第3章

主要语法及典型程序

3.1 C语言的语法重点

3.1.1 合法的标识符

合法的标识符由字母或数字或下画线组成，第一个字符必须是字母或下画线，而且不能是关键字，见主教材附录A。变量名、函数名、标签名等均是标识符，示例见表3-1。

表3-1 合法与非法的标识符举例对比

合法标识符举例V		非法标识符举例X	
例　子	解　释	例　子	解　释
A	大写与小写均可	**4a**	第一个字符是数字
a	同名大小写是两个不同变量	**a－1**	非法符号－
a4	第一个字符是字母就可以	**a＊2**	非法符号＊
IF	大写与小写不是同一个	**a.2**	非法符号.
_if	第一个字符是下画线就可以	**if**	关键字
_4	第一个字符是下画线就可以	**4_a**	第一个字符是数字

3.1.2 数据类型

为了高效表达、存储数据，设置不同的数据类型，主要有整型、实型、字符型。

1. 整型

整型又分为整型(int，4字节)、短整型(short，2字节)、无符号整型(unsigned)。常数默认是int类型。短整型只是占内存长度，可表示的数范围小一些。

```
01  int a;                  //最高位表示正负，数用补码表示，4字节
02  short b;                //最高位表示正负，数用补码表示，2字节
03  unsigned int a;         //最高位也是数据，没有负数，原码表示，4字节
04  unsigned a;             //与上一句等价
05  unsigned short b;       //最高位也是数据，没有负数，原码表示，只有2字节
```

2. 实型

实型又分为单精度类型(float)、双精度类型(double)。它们占用内存字节数不同：float 占 4 字节、double 占 8 字节，后者精度更高、表示的数据范围更大。

实型常数默认是 double 类型。不同类型混合运算先转换为 double 类型，将实型赋值为整型是简单取整。例如：

```
01  double d=1/2.;                    //d 将被赋值为 0.5,注意 2 后面有小数点
02  double d=1/2;                     //d 将被赋值为 0
03  int a=3.9;                        //a 将被赋值为 3
```

3. 字符型

字符型为 char，占 1 字节，按照 ASCII 码表示、存储，也可以按照整型读出或显示。字符常量用单引号表示。

```
01  char c='a', b='0';
02  printf("%c,%c,%d,%d",c,b,c,b);        //将输出 a,0,97,48
```

3.1.3 常量的表示

读者可以通过表 3-2 了解合法与非法常量的表示。

表 3-2 合法与非法的常量表示对比

类型	合法常量表示举例 V		非法常量表示举例 X	
	例子	解　　释	例子	解　　释
整数	0172	0 开头是八进制整数，即 122	**0192**	八进制不能出现 9
	0x5f	0x 开头是十六进制整数，即 95	**31a5**	十进制不能出现 a
	0b101	0b 开头是二进制整数，即 5	**0b121**	二进制只能有 0 和 1
实数	3.	就是 3.0，无歧义	**.**	只有小数点不行
	.5	就是 0.5，无歧义	**3.2e**	不能没有指数
	1e-5	双精度实数，即 0.00005	**2e0.5**	指数只能是整数
	.3e2	即 30	**.e2**	数值部分必须有数
	300e+5	即 30000000	**e2**	变成一个变量名了
字符	'a'	单引号	**'ab'**	多于 1 个字符不行
	'\x61'	十六进制的 ASCII 码表示	**'\181'**	八进制数字不能大于 7
	'\141'	八进制 ASCII 码表示		
	'\n'	转义字符：换行		

通过下面程序的输出可以看到整型常量的不同进制表示方法。

```
01  printf("%d",0b101);                    //输出显示:5
02  printf("%d",0101);                     //输出显示:65
03  printf("%d",0x101);                    //输出显示:257
```

注意掌握二进制与十进制、八进制、十六进制之间的人工转换。

3.1.4 运算符

(1) 区分赋值运算符(=)与关系运算符(==)。

通过表 3-3 的两个程序段中赋值运算符(=)与关系运算符(==)的使用对比,可以体会这两个运算符的区别。

表 3-3 赋值运算符与关系运算符的对比

关系运算符	误用赋值运算符(逻辑错)

int a=2; //赋值
if(a==2){d++;} //如果 a 等于 2 就执行
	if(a=2) {d++;}//逻辑错,括号中总是逻辑真
	//而没有判断 a 是否等于 2,只是将 2 赋值给 a,
	//括号中的结果总是 2,是非零,所以永远是真

(2) 复合赋值运算符有+=、-=、*=、/=、%=、&=、|=、^=、<<=、>>=。如 a+=2 等价于 a=a+2。

%为求余运算符,求余运算的结果与被除数同号。

(3) 数学上的 3<x<5,在 C 语言中的逻辑关系表达式应为 if(3<x && x<5)。

(4) 三目运算符:(?:),例如:

a=a>0? a:-a;//如果? 前为真则取前面的值,否则取后面的值,这里相当于绝对值。

(5) 注意区分逻辑运算符(&&、||、!)与位逻辑运算(&、|、~、^)。

逻辑运算的结果只有两个:1、0,分别表示真、假。规则见表 3-4。

表 3-4 逻辑运算的运算结果规则

变量的数据			逻辑表达式的结果		
表达式	a	b	a&&b	a||b	! a
数据结果	0	0	0	0	1
	0	非零	0	1	1
	非零	0	0	1	0
	非零	非零	1	1	0

对于逻辑值,只要不是 0,都认为是非零,是真。

位逻辑运算将操作数按二进制每一位进行处理,结果仍然是一个整数。举例如下:

```
01  short a=3,b=2,c;                   //00000011、00000010
02  c=a&b;                             //结果 c 为 2:00000010
03  c=a|b;                             //结果 c 为 3:00000011
```

```
04  c=a^b;                          //结果 c 为 1:00000001(每一位分别比较,不同则为 1)
05  c=~a;                           //结果 c 为-4:11111111 11111100(-4 的补码)
```

(6) 移位运算：左移位<<、右移位>>。

```
c=a<<2;     //a 中的二进制数左移 2 位后赋值给 c,左移时最右边位补 0
c=a>>1;     //右移 1 位,有符号数最左边位复制原符号位补到最左,无符号数补 0
            //位逻辑、移位等位运算在硬件控制中比较有用
```

位运算进行位控制和位逻辑判断的例子：

```
01  unsigned short a;
02  a&=0xfffd;                      //将 a 的 a1 位(最右边是 a0 位)置 0,其余位不变
03  a|=4;                           //将 a 的 a2 位置 1,其余位不变
04  if(a&4==1)                      //判断:a2 位是否为 1
05  if(a&2==0)                      //判断:a1 位是否为 0
```

难点：运算符的优先级，参见主教材附录 C。

3.1.5 程序的选择结构

程序的选择结构也称为分支结构。C 语言有 3 种选择结构形式，对比如表 3-5 所示。

表 3-5 3 种选择结构(分支结构)形式对比

if 单分支	if-else 两分支	switch 多分支

if(a>0)　　//注意括号
　　//后无分号
　b=a;
if(a>0);　　//多分号
　b=a;
//逻辑错误,无法选择
	if(a>0)
	　b=a;
	else
	　b=-a;
	//else 与其前面可见的、最近的、未配对的 if 配对
		switch(a)　　//括号中可以是算式
		{//case 后必须是整型或字符型常量
		　case 1: b=1;
		　　break;
		　case 2: b=2; break;
		　　default: b=5;
		}　　//break 根据需要加上

多分支是选择结构的难点。if else if 多分支可替代 switch 多分支，且功能更强。

举例：

(1) year 如果是 4 的倍数且不是 100 的倍数或者是 400 的倍数，就是闰年，输出判断，变量 islunar 赋值为 1；否则赋值为 0。

```
01  if(year%4==0&&year%100!=0||year%400==0)
02                                      //注意区分=与==,=是赋值,==是判断等否
03  {                                   //此处花括号不能少,因为分支中有 2 条语句
04    printf("%d 是闰年。",year);
05    islunar=1;
06  }
07  else
```

```
08  { printf("%d不是闰年。",year);
09    islunar=0;
10  }
```

(2) 判断输出结果。对比下面的程序段,分别分析它们的输出结果。

```
int a=3;
if(a>5)
    if(a>8) { a++;a*=2;}
    else a--;
printf("%d",a);          //将输出 3
//如果第一句改为 int a=6,将输出 5
//如果第一句改为 int a=9,将输出 20
```

```
int a=3;
if(a>5)
{ if(a>8) a++;a*=2;}
else a--;
printf("%d",a);          //将输出 2
//如果第一句改为 int a=6,将输出 12
//如果第一句改为 int a=9,将输出 20
```

(3) 根据整型变量 year、month 的数据,显示该月天数。对比下面正确和错误的程序。

正　　确

```
switch(month)
{
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12: printf("31"); break;  //跳走
    case 4:
    case 6:
    case 9:
    case 11: printf("30"); break;
    default:
        if(year%4==0&&year%100!=0
                    ||year%400==0)
            printf("29");
        else        //if  else 算一条语句
            printf("28");
}
//2018、1 将输出 31
//2018、2 将输出 28
//2018、4 将输出 30
//2016、2 将输出 29
```

错　　误

```
switch(month)
{
    case 1: break;    //逻辑错,什么都没干
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12: printf("31"); //继续向下
    case 4:
    case 6:
    case 9:
    case 11: printf("30"); //继续向下
    default:
        if(year%4==0&&year%100!=0
                    ||year%400==0)
            printf("29");
        else          //if  else 算一条语句
            printf("28");
}
//2018、1 将没有输出
//2018、3 将输出 313028
//2018、2 将输出 28
//2018、4 将输出 3028
//2016、4 将输出 3029
//2016、2 将输出 29
```

(4) 根据百分制分数,显示“优”“良”……对比下面的不同程序实现。

经　　典	可以接受,算正确	错误 X

switch(score/10)
{
　case 9: printf("优");
　　break;
　case 'a': ...
　case '=': ...
　default:
}
	switch(score/10)
	{
	　case 9/10: printf("优");
	　　break;
	　case 3-2: ...
	//switch 后的括号中可以是整
	//型表达式或结果转换为整型
	//case 后的标签可以是常量或
	//无变量的表达式,不能有变量
		switch(score * 0.1)
		{
		　case m * 10:printf("优");
		　　break;
		...
		//错误原因:
		//switch 后括号中是实型
		//case 后的标签不是常量

3.1.6　程序的循环结构

C 语言有 3 种循环结构形式,如表 3-6 所示。

表 3-6　3 种循环结构形式对比

for 循环	while 循环	do…while 循环

for(i=0;i<5;i++)
{ …… }
//注意 for 的括号后面不能有分号
//本循环方法最常用
	while(e>=1e-5)
	{ …… }
	//注意 while 的括号后面不
	//能有分号,否则逻辑错;没
	//有进行循环
		do
		{ …… }while(e>=1e-5);
		//注意 while 的括号后面必须有分
		//号,否则语法错

循环嵌套、循环控制变量、循环中变量的迭代是难点。

举例:

(1) 求 1!+2!+3!+4!+5!=?。

```
01  #include<stdio.h>
02  int main()
03  { int i;
04    double term=1.,sum=0.;
05              //定义为双精度实型变量,表示范围更大,更不容易溢出
06    for(i=1;i<=5;i++)
07    { term=term * i;                              //或写为 term * =i;
08      sum=sum+term;                               //或写为 sum+=term;
09    }
10    printf("%e",sum);
11    return 0;
12  }
```

(2) 求 1! * 2! * 3! * 4! * 5!=?。

```
01  #include<stdio.h>
02  int main()
```

```
{ int i;
  double term=1.,sum=1.;          //注意初始值与上一题的区别
  for(i=2;i<=5;i++)               //循环可以从 2 开始
  { term=term*i;                  //或写为 term*=i;
    sum=sum*term;                 //或写为 sum*=term;
  }
  printf("%e",sum);
  return 0;
}
```

(3) 编程输出图形：

```
    *
   ***
  *****
 *******
```

```
#include<stdio.h>
int main()
{ inti,j;
  for(i=0;i<4;i++)                //控制输出 4 行
  { for(j=0;j<10-i;j++)           //控制输出空格
                                  //注意理解循环次数为什么是 10-i
      printf(" ");
    for(j=0;j<2*i+1;j++)
                                  //控制输出*,注意理解循环次数为什么是 2*i+1
      printf("*");
    printf("\n");                 //换行不能少
  }
  return 0;
}
```

(4) 根据 $\pi=4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots\right)$,计算 π 的近似值,要求误差小于 10^{-5}。

方法 1：

```
#include<stdio.h>
int main()
{ int i=1,sign=1;
  double pi=0.,term=1;            //注意初始化赋值
  while(term>2.5e-6)              //注意循环条件
  { term=1./i;                    //注意不能写为 term=1/i; 那样始终为 0
    pi=pi+sign*term;              //或写为 pi+=sign*term;
    sign=-sign;                   //注意如何控制循环中符号的变化
    i+=2;
  }
  pi*=4;                          //注意不能掉了这一句
  printf("%f",pi);                //输出结果为 3.141598,误差为 8*10-6
  return 0;
}
```

方法 2：

```
#include<stdio.h>
int main()
{ int i=1,sign=1;
  double pi=0.,term;                    //注意初始化赋值与方法 1 的变化
  do
  { term=1./i;                          //注意不能写为 term=1/i; 那样始终为 0
    pi=pi+sign*term;                    //或写为 pi+=sign*term;
    sign=-sign;
    i+=2;
  }while(term>2.5e-6);                  //注意分号不能掉
  pi*=4;
  printf("%f",pi);                      //输出结果为 3.141598,误差为 8*10-6
  return 0;
}
```

（5）根据 $\pi=3+4\left(\frac{1}{2\times3\times4}-\frac{1}{4\times5\times6}+\frac{1}{6\times7\times8}-\cdots\right)$ 计算 π 的近似值，要求误差小于 10^{-9}。

```
#include<stdio.h>
int main()
{ int i=2,sign=1;
  double pi=0.,term;                    //注意初始化赋值
  do
  { term=1./i/(i+1)/(i+2);
    pi+=sign*term;
    sign=-sign;
    i+=2;
  }while(term>2.5e-10);                 //注意分号不能掉
  pi=3+4*pi;
  printf("%.19f",pi);                   //输出结果为 3.1415926531,误差为 5*10-10
  return 0;
}
```

3.1.7 特殊的程序流程控制语句

循环中碰到 break 语句，将结束本层的循环，执行本层循环语句后面的句子。循环中碰到 continue 语句，将跳过本次循环后面的语句，进入循环控制。比较示例如下：

```
for(i=0;i<2;i++)
{
    for(j=0;j<3;j++)
    {
        cout<<j;
        cout<<",";
    }
    cout<<endl;
}
```

```
for(i=0;i<2;i++)
{
    for(j=0;j<3;j++)
    {
        cout<<j;
        if(j==1) break;
        cout<<",";
    }
    cout<<endl;
}
```

```
for(i=0;i<2;i++)
{
    for(j=0;j<3;j++)
    {
        cout<<j;
        if(j==1) continue;
        cout<<",";
    }
    cout<<endl;
}
```

续表

将输出： 0,1,2, 0,1,2,	将输出： 0,1 0,1	将输出： 0,12, 0,12
解释： 外循环 2 次、嵌套每次内循环 3 次。	解释： j 为 1 时，内循环提前结束，进入第 2 次外循环，且再次当 j 为 1 时再次提前结束内循环。	解释： j 为 1 时，不执行后面的输出逗号语句，直接去执行 j++。

3.1.8 函数

1. 函数的声明、定义

函数是 C 语言的程序结构基本单位，一个函数就是一个功能、结构独立的模块。自定义函数名必须遵守标识符命名规则。

与函数有关的有声明、定义、调用。声明就是表示有这么一个函数。定义就是详细写出函数的完整程序语句。调用就是使用函数。

求两个实数平均值的函数的示例：

```
#include<iostream>
using namespace std;
double my_mean(double, double);          //函数的声明，必须给出函数名、返回值类型、
//参数类型、参数个数，注意函数声明最后有分号
//声明时参数名可以给或不给出
//自定义函数必须先声明、后调用
int main(void)
{
    double a=3,b=4,c;
    c=my_mean(a,b);                      //函数的调用，给出参数的实际值
    return 0;
}
//-------------------------------------------------------------
double my_mean(double x, double y)       //函数的定义，注意头部后面没有分号
//定义时必须给出参数名、参数类型、返回值类型
//定义时的参数个数、各类型必须与声明一致
//参数名可以与声明时参数名不同
{
    return (x+y)/2;                      //函数的返回值，声明时函数无返回类型的可
                                         //以没有 return 语句
}
```

函数一般会返回一个值，可以是各种类型，如整型、实型、字符型、指针、结构体类型。

难点：参数传递、递归函数、静态变量。

2. 函数的参数传递

调用函数时，向函数传递的是值，而不是变量；但是传递指针可以使指针所指向的空间

或后续连续空间在函数中得到赋值而相当于得到返回值。通过指针作参数，可以在主调函数和函数之间相互传递多个数据。

函数只能有一个显性返回值，函数中通过 return 返回，调用通过函数名得到该返回值。

例如：

```
#include<stdio.h>
int swap(int a, int b)                    //函数的定义,此时声明、定义在一起
{                                         //这里的 a、b、c 是函数里面的局部变量
  int c;                                  //与主程序中的 a、b、c 没有任何关系
  c=a,a=b,b=c;
  return a+b;
}
int main(void)
{ int a=3,b=4,c;
  c=swap(a,b);                            //传递给函数的是数据 3、4,而不是变量 a、b
  printf("a=%d, b=%d, c=%d",a,b,c);
  return 0;                               //输出仍然是 a=3, b=4, c=7
}
```

如果想交换两个参数的值，并返回主调函数，应修改为

```
#include<stdio.h>
int swap(int *a,int *b)                   //参数为整型指针,这里参数名改为 pa 更好
{ int c;
  c=*a,*a=*b,*b=c;                        //*是间接寻址运算符
  return *a+*b;
}
int main(void)
{ int a=3,b=4,c;
  c=swap(&a,&b);                          //传递给函数的是 a、b 的地址,&是取地址运算符
  printf("a=%d, b=%d, c=%d",a,b,c);
  return 0;                               //输出就变为了 a=4, b=3, c=7
}
```

3. 递归函数

在函数定义中调用自己，或者 A 函数定义中要调用 B 函数，而 B 函数定义中又会调用 A 函数。注意：定义中肯定有选择结构，某种情况下不再调用，而是可以得到结果。

例如，求数列的值，f(0)＝0，f(1)＝1，f(n＋2)＝f(n＋1)＋f(n)，n＝0,1,2,…

```
int fun(int n)                            //函数定义
{ if(n==0) return 0;
  if(n==1) return 1;                      //这里没必要写 else,写上也可以
  return fun(n-1)+fun(n-2);
}
```

4. 变量的作用域、生命期、可见性

局部变量：在一个块中(也就是花括号中)定义的变量就是局部变量，作用域只在该块内，但在其生命期内可以通过地址使块外访问(静态变量不同)。一般数据中的变量就是这一类。块运行结束后生命期结束(静态变量不同)。

同一块中不能定义重名的两个局部变量，但不同块或不同层次的块的局部变量可以重名，甚至局部变量可以和全局变量重名。重名时块内的变量屏蔽块外的变量。

全局变量：在所有块外定义的变量，所有块可以访问。

静态变量：关键字为 static，生命期一直到程序结束，只初始化一次，块结束后不可见。

例如：

```
#include<iostream>
using namespace std;
int main(void)
{   int i=-5,j;          //局部变量 i
    for(j=0;j<3;j++)
    {   int i=0;         //子块中的 i
                         //重名局部变量 i
        cout<<i++;
    }                    //输出 0, 0, 0,
    cout<<i;             //输出-5
    return 0;
}
```

```
#include<iostream>
using namespace std;
int main(void)
{   int i=-5,j;          //局部变量 i
    for(j=0;j<3;j++)
    {   static int i=0;
        //静态局部变量 i,这一句
        //在循环时不第二次执行初始化
        cout<<i++;
    }                    //输出 0, 1, 2,
    cout<<i;             //输出-5
    return 0;
}
```

全局变量、局部变量示例：

```
#include<stdio.h>
extern int m;                                  //全局变量声明
fun()
{ int a=3,b=4;
  m++;
  printf("a=%d, b=%d, m=%d",a,b,m);            //输出 a=3, b=4, m=2
}
int m=1;                                       //全局变量定义
int main()
{ int a=-3,b=-4;                               //与函数 fun()中的 a、b 没有关系
  fun();                                       //调用函数
  m*=2;
  printf("a=%d, b=%d, m=%d",a,b,m);            //将输出 a=-3, b=-4, m=4
  return 0;
}
```

静态局部变量示例：

```
#include<stdio.h>
int m=0;                                  //全局变量
fun()
```

```
{ static int a=3,b;                          //定义静态局部变量并初始化
  b=1,a++,m++;                               //b=1 不是初始化,而是赋值,循环每次都要执行
  printf("a=%d, b=%d, m=%d, ",a,b,m);
}
int main()
{    int i;
    for(i=0;i<3;i++)
        fun();                               //调用函数 3 次
    printf("m=%d",m);                        //输出 m=3
    return 0;
}
```

程序运行将输出：a=4，b=1，m=1，a=5，b=1，m=2，a=6，b=1，m=3，m=3。

3.1.9 数组

1. 数组的定义、引用和初始化

数组就是一组变量，这些变量类型相同、名称相同但下标不同、连续存放。

定义一个一维数组后，其数组名就成了一个指针，表示第一个元素的地址、类型。

```
#include<stdio.h>
int main()
{ int a[5],i;                //数组定义,数组名为 a,长度为 5,元素为 a[0]到 a[4]
                             //此时,数组 a 的各元素的值是不确定的,未必是 0
  int b[]={1,3,0,2,5};       //定义时可以一次性初始化全数组元素,长度为 5
  int c[8]={3};              //除第一个元素外,c 的其余元素值为 0
  a[0]=3;                    //数组元素引用,不能简单用数组名来试图访问全部元素
  a[4]=b[1]*2;               //每次访问一个元素,可通过循环来访问全部元素
  for(i=0;i<5;i++)
    a[i]=i*2;                //通过循环语句,访问数组的全部元素。注意下标的范围
  return 0;
}
```

难点：循环中数组的下标选择、二维数组。

2. 常见错误

正　　确	错　　误

#define N 100
int main(void)
{
　int a[N];　　//编译时长度确定
　int b[11/2]={5};　//长度确定
　b[1]=a[2];
　a[2]=3;
　return 0;
}
	int main(void)
	{
	　int i=5,b[5];
	　int a[i];　　//错误!编译时长度不确定
	　a=b;　　//错误!数组名是指针常量
	　b[5]=5;　　//错误!下标越界,逻辑错误
	　b={1,3};　　//错误!无法访问整个数组
	　return 0;
	}

3. 数组名作函数的参数

数组名作函数的参数，向函数传递的是地址，这样可以将多个值通过该地址空间开始的连续空间带回。

例如，输入 10 个分数，求平均分及每个分数的降序名次。

```
#include<stdio.h>
#define N 10
double mySort(int score[],int number[]);
                                          //函数的声明,参数是 2 个数组:分数、名次
int main()
{ int i,a[10],b[10];
  double average;
  printf("Please input 10 scores : ");
  for(i=0;i<N;i++)
    scanf("%d",&a[i]);                    //输入 10 个分数
  average=mySort(a,b);                    //函数调用
  printf("\n 平均分是: %.2f\n 名次分别是: ",average);
  for(i=0;i<N;i++)
    printf("%d, ",b[i]);                  //输出 10 个名次
  return 0;
}
double mySort(int score[],int number[])
{ int i,j,sum=0;
  for(i=0;i<N;i++)
  { number[i]=1;                          //初始名次为 1
    for(j=0;j<N;j++)
      if(score[i]<score[j])
        number[i]++;                      //有一个比它大的,则名次降 1
      sum+=score[i];                      //求分数和
  }
  return (double)sum/N;
}
```

4. 三种排序算法（以升序排序为例）

（1）交换法：依次找出最小的数交换到第一个元素、次小的数交换到第二个元素，以此类推。用第一个元素与后面的全部元素依次比较，如果后面某个元素比它小，则交换。全部比较后第一个元素就是最小，然后第二个元素同样与其后面的元素比较、交换……

```
void sortExchange(int a[],int n)          //对数组 a 的 n 个元素进行升序排序
{ int i,j,m;
  for(i=0;i<n-1;i++)                      //依次找出 n-1 个最小数、次小数…
    for(j=i+1;j<n;j++)                    //j 是 a[i]后面的所有元素的下标
      if(a[i]>a[j])                       //若后面大则交换
                                          //以使 a[i]始终比其后面的元素小
      { m=a[i];
        a[i]=a[j];
        a[j]=m;
      }
}
```

(2) 选择法：与交换法类似，依次找出最小的数交换到第一个元素、次小的数交换到第二个元素，以此类推。只是每次用最小的元素与后面的全部元素依次比较，如果后面某个元素比它小，则记下新的最小数的下标。全部比较后，就得到最小值的下标，将其与第一个元素交换。然后第二小的数同样与其后面的元素比较、记下下标，最后交换……这样交换次数少，只是选择最小的值与第一个交换。

```
01  void sortSelect(int a[],int n)                    //对数组 a 的 n 个元素进行升序排序
02  { int i,j,k,m;                                    //变量 k 表示最小数的下标
03    for(i=0;i<n-1;i++)                              //依次找出 n-1 个最小数、次小数…
04    { k=i;                                          //先假设第一个最小
05      for(j=i+1;j<n;j++)                            //j 是 a[i]后面的所有元素的下标
06        if(a[k]>a[j])                               //若后面大则修改 k
07                                                    //以使 k 始终是 a[i]后最小数的下标
08          k=j;                                      //记下新的下标到 k
09        m=a[i];
10        a[i]=a[k];
11        a[k]=m;                                     //将第一个与选择的最小的元素交换
12    }
13  }
```

(3) 冒泡法：从最后一个元素开始，与其前面相邻的元素比较，若后面的元素小则交换。然后次后的元素同样与其前面相邻的元素比较、交换。到第二个后则最小的元素“冒泡”到了最前面。然后将次小的数冒出、第三小的数冒出……

```
01  void sortBubble(int a[],int n)  //对数组 a 的 n 个元素进行升序排序
02  { int i,j,m;
03    for(i=0;i<n-1;i++)             //通过"冒泡"依次找出 n-1 个最小数、次小数…
04      for(j=n-1;j>i;j--)           //j 是从后向前冒泡的元素下标
05        if(a[j]<a[j-1])            //若后面的元素小则交换,以使相邻两个元素始终前面的小
06        { m=a[j];
07          a[j]=a[j-1];
08          a[j-1]=m;
09        }
10  }
```

冒泡法可以改进，当一次冒泡中没有发生交换，则表示已符合顺序条件，可以提前结束。

```
01  void sortBubble(int a[],int n)//对数组 a 的 n 个元素进行升序排序
02  { int i,j,sign,m;             //sign 表示是否有交换
03    for(i=0;i<n-1;i++)          //依次找出 n-1 个最小数、次小数…
04    { sign=0;
05      for(j=n-1;j>i;j--)        //j 是从后向前冒泡的元素下标
06        if(a[j]<a[j-1])         //若后面的元素小则交换,以使相邻两个元素始终前面的小
07        { m=a[j];
08          a[j]=a[j-1];
09          a[j-1]=m;
10          sign=1;               //有交换,则标记为 1
```

```
        }
        if(sign==0)                    //一轮相邻元素比较下来无交换,则提前结束
          break;
    }
}
```

3.1.10 指针

1. 指针的概念、指针变量的定义与使用

指针是一种新的数据类型,它表示变量的地址和该地址空间中数据的数据类型。指针同样有指针变量、指针常量。

例如,定义 int *p; 后,p 就是指针变量,定义 int m,a[5]; 后,&m、a 都是指针常量。&m 表示变量 m 的地址及该地址中数据类型是整型。a 表示变量 a[0]的地址及类型为整型。

指针变量的定义、初始化、赋值、使用举例:

```
#include<stdio.h>
int main()
{ int a[5],m,i,*p1,*p2=a;          //定义了两个指针变量
  int b[]={1,3,0,2,5};
  p1=b;                            //给指针赋值
  p2[0]=33, *(p2+2)=55;            //使用指针间接寻址
  printf("%d, %d, %d",a[0],*p1,*(p1+4));
                                   //相当于输出*p2、b[0]、b[4],将输出显示 33,1,5
  p1=p2;                           //指针变量重新赋值
  *p1=22;                          //相当于 a[0]=22
  printf("%d, %d, %d",a[0],*p1,*(p1+2));
                                   //相当于输出*p2、a[0]、a[2],将输出显示 22,22,55
  return 0;
}
```

难点:指针的使用、间接寻址。& 是取地址运算符,* 是间接寻址运算符。

2. 常见错误

指针使用中的常见错误举例如表 3-7 所示。

表 3-7 指针使用中的常见错误举例

正 确	错 误

int main(void)
{ int a[5],*p;
p=a;
p=&a[2]; *p=3;
a[0]=*p+1;
a[2]=*(p+3);
}
	int main(void)
	{ int a[5],b[5],*p=b;
	a=p; //a 是指针常量
	a=&p[2]; //a 是指针常量
	*a=3; p[0]=*a+1; //正确,间址
	a[2]=*(p+3); //正确
	} //指针常量不能赋值,但可以用来间址

3. 指针作函数的参数

指针作函数的参数，向函数传递的是地址，这样可以将一个或多个值通过该地址空间开始的连续空间带回。与数组名作函数参数类似。

例如，给函数两个参数值，在函数里让它们交换数据，并返回结果(要返回两个结果)。

```
#include<stdio.h>
void Swap(int *,int *);                          //函数的声明,参数是 2 个整型指针
int main()
{   int a,b;
    printf("Input 2 data : ");
    scanf("%d%d",&a,&b);                         //输入 2 个整数
    Swap(&a,&b);                                 //通过 a、b 将结果带回
    printf("%d, %d",a,b);                        //输出交换后的数据
    return 0;
}
void Swap(int *p1,int *p2)
{   int m;                                       //中间变量
    m=*p1;                                       //间接寻址访问 a
    *p1=*p2;                                     //间接寻址访问 a、b
    *p2=m;
}                                                //若输入 2 3,结果为 3,2
```

3.1.11 字符串

1. 字符串的概念、字符串的使用

字符串是特殊的数据常量，它是连续存储的字符型数据并以\0 结束。由于在人机交互、字符型信息处理中经常用到字符串，所以定义了字符串的表示形式及字符串处理函数。

字符串的处理常常是用字符指针，它的表示标志是双引号。

以下是字符串的使用举例，用来检测输入的密码是否正确。

```
#include<stdio.h>
#include<string.h>                    //字符串处理函数头文件
int main()
{ char a[100]="Welcome!";             //定义一个数组,初始化存储了一个字符串
  char *p="China";                    //定义了一个字符指针,指向一个字符串常量
  char b[100];
  gets(b);                            //输入一个字符串,按回车键或空格键结束
  if(strcmp(p,b)==0)                  //比较两个字符串是否相等的函数
    puts(a);
  else
    puts("密码错误!");
  return 0;
}
//数组 a 中实际存储:'W'、'e'、'l'、'c'、'o'、'm'、'e'、'!'、'\0'。
//指针 p 指向的内存存储情况:'C'、'h'、'i'、'n'、'a'、'\0'。
//字符串最后的'\0'是计算机自动加上的
//这里,*p 只能读,不能写;数组 a 的 *a 能读、能写
```

重点：字符串的存储、相关函数。

2. 常见错误

字符串使用中的常见错误举例如表 3-8 所示。

表 3-8　字符串使用中的常见错误举例

正　　确	错　　误
int main(void) {　char a[5]="Tom", * p1=a, * p2="Go!"; 　printf("%s%s%c",p2,p1, * (p2+2)); 　return 0; }　　//将输出 Go!Tom!	int main(void) {　char a[5]="Tom", * p1=a, * p2="Go!"; 　* p2='a';　　//字符串不能重新赋值 　a[0]='a';　　//数组可以重新赋值 　return 0; }

3. 字符串处理相关函数

```
#include<string.h>                    //字符串处理函数头文件
```

常用的字符串处理函数如表 3-9 所示。

表 3-9　常用的字符串处理函数

函 数 原 型	解　　释
char * strcpy(char * str1,char * str2);	串复制，str2 指向的串复制到 str1，返回 str1
int strcmp(char * str1,char * str2);	串比较，str1 小则为负，相等则为 0，大则为正
char * strcat(char * str1,char * str2);	串连接，str2 指向的串加到 str1 后，返回 str1
unsigned int strlen(char * str1);	统计 str1 串长度，不计结尾的 0(写为'\0')

4. 字符串的输入输出

C 语言的输入输出需要加上头文件：

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

C++ 的输入输出流方法需要加上头文件和命名空间语句：

```
#include<iostream>
using namespace std;
```

字符串的常用输入输出函数如表 3-10 所示。

表 3-10　字符串常用的输入输出函数

函　　数	解　　释
scanf("%s",p);	输入字符串，遇到空格、Tab 键或回车符就结束，且不读入空格、Tab 键或回车符及后面字符，留在输入缓冲区队列。p 是字符指针或字符数组名(C 语言的输入函数)

续表

函　　数	解　　释
printf("%s",p);	输出字符串,遇到'\0'结束,可输出空格等。p 是字符指针或字符数组名(C 语言的输出函数)
cin>>p;	输入字符串,遇到空格、Tab 键或回车符就结束,且不读入空格、Tab 键或回车符及后面字符,留在输入缓冲区队列。p 是字符指针或字符数组名(C++ 的输入)
cout<<p;	输出字符串,遇到'\0'结束,可输出空格等。p 是字符指针或字符数组名(C++ 的输出)
gets(p);	输入字符串,回车符才结束,可接收空格、Tab 键,且将回车符读走但不作为字符串的一部分。p 是字符指针或字符数组名
fgets(p,N,stdin);	输入字符串,最多输入 N−1 个或遇回车符结束,且将回车符读走并存入 p 中。C11 标准已推荐此函数替代 gets()函数
puts(p);	输出字符串,遇到'\0'结束,可输出空格等,并在最后输出回车符。p 是字符指针或字符数组名

3.1.12　C 语言的动态内存管理

1. 概念

数组在定义时必须确定其长度,如果不确定需要使用多长,必须将数组定义得很大,比较浪费空间。动态内存使用可以在程序运行时根据需要来申请空间。

2. 方法

头文件 #include<stdlib.h>

函数 malloc(字节数)将申请该字节数的连续内存空间,返回申请的内存空间的头指针,但指针无类型。如果没申请到将返回 NULL。必须将该指针做类型转换后赋值给某指针变量。

函数 calloc(个数,块字节数),将申请该个数 * 块字节数的连续内存空间,返回申请的内存空间的头指针,但指针无类型。如果没申请到将返回 NULL,并将空间全部赋值为 0。

函数 free(p)将释放申请的空间。

使用举例:根据输入的数字,申请一个动态整型数组。C11 标准已直接支持动态数组。

```
01  #include<stdio.h>
02  #include<stdlib.h>
03  int main()
04  { int m, *p;                                    //定义整型指针变量
05    scanf("%d",&m);                               //输入长度给变量 m
06    p=(int *)malloc(m*sizeof(int));               //申请 m 个整型空间
07    if(p==NULL)
08      { printf("申请内存出错");exit(1);}           //申请失败将显示并退出
09    p[0]=5;                          //可以使用该空间,建议不要修改 p 的值,否则不好释放了
10    printf("%d",p[0]);
11    free(p);                                      //释放申请的空间
12    return 0;
13  }
```

重点：使用的语法格式，动态空间使用完后注意要释放。

3.1.13 构造数据类型

1. 结构体

为了更方便地存储具有不同类型的复杂数据，如学生信息，包括姓名（字符数组）、学号（字符数组）、性别（字符型）、分数（整型）等，可自己定义一个数据类型，每个变量将包括多个成员，每个成员有名称、自己的类型。如果没有结构体的概念，通过多个变量或多个数组也可以实现存储全班学生信息，但不如结构体方便。数字图像处理、自动控制等的 C 语言函数中常用到函数体。

1）格式 1

```
#include<stdio.h>
#include<string.h>
struct student                      //struct 是关键字，student 是自定义的结构体类型名
{ char name[10];                    //3 个成员
  int math;
  int computer;
}stu1={"Li",80,88};                //定义结构体变量并初始化，是全局变量
int main()
{ struct student stu2, * p=&stu2;  //定义结构体变量、指针
  stu1.math=95;                     //一次只能访问一个成员
  strcpy(stu2.name,stu1.name);      //字符串复制
                                    //访问成员的格式：stu1.name
  stu2=stu1;                        //同类型的结构体变量可以直接赋值
  printf("%s: %d",p->name,p->math);
                                    //通过指针访问成员，格式：p->name
  return 0;
}
```

2）格式 2

```
#include<stdio.h>
#include<string.h>
typedefstruct student                          //两个关键字，给结构体类型以别名 STD
{ char name[10];                               //3 个成员
  int math;
  int computer;
}STD;
STD stu1={"Li",80,88};                         //定义结构体变量并初始化，是全局变量
int main()
{ STD stu2,stu[5], * p=&stu2;                  //定义结构体变量、指针、结构体数组
  stu1.math=95;                                //一次只能访问一个成员
  strcpy(stu2.name,stu1.name);                 //数组的复制
  stu2=stu1;                                   //同类型的结构体变量可以直接赋值
  printf("%s: %d",p->name,p->math;             //通过指针访问成员
  return 0;
}
```

2. 枚举类型

枚举类型还是整型，不过以集合的形式列出了所有的取值，即枚举常量。这样使用时直观些，但实质还是保存的是整数。

```
enum week{SUN,MON,TUE,WED,THU,FRI,SAT}today;
                                                    //定义了枚举类型 week
enum week nextday;
                                                    //定义了枚举变量 today、nextday
                                                    //内存中 SUN 保存为 0,MON 为 1,…
enum response{no=-1,yes=1,none=0};                  //也可以设定为某个常量的值
nextday=SUN;                                        //相当于 nextday=0
```

重点：结构类型、枚举类型的使用格式。

3.1.14 文件操作

1. 概念

如果希望长期保存数据，就必须将数据保存到文件中。常常还需要读取文件中的数据。这些都是文件的操作。文件的操作有 4 个函数：打开文件(fopen)、读文件(fread)、写文件(即将数据保存到文件，fwrite)、关闭文件(fclose)。对文件是以指针进行应用的。

两种文件类型如下。

(1) 文本文件：全部内容以 ASCII 码表示、保存，适合保存文档。

(2) 二进制文件：全部是以二进制表示、保存，适合保存数据。

文件是怎么保存的就需要怎么读取。

2. 文件操作的一般方法(注意加粗的常用相关函数或语句)

```
#include<stdio.h>
#include<stdlib.h>                          //必须用该头文件
int main()
{ FILE * fp;                                //文件指针
  int i,a[10];
  double d[10];
  if((fp=fopen("demo.dat","wb+"))==NULL)
  { printf("Failure to open file.\n");
    exit(1);
  }                                         //打开文件,如果不存在则显示出错并退出
  fread(d,sizeof(double),5,fp);             //读文件
  fwrite(a,sizeof(int),10,fp);              //写文件
  fclose(fp);                               //关闭文件
  return 0;
}
```

函数 fread(d,sizeof(double),5,fp)中，d 必须是内存的指针，5 表示读取的数据个数。

文件读、写完成后必须关闭文件。