array[i];  
    aver=sum/10;  
    return(aver);  
}
```

```
int main(){  
    float score[10],ave;int i;  
    printf("Input 10 scores:\n");  
    for(i=0;i<10;i++)  
        scanf("%f",&score[i]);  
    ave=average(score);  
    printf("Average score is %5.2f",ave);  
    return 0;  
}
```



主要内容

- C语言简介
- 数据类型与声明
- 运算符与表达式
- 数组
- 函数
- 程序流程
- 指针
- 结构体



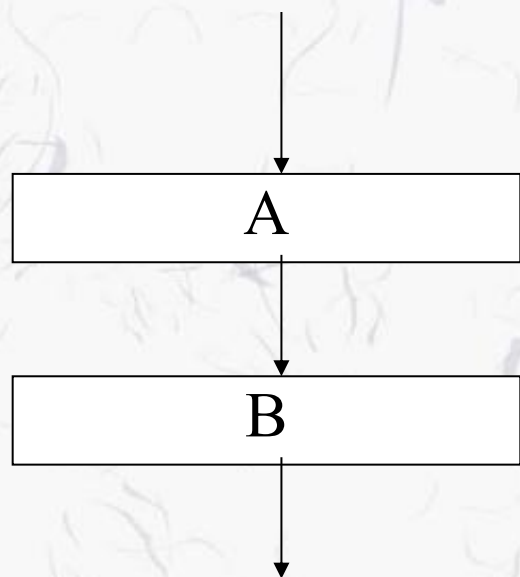
算法的基本结构

- 顺序结构
- 选择结构
- 循环结构



顺序结构

- 顺序型：几个连续的加工步骤依次排列构成



格式输出函数是 `printf`
格式输入函数是 `scanf`

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



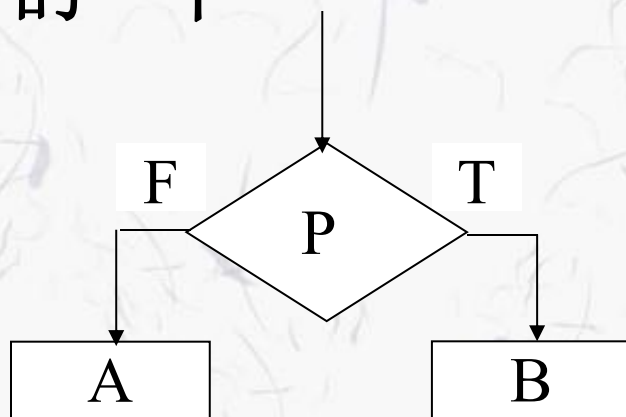
算法的基本结构

- 顺序结构
- 选择结构
- 循环结构



选择结构

- 选择型：由某个逻辑判断式的取值决定选择两个加工中的一个



- 在C语言中,选择结构可以用**if**语句、条件运算符、**switch**语句实现。

if语句

- if 语句的三种形式:
 - ① if(表达式) 语句
 - if (x>y) printf(“%d”,x);
 - ② if(表达式) 语句1 else 语句2
 - ③ if(表达式 1) 语句1
else if (表达式2) 语句2
...
else if (表达式m) 语句m
else 语句n



关系表达式

- 关系运算实际是比较运算。
- C语言提供了6种关系运算符：

①	<	小于	}	优先级相同(高)
②	<=	小于或等于		
③	>	大于		
④	>=	大于或等于		
⑤	==	等于	}	优先级相同(低)
⑥	!=	不等于		

- 关系运算符的优先级别低于算术运算符
- 关系运算符的优先级别高于赋值运算符
- 用关系运算符将两个表达式连接起来的式子称为关系表达式，它的值是一个逻辑值，以1代表真，0代表假。



逻辑表达式

- C语言提供三种逻辑运算符：

① && 逻辑与

② || 逻辑或

③ ! 逻辑非

- 用逻辑运算符将关系表达式或逻辑量连接起来就是逻辑表达式。

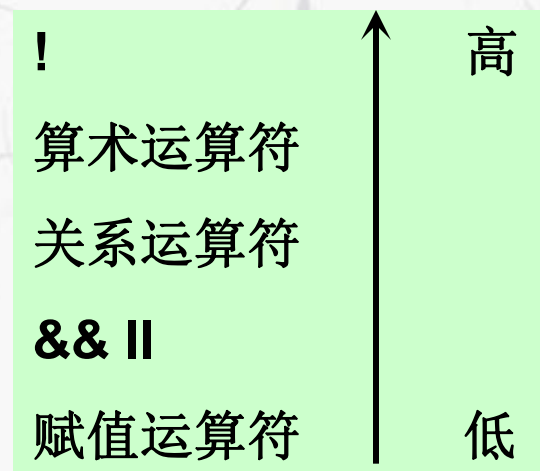
- 运算优先级别：

① ! → && → ||

② && || 低于关系运算符，!高于算术运算符

`a > b && x > y`

`!a || x > y`



if语句说明

- **if** 后面的表达式的值按逻辑值处理。
- **else** 必须与 **if** 配对使用。
- 当 **if** 或 **else** 后面需要执行若干条语句时，应用花括号将它们括起来组成复合语句。

```
int main ( )
{   int a=10,b=0;
    if (a<=12)
    {   a=a+1;b=b+1;}
    else {a=a+4;b=b+4;}
    printf("%d;%d\n",a,b);
    return 0;
}
```



switch语句

- 是多分支选择语句，实现多分支选择结构

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



switch语句说明

- 当表达式的值与某一个**case**后面的常量相等时，执行该**case**后面的语句。然后执行下一个**case**后面的语句，若要跳出**switch**结构，应使用**break**语句。
- 若所有的**case**后面的常量都不和表达式的值相匹配，就执行**default**后面的语句。
- 每一个**case**的常量表达式必须互不相同。
- 常量可以是**整数**或**字符**



switch例子

```
#include <stdio.h>
int main ( )
{   int x=1,y=0,a=0,b=0;
    switch(x)
    {   case 1:
        switch(y)
        {   case 0:a++;break;
            case 1:b++;break; }
        case 2:a++;b++;break;
    }
    printf("a=%d,b=%d\n",a,b);
    return 0;
}
```

运行结果:
a=2,b=1



条件运算符

- 条件运算符的一般形式为：
 - 表达式 1? 表达式2: 表达式3
- $\text{max}=(a>b)?a:b$ 等价于
 - `if (a>b) max=a;`
 - `else max=b;`
- 说明：
 - ① 条件运算符的优先级别比关系运算符和算术运算符都低， $a>b?a:b+1$ 相当于 $a>b?a:(b+1)$
 - ② 条件运算符的结合方向为 “自右至左”， $a>b?a:c>d?c:d$ ，相当于 $a>b?a:(c>d?c:d)$
 - ③ 表达式2和表达式3不仅可以是数值表达式,还可以是赋值表达式或函数表达式



算法的基本结构

- 顺序结构
- 选择结构
- 循环结构



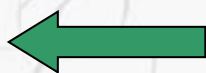
循环结构

- 主要的实现语句有

- **while** 语句

- **do-While** 语句

- **for**语句

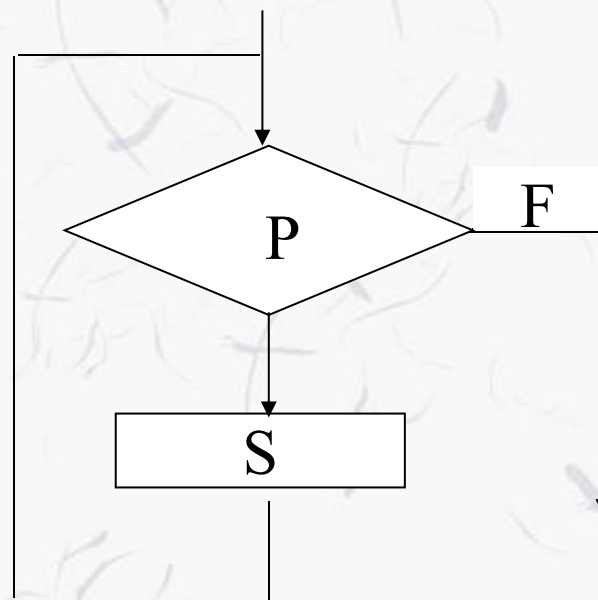


while语句

- 在循环控制条件成立时，重复执行特定的加工

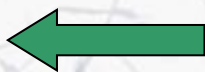
- **while**（表达式）语句

```
#include <stdio.h>
int main ( )
{   int i,sum=0;
    i=1;
    while(i<=100)
    {   sum=sum+i;
        i++;
    }
    printf(“%d\n”,sum);
    return 0;
}
```



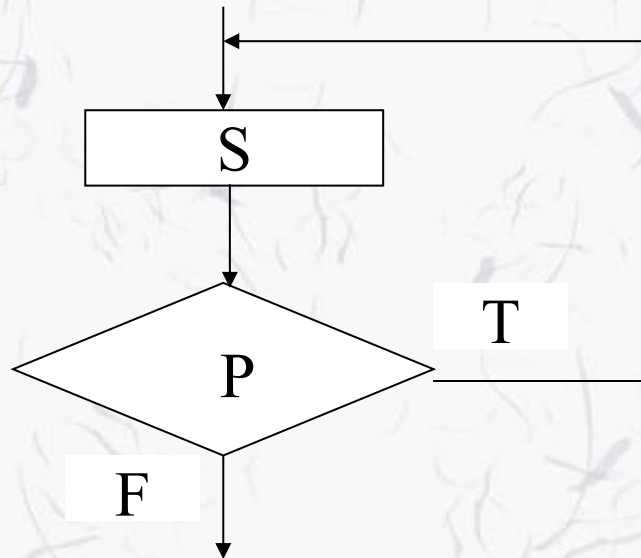
循环结构

- 主要的实现语句有
 - **while** 语句
 - **do-While** 语句
 - **for**语句



do-while语句

- 后判定型循环(**do_while**): 先执行循环体, 然后判断循环条件是否成立, 重复执行某些特定的加工, 直至控制条件不成立。



do-while语句和while语句的区别

```
int main ( )
{   int a;
    a=4;
    do printf("abc\n");
    while(a!=4);
    /*while(a!=4)
    printf("abc\n");*/
    return 0;
}
```



break语句和continue语句

- **break**语句的作用是跳出循环体，转去执行下一条语句；
- **continue**语句的作用是结束本次循环，转向判断循环条件是否否为‘真’。



循环结构

- 主要的实现语句有

- **while** 语句

- **do-While** 语句

- **for**语句



for语句

- for (表达式1;表达式2;表达式3) 语句
- 求100~200之间的能够被3整除的数

```
int main ( )
{
    int n;
    for (n=100;n<=200;n++)
    {
        if(n%3!=0) continue;
        printf("%d,",n);
    }
    printf("\n");
    return 0;
}
```



主要内容

- C语言简介
- 数据类型与声明
- 运算符与表达式
- 数组
- 函数
- 程序流程
- 指针
- 结构体



指针

- 是C语言的一个重要概念
 - ① 可以有效地表示复杂的数据结构
 - ② 能动态分配内存
 - ③ 能方便地使用字符串
 - ④ 能有效而方便地使用数组
 - ⑤ 在调用函数时能得到多于一个的值
 - ⑥ 能直接处理内存地址



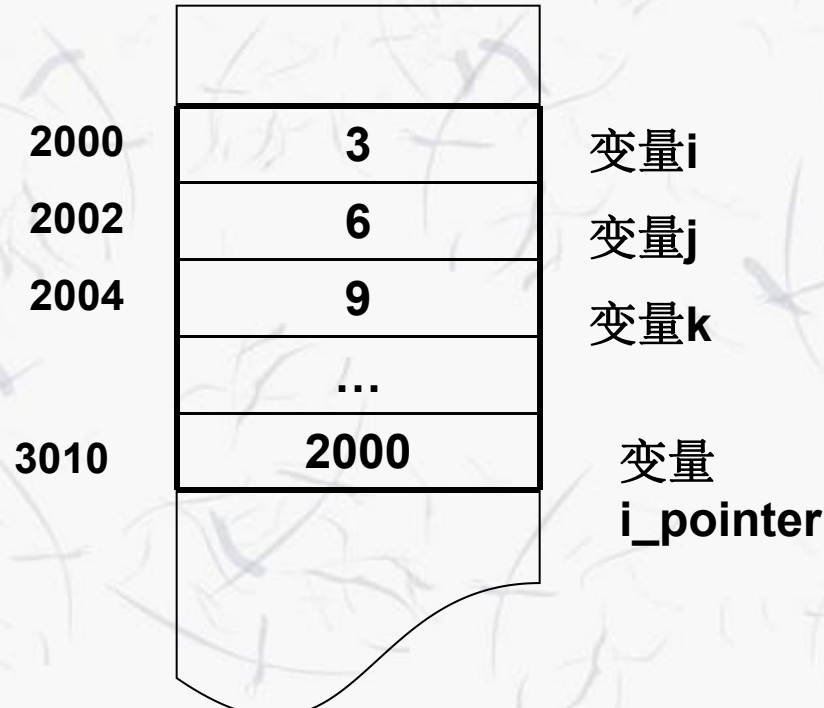
指针

- 地址和指针
- 变量的指针和指向变量的指针变量
- 数组与指针
- 字符串与指针
- 指向指针的指针
- 指针变量的空值



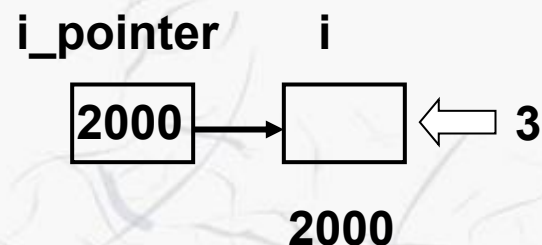
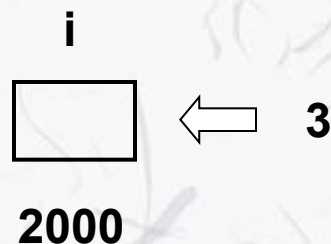
地址的概念

- 程序定义了一个变量，在编译时就给这个变量分配内存单元，内存区的每一个字节有一个编号，这就是“地址”
- 在程序中通过变量名来对内存单元进行存取操作
- 程序编译以后已经将变量名转换为变量的地址。
- `scanf("%d",&i);`
- `printf("%d",i);`
- `i_pointer=&i;`



地址和指针

- 一个变量的地址称为该变量的**指针**。通过变量的地址能找到该变量在内存中的存储单元，从而得到它的值。
- 变量*i*访问方式：
 - 直接访问：按变量地址存取变量值的方式。
 - ◆ `i=3; printf("%d",i);`
 - 间接访问：`*`表示指向，定义 `i_pointer`, `* i_pointer`
 - ◆ `* i_pointer=3; scanf("%d",&i);`



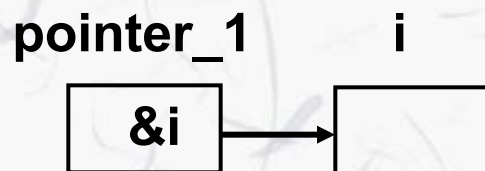
指针

- 地址和指针
- 变量的**指针**和指向变量的**指针变量**
- 数组与指针
- 字符串与指针
- 指向指针的指针
- 指针变量的空值



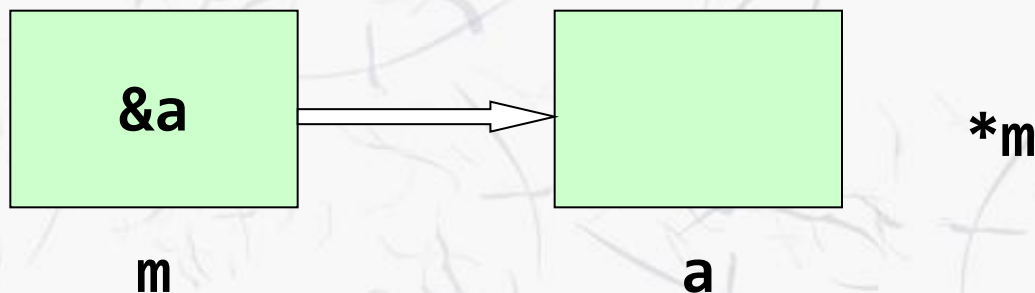
指向变量的指针变量

- 存放地址的变量叫做**指针变量**。
 - 在一个指针变量中只能存放同一类型变量的地址。指针变量必须先定义，指定它的类型，才能使用。
 - **基类型 *指针变量名**；例如：`int *a,*b`;
- 指向某数据的指针变量存放的是该数据的首地址，也就是说，指向的是该数据的首字节。
 - `int` 占2个字节，`char` 占1个字节，`float` 占4个字，...
- 使一个指针变量指向另一个变量
 - 指针变量名=`&`变量名；`i_pointer=&i`;
`pointer1=&i; pointer2=&j`;



指针变量的引用-1

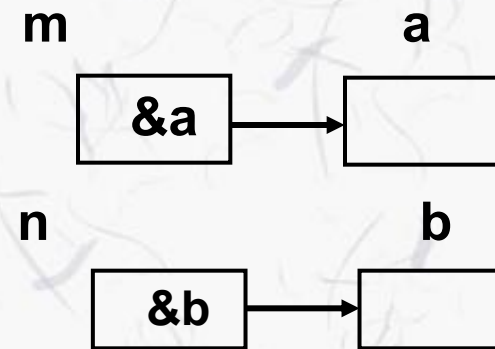
- 在表达式中，可以使用以下两个运算符
 - **&** 取地址运算符，如： $m = \&a$ 将a的地址送入m
 - ***** 指针运算符（间接访问运算符）
 - **&**与 $*$ 、 $++$ 、 $--$ 运算优先级别相同，但结合方向自右向左
 - ◆ 如： $*m$ 就是指针变量m所指向的变量，也就是a。
- 因此，有了 $m = \&a$ 后，就有了 $*m = a$ 。
- ◆ $\&*m$ 与 $\&a$ 相同， $*\&a$ 与a等价
- ◆ $(*m)++$ 相当于 $a++$ ， $*m++$ 相当于 $*(m++)$



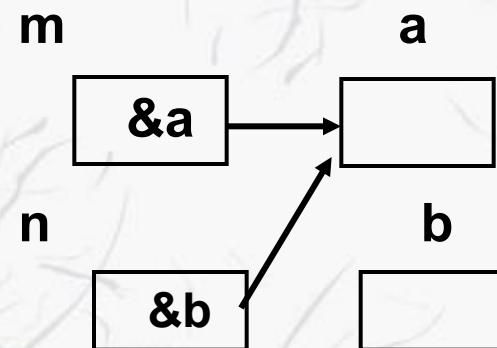
Example-1

```
int main ()
{  int a=10,b=20;
   int *m,*n;
   m=&a;n=&b;
   printf("%d,%d\n",a,b);
   printf("%d,%d\n",*m,*n);
   return 0;
}
```

运行结果:
10,20
10,20



● `n=&*m;`



例如2

```
int main ( )
{   int a,b;
    int *p1,*p2;
    a=100;b=10;
    p1=&a;p2=&b;
    printf("%d,%d\n",a,b);
    *p1=*p1+2;
    (*p2)++;
    printf("%d,%d\n",*p1,*p2);
    printf("%d,%d\n",a,b);
    p1=p2;
    printf("%d,%d\n",*p1,*p2);
    printf("%d,%d\n",a,b);
    return 0;
}
```

运行结果:

100,10

102,11

102,11

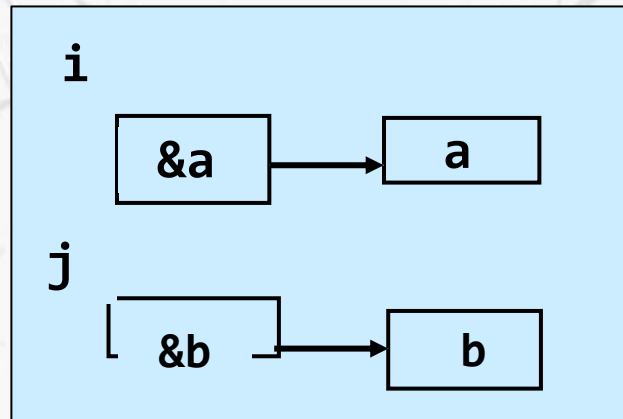
11,11

102,11



交换两个数据

```
int main()
{
    int a ,b;
    int *i,*j;
    i=&a;j=&b;
    scanf("%d,%d",&a,&b);
    swap(i,j);
    printf("%d,%d\n",a,b);
    return 0;
}
```



```
swap(int *p1, int *p2)
{
    int temp;
    temp=*p1;
    *p1=*p2;
    *p2=temp;
}
```

```
swap(int x,
int y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

x

y



如何使函数中改变了的变量被main函数使用？

- 不能采用把要改变值的变量做参数的办法
- 而应该用指针变量作为函数参数
- 通过调用函数使变量的值发生变化，在主调函数中使用这些改变了的值。



指针

- 地址和指针
- 变量的指针和指向变量的指针变量
- 数组与指针
- 字符串与指针
- 指向指针的指针
- 指针变量的空值



数组的指针和指向数组的指针变量

- 数组元素的指针是指数组元素的地址。
- `int a[10];` (定义a为包含10个整型数据的数组)
- `int *p;` (定义p为指向整型变量的指针变量)
- `p=&a[0];` (赋值)
- `p=a;` (数组名代表数组第一个元素的首地址)
- `int *p=&a[0]` (定义的同时赋值)
- `int *p=a;`



通过指针引用数组元素-1

- 可以通过指针为数组元素赋值。

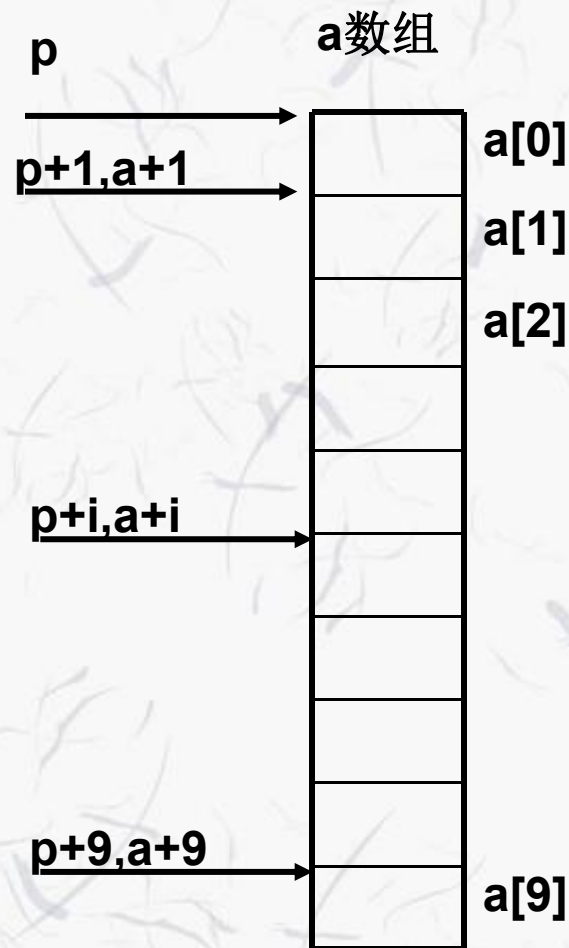
```
int a[10], *p;  
p=&a[0];  
*p=5;
```

- 指针变量指向数组中的元素。

```
int a[10];  
int *p=&a[0] ;  
p=p+1
```

此时， p 指向数组 a 数组的第二个元素。

```
*(p+1)=a[1]
```



通过指针引用数组元素-2

- 指向数组的指针变量也可以带下标。
 - 例如: `int a[10]; int *p=&a[0] ;`
可以使用`a[i]`、`m[i]`、`*(a+i)`或`*(m+i)`引用数组中的第`i+1`个元素。
- 移动指针可以使用`p++`、`p--`，但注意指针不能超出数组元素的范围。
 - `++`和`*`是同优先级，从右至左的结合方向。 例如`* m++`和`*(m++)`
等价
- 指针不允许进行乘除运算。



用数组名做函数参数

- 实参数组名代表该数组元素的首地址，因此，形参是用来接收从实参传递过来的数组元素的首地址，因此形参是一个指针变量。
 - `float average(float *array)`与`float average(float array[])`等价
- `*array` 就是`score[0]`的值，`array+1`指向`score[1]`，`array+2`指向`score[2]`，`*(array+1)`，`*(array+2)`就是`score[1]`，`score[2]`的数值

实参类型	变量名	数组名
要求形参的类型	变量名	数组名或指针变量
传递的信息	变量的值	实参数组首元素的地址
通过函数调用能否 改变实参的值	不能	能

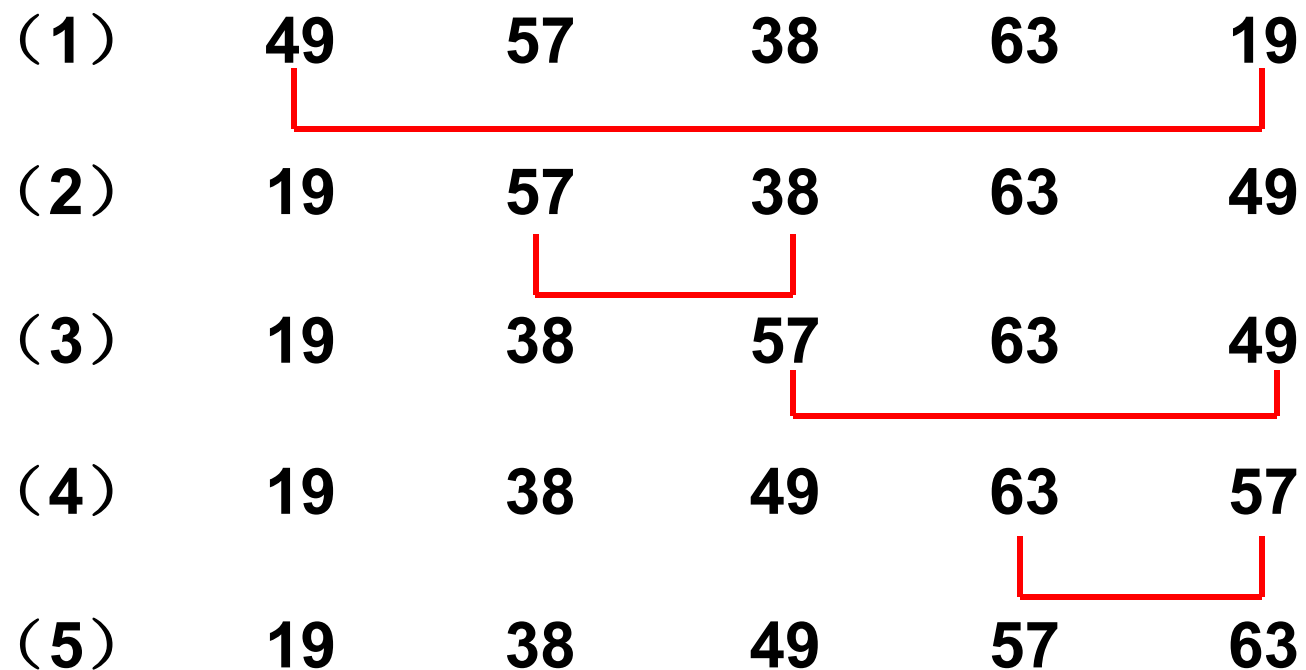


通过函数调用得到n个要改变的值

- ① 在主调函数中设n个变量，用n个指针指向它们
- ② 编写被调用函数，其形参为n个指针变量，与主调函数的n个指针变量的基类型相同
- ③ 在主调函数中将n个指针变量做实参，将它们的值（地址）传给所调用的函数的n个形参指针变量，这样形参指针变量也指向这n个变量
- ④ 通过形参变量的指向，改变该n个变量的值
- ⑤ 主调函数中就可以使用这些改变了值的变量



Example-直接选择排序



用选择法排序

```
int main()
{
    int *p,i,a[5];
    p=a;
    for(i=0;i<5;i++)
        scanf("%d",p++);
    p=a;
    sort(p,5);
    for(p=a,i=0;i<5;i++)
    { printf("%d",*p);
      p++;}
    return 0;
}
```

```
sort(int x[ ],int n)
/* sort( int *x,int n) */
{ int i,j,k,t;
  for (i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++)
        if(x[j]<x[k]) k=j;
    if(k!=i)
    { t=x[i];
      x[i]=x[k];
      x[k]=t;
    }
  }
}
```



实参数组与形参的对应关系

- 可以有以下四种情况：

- 形参和实参都用数组名

```
int main()                f(int x[ ],int n)
{  int a[10];            {  ...
    ...
    f(a,10);              }
    ...}
```

- 实参用数组名，形参用指针变量 `f(int *x,int n)`

- 实参和形参都用指针变量

- 实参为指针变量，形参为数组名



指针

- 地址和指针
- 变量的指针和指向变量的指针变量
- 数组与指针
- 字符串与指针
- 指向指针的指针
- 指针变量的空值



字符串的表示形式

- 用字符数组 `char string[]` 存放一个字符串

- `char string[]="I am fine";`

- 用字符指针 `char *p` 指向一个字符串 `char string[]`

- `char string[]="I am fine";`

- `char *p=string;`

- `p` 是一个字符指针变量



```
#include <stdio.h>
int main()
{
    char *string;
    string="I am fine";
    printf("%s\n",string);
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    char *string;
    string=(char *)"I am fine";
    printf("%s\n",string);
    return 0;
}
```

[Warning] deprecated conversion from string constant to 'char*' [-Wwrite-strings]

E:\wz2\course\compt2020s\cpp\ch11\poin

I am fine

Process exited after 0.144
请按任意键继续. . .




```
#include <stdio.h>
int main()
{
    const char *string;
    string="I am fine";
    printf("%s\n",string);

    return 0;
}
```

```
E:\wz2\course\compt2020s\cpp\ch11\poin
I am fine
-----
Process exited after 0.144
请按任意键继续. . .
```

```
#include <stdio.h>
int main()
{
    char string[]="I am fine";
    char *p=string;
    printf("%s\n",p);
    printf("%s",string);
    return 0;
}
```

```
E:\wz2\course\compt2020s\cpp\ch11\pointer9.exe
I am fine
I am fine
-----
Process exited after 0.1958 seconds w
请按任意键继续. . .
```



指针

- 地址和指针
- 变量的指针和指向变量的指针变量
- 数组与指针
- 字符串与指针
- 指向指针的指针
- 指针变量的空值



指针数组

- 一个数组，其元素均为指针类型的数据，称为**指针数组**。
- 定义形式为：
 - 类型名 *数组名[数组长度]
 - `int *p[4];`
- 与`int (*p)[4]`不同，指向**一维数组的指针变量** 也称**行指针**。
- `p`是指向一个有4个元素的`int`数组的指针（相当于一个二维指针），如果执行`p+1`，那么它将移动4个`int`大小的地址。其本质为指针
- `int a[4];`
- 比较适合于指向若干字符串



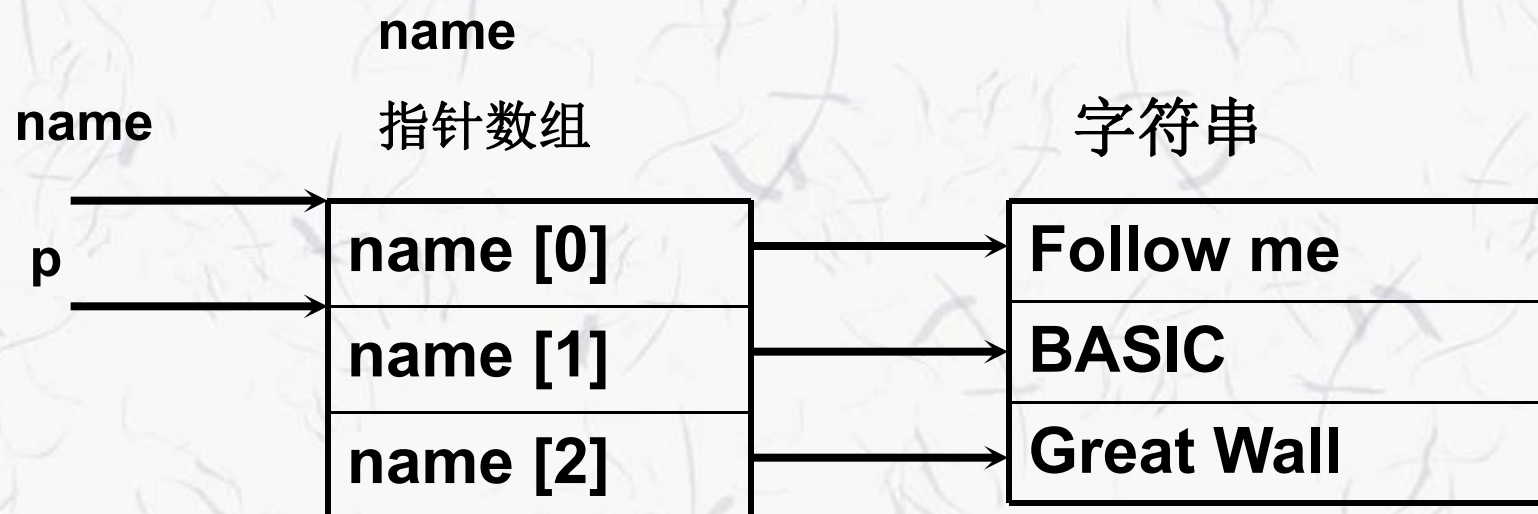
指针数组

- `const char *name[]={"Follow me", "BASIC", "Great Wall"};`



指向指针的指针

- 指向指针数据的指针变量，简称**指向指针的指针**。



- `char **p`



指向指针的指针

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

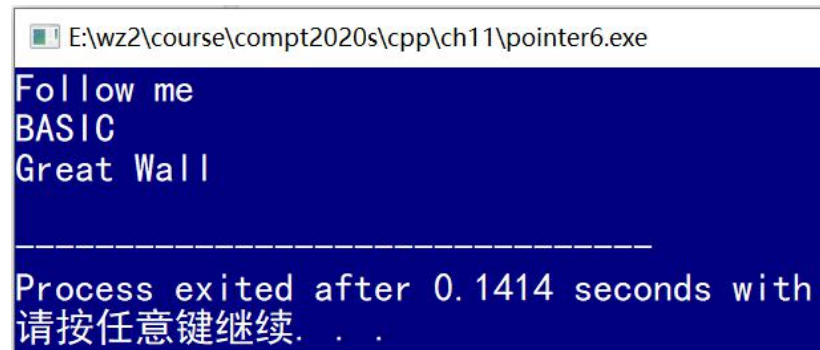
int  main( )
{
    char *name[]={(char *)"Follow me",(char *)"BASIC", (char *)"Great Wall"};
    char **p;
    int i;
    for (i=0;i<3;i++)
    {   p=name+i;
        printf("%s\n",*p);
    }
    return 0;
}
```

运行结果: **Follow me**
BASIC
Great Wall

- 指针变量也有地址，这个地址可以存放在另一个指针变量中。
- 例如定义一个指向字符型指针数据的指针变量：
`char **p;` 后，如果有
`char *q="abc"; p=&q;`
则变量p是指向指针变量q的指针

指向指针的指针

```
#include<stdio.h>
#include <stdlib.h>
int  main( )
{
    const char *name[]={"Follow me","BASIC","Great Wall"};
    char **p;
    int i;
    for (i=0;i<3;i++)
    {    p=(char**)name+i;
        printf("%s\n",*p);
    }
    return 0;
}
```



```
E:\wz2\course\compt2020s\cpp\ch11\pointer6.exe
Follow me
BASIC
Great Wall

-----
Process exited after 0.1414 seconds with
请按任意键继续. . .
```



指针

- 地址和指针
- 变量的指针和指向变量的指针变量
- 数组与指针
- 字符串与指针
- 指向指针的指针
- 指针变量的空值



空指针类型

- **ANSI** 新标准增加了**void**指针类型 (**void ***)，即可以定义一个指针变量，但不指定它是指向哪一种数据类型。
- **malloc ()** 函数
 - 其函数原型为**void * malloc(unsigned int size);**
 - 其作用是在内存的动态存储区域中分配一个长度为**size**的连续空间.如果未成功，返回空指针**NULL**.
- **free()**函数
 - 其函数原型为**void free(void *p);**
 - 其作用是释放由**p**指向的内存区.



指针变量可以有空值

- 即该指针变量不指向任何变量
- `p=NULL;`
 - `# define NULL 0`
 - `p=NULL;`
 - `/* 表示p不指向任何有用单元。*/`
 - 任何指针都可以与NULL作比较
 - `if(p==NULL)`

```
char *sa;  
sa=(char *)malloc(sizeof(char));  
if(sa==NULL)  
{  
    printf("No space for sa");  
    return 0;  
}  
/*检验内存是否分配成功*/
```



主要内容

- C语言简介
- 数据类型与声明
- 运算符与表达式
- 数组
- 函数
- 程序流程
- 指针
- 结构体



结构体

- 什么时候需要结构体类型？
- 结构体类型变量的定义
- 结构体变量的引用
- 结构体变量的初始化
- 结构体数组
- 指向结构体类型数据的指针
- 结构体与函数



什么时候需要结构体类型?


- 一类用户 自定义 的数据类型
- **struct**

num	name	sex	age	score	addr
10010	Li Fun	M	18	87.5	Beijing

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[ 30];
};
```



结构体

- 什么时候需要结构体类型？
- 结构体类型变量的定义 
- 结构体变量的引用
- 结构体变量的初始化
- 结构体数组
- 指向结构体类型数据的指针
- 结构体与函数



结构体类型变量的定义-1

① 先**声明**结构体类型再**定义**变量名

□ 声明一个结构体类型，并指定其中各成员的名字和它们的类型。
。声明结构体类型的形式：

struct 结构体名 {成员表列};

□ 再定义变量名

```
struct point {  
    float x;  
    float y;  
};  
struct point point1;  
struct point point2;
```

□ 系统会为结构体
类型变量**分配内
存空间**



结构体类型变量的定义-1

①先声明结构体类型再定义变量名

```
struct abc
{
    int a;
    char b;
    float c;
    char d[30];
};
struct abc xx1,xx2,xx3;
```

定义了三个变量：
xx1、xx2和xx3，
它们都具有
struct abc的
类型结构。

结构体类型变量的定义-2

② 在声明类型的**同时**定义变量，一般形式为：

```
struct 结构类型名  
{ 成员表列  
}变量名表列;
```

```
struct abc  
{  
    int a;  
    char b;  
    float c;  
    char d[30];  
} xx1,xx2,xx3;
```

结果和前面一样

```
struct city{  
    float x, y;  
    int population;  
} city1, city2;
```



结构体类型变量的定义-3

③也可以直接定义结构体类型变量

struct {
 成员表列
}变量名表列;

```
struct  
{    int  a;  
    char b;  
    float c;  
    char d[30];  
} xx1,xx2,xx3;  
结果和前面一样
```

- 定义了三个变量：xx1、xx2和xx3，它们都具有**struct**

```
{    int  a;  
    char b;  
    float c;  
    char d[30];  
} 的类型结构。
```



结构体类型变量占用的内存

- 只有在定义变量后，才为该变量分配内存单元。结构体类型变量所占的内存长度大致等于每个成员长度之和。

```
#include <stdio.h>
int main ( )
{
    struct date
    {
        int year,mouth,day;
    }today;
    printf("%llu\n", sizeof(today));
    return 0;
}
```

- 运行结果：12 (DEV C++)



结构体

- 什么时候需要结构体类型？
- 结构体类型变量的定义
- 结构体变量的引用
- 结构体变量的初始化
- 结构体数组
- 指向结构体类型数据的指针
- 结构体与函数



结构体变量的引用

- 不能对结构体变量整体进行输入输出，只能对结构体变量中的成员分别进行输入和输出。
- 引用结构体变量中各个成员的方式为：
 - 结构体变量名.成员名
- 可以对变量的成员赋值
 - ◆ `student1.num=99082;`
- 如果成员又是结构体类型，则要用多个成员运算符，找到最低一级的成员。
 - ◆ `student1.birthday.month`



结构体类型变量的使用

- 结构体类型变量的成员，可以单独使用，作用与地位相当于普通变量，成员也可以是一个结构体类型变量。

```
struct date
{   int month;   int day;   int year;};

struct student
{   int num;
    char name[20];
    char sex;
    struct date birthday;
    /*birthday 是struct date 类型*/
} student1, student2;
```



结构体

- 什么时候需要结构体类型？
- 结构体类型变量的定义
- 结构体变量的引用
- 结构体变量的初始化
- 结构体数组
- 指向结构体类型数据的指针
- 结构体与函数



结构体类型变量的初始化

- 可以在定义时指定初始值

```
#include "stdio.h"
int main ( )
{ struct student
  { int num;
    char name[20];
    char sex;
  } a={89031, "Li Lin", 'M'};
  printf(" No.: %d\n name:%s\n sex:%c\n",a.num,a.name,a.sex);
  return 0;
}
```

E:\wz2\course\compt2020s\cpp\ch13\struct1.e

```
No. : 89031
name:Li Lin
sex:M

-----
Process exited after 0.2579 seconds
请按任意键继续. . .
```




同类型结构体变量之间的赋值运算

- 结构体变量之间进行赋值时，系统将按成员一一对应赋值

```
struct date
{   int year, month, day;};
struct student
{   char num[8], name[20], sex;
    struct date  birthday;
    float score;
}a={"9606011", "Li ming", 'M', {1977, 12, 9}, 83}, c;
c = a;
```



结构体

- 什么时候需要结构体类型？
- 结构体类型变量的定义
- 结构体变量的引用
- 结构体变量的初始化
- 结构体数组 
- 指向结构体类型数据的指针
- 结构体与函数



结构体数组

- 和定义结构体变量相似，只需要说明其为数组即可

① struct student

```
{    int num;  
    char name[20];  
    char sex;  
} ;
```

```
struct student stu[3];
```

/*先声明类型，再定义结构体数组*/

② struct student

```
{    int num;  
    char name[20];  
    char sex;  
} stu[3];
```

/*直接定义结构体数组*/



结构体数组初始化

```
struct student
{
    int num;
    char name[20];
    char sex;
} stu3[3]={89031, "Li Lin", 'M'}, {89032, "Liu Ying",
    'M'}, {89036, "Wang Min", 'F'};
```

```
struct student
{
    int num;
    char name[20];
    char sex;};
struct student stu3[3]={ {...}, {...}, {...}}
```



结构体类型变量中分量的访问

- 结构体类型变量的值由其各个分量构成
- 对分量的访问一般通过“**变量名.分量名**”完成
- 结构体赋值及访问的例子：

```
float dx, dy;  
struct point {  
    float x, y;  
} p1, p2, points[2];
```

```
p1.x = p1.y = 3.5f;  
p2.x = p2.y = 1.5f;  
dx = p1.x - p2.x;  
dy = p1.y - p2.y;
```

结构体变量本身可以作为一个整体来使用

```
points[0] = p1;  
points[1] = p2;
```



结构体类型中的分量

- 结构体类型中分量的类型可以是任何类型

- 基本数据类型的分量

```
struct point{  
    float x, y;  
};
```

- 其他类型的分量：结构体类型、数组类型

```
struct city{  
    struct point location;  
    int population;  
    char name[32];  
}city1;
```

`(city1.location).x`

```
struct city {  
    struct point{  
        float x, y;  
    }location;  
    int population;  
    char name[32];  
}city1;
```

~~```
struct city {
 char name[32];
 struct city city1;
}x;
```~~

- 分量的类型不能是未定义的结构体类型
- 分量的类型不能是正在定义的结构体类型





# 结构体

- 什么时候需要结构体类型？
- 结构体类型变量的定义
- 结构体变量的引用
- 结构体变量的初始化
- 结构体数组
- 指向结构体类型数据的指针
- 结构体与函数



# 指向结构体类型数据的指针

- 一个指向结构体变量的指针就是该变量占据的内存段的起始地址。可以设一个指针变量，用来指向一个结构类型变量。
- 指向结构类型变量的指针

```
struct student
{ int num;
 char name[20];
 char sex;
 float score;
};
struct student stu1;
struct student *p;
p=&stu1;
```

• 以下三种形式等价：

① 结构类型变量名.成员名

② (\*p).成员名

③ P->成员名

为了方便，可以把(\*p).num改用p->num来代替



# 指向结构体的指针

- 当指针指向一个结构体时，可用“**指针->分量**”或“**(\*指针).分量**”形式访问结构的分量，例如：

```
struct point {
 float x, y;
} point1, point2;
struct point * pc = &point1;
// pc->x等价于 (*pc).x, 等价于 point1.x
pc->x = 10;
pc->y = 20;
pc = &point2;
(*pc).x = 100;
(*pc).y = 200;
```

```
point1
(x = 10, y = 20)
point2
(x = 100, y = 200)
```



# 结构体变量的引用

- 结构变量名. 成员名 例如: `xx1.year`
- 指针结构变量名->成员名 例如: `p->year`

```
#include "stdio.h"
int main ()
{
 struct ex {
 int x;
 int y; };
 struct EXAMPLE
 { struct ex in;
 int a;
 int b;
 } e, *p;
 e.a=1; e.b=2;
 e.in.x=e.a*e.b;
 e.in.y=e.a+e.b;
 printf("%d, %d",e.in.x,e.in.y);
 return 0;
}
```

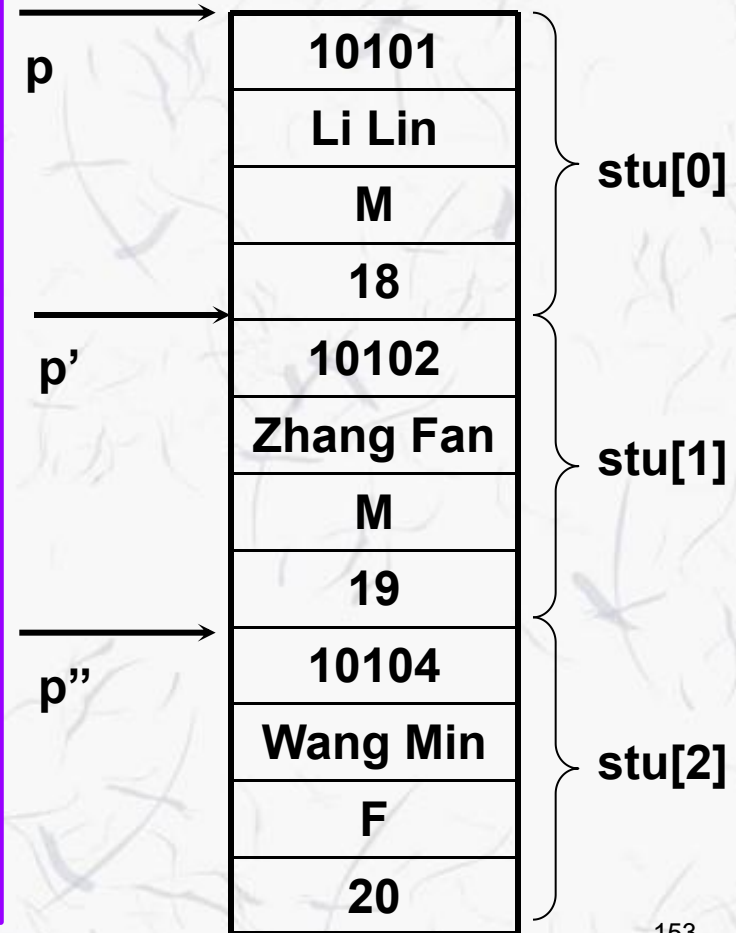
```
p->a=1;p->b=2;
p->in.x=p->a+p->b;
p->in.y=p->a*p->b;
printf("%d,%d",p->in.x,p->in.y);
```

运行结果: 2,3



# 指向结构体数组的指针示例

```
include "stdio.h"
struct student
{
 int num;
 char name[20];
 char sex;
 int age;
};
struct student stu[3]={{10101,"Li
 Lin",'M',18},{10102,"Zhang Fan",'M',19},
{10104,"Wang Min",'F',20}};
int main()
{
 struct student *p;
 printf(" No. Name Sex age\n");
 for (p=stu;p<stu+3;p++)
 printf ("%5d %10s %2c %4d\n",
 p-> num, p->name, p->sex, p->age);
}
```





# 结构体

- 什么时候需要结构体类型？
- 结构体类型变量的定义
- 结构体变量的引用
- 结构体变量的初始化
- 结构体数组
- 指向结构体类型数据的指针
- 结构体与函数



# 结构体与函数

- 用结构体类型变量的**成员**作参数
  - 如 `stu[1].num`
- 用**结构体类型变量**做实参数
- 用指向结构体类型变量（数组）的**指针**做实参，将结构体类型变量（或数组）的地址传给形参



# 用typedef声明类型

- 可以用typedef声明新的类型名来代替已有的类型名
- `typedef int INTEGER;`  
□ `int i,j;` 与 `INTEGER i,j;` 等价
- `typedef int DataType ;`
- `DataType *element ;` 与 `int *element;` 等价
- 习惯上把用typedef声明的类型名用大写字母表示



# ①返回结构体值的函数

- 函数mkpoint 返回一个POINT 结构类型的值，这个值可以赋给任何POINT 结构类型的变量。

```
POINT mkpoint(double x, double y) {
 POINT temp;
 temp.x = x;
 temp.y = y;
 return temp;
}
```

- POINT pt1,pt2;
- pt1 = mkpoint(3.825, 20.7);
- pt2 = mkpoint(pt1.x, 0.0);

```
struct point {
 double x,y;
};
typedef struct point
POINT;
```



## ②以结构体为参数的函数

- **/\*\*计算两点之间距离的函数\*\*/**

```
double distance(POINT p1,POINT p2){
 double x=p1.x-p2.x, y=p1.y-p2.y;
 return sqrt(x*x+y*y);
}
```

- **/\*\* 打印学生的学号和姓名的函数\*\*/**

```
void prtStudent(STUDENT stu) {
 printf("%d\n", stu.num);
 printf("%s\n", stu.name);
}
```

```
struct student
{ int num;
 char name[20];
 char sex;
 int age;
};
typedef student STUDENT;
```





### ③具有结构体参数并返回结构体值的函数

- 由参数值出发构造一个 **CIRCLE** 类型的结构值

```
CIRCLE mkcircle(POINT c, double r) {
 CIRCLE temp;
 temp.center = c;
 temp.radius = r;
 return temp;
}
```

- **CIRCLE** circ1,circ2,circ3;
- **POINT** pt1,pt2;
- circ1 = mkcircle(pt1, 5.254);
- circ2=mkcircle(circ1.center,11.7);
- circ3=mkcircle(mkpoint(2.05,3.7),3.245);

```
struct point {
 double x,y;
};
typedef struct point POINT;

struct circle {
 POINT center;
 double radius;
};
typedef struct circle CIRCLE;
```

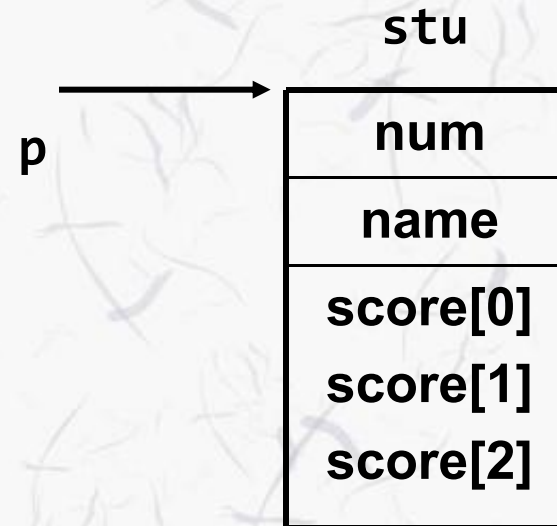


## ④用指向结构体变量的指针做参数

```
#define FORMAT "%d\n%s\n%f\n%f\n"
#include "stdio.h"
struct student
{ int num;
 char name[20];
 float score[3];
}stu={10101,"Li Lin",67.5,89,98};
void print(struct student *p);
```

```
int main()
{ print(&stu);
 return 0;
}
```

```
void print(struct student *p)
{ printf(FORMAT,p->num,p->name,p->score[0],p->score[1],p->score[2]);
 printf ("\n");
}
```



## ⑤用指向结构体数组的指针做参数

```
•/** 打印一组学生信息**/
STUDENT pstu[100], *p;
.../**假设pstu的元素都有值**/
for(p=pstu;p<pstu+100;p++)
 prtStudent(p);
```

```
void prtStudent(STUDENT pstu) {
 printf("%d\n", pstu->num);
 printf("%s\n", pstu->name);
}
```

```
struct student
{
 int num;
 char name[20];
 char sex;
 int age;
};
typedef struct student STUDENT;
```



# 结构体使用中的注意事项

- 结构体定义在头文件中(.h文件中)可以共享使用
- 结构体名不能相重、结构内分量名也不能相重
- 结构体占用内存字节数不完全等同于成员占用字节总数
- 只把真正相互紧密关联的成员定义在同一个结构体中
- 结构体分量可以是任何数据类型，但不能是正在定义的或未定义的类型
- 结构体也可用作数组的元素类型（结构体数组）

```
struct roof{
 float x, y;
 int p;
}roofs[100];
```

- 数组也可作为结构体的分量类型
- 使用结构体变量时首先要对成员进行初始化或赋值



# 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`

