1. 函数

一般形式：

返回值类型 函数名（参数）

{

函数主体；

}

1. 返回值类型：一个函数可以返回一个值。有些函数执行所需的操作而不返回值，在这种情况下，返回值类型是关键字 void。
2. 函数名：函数的实际名称
3. 参数：参数是可选的，也就是说，函数可能不包含参数。
4. 函数主体：函数主体包含一组定义函数执行任务的语句。

函数的声明

如果要在函数定义之前调用函数，那么必须对调用的函数进行声明。

格式：

返回值类型 函数名（参数）

参数名可以省略，但是参数的类型不能省略

举例：

#include "stdafx.h"

#include <iostream>

#include <math.h>

using namespace std;

void FunA(int a);

void FunB(int);

int main()

{

FunA(1);

FunB(2);

system("PAUSE");

}

void FunA(int a)

{

cout << a << endl;

}

void FunB(int a)

{

cout << a << endl;

}

1. 随机数

**rand()**函数返回一个伪随机数。**srand()** 函数生成一个随机种子。

取得一定范围内的随机数

取得(0,x)的随机整数：rand()%x

取得(a,b)的随机整数：rand()%(b-a)

取得[a,b)的随机整数：rand()%(b-a)+a

取得[a,b]的随机整数：rand()%(b-a+1)+a

取得(a,b]的随机整数：rand()%(b-a)+a+1

取得0-1之间的浮点数：rand()/double(RAND\_MAX)

举例：

#include "stdafx.h"

#include <iostream>

#include <ctime>

#include <cstdlib>

using namespace std;

int main()

{

int i, j;

// 设置种子

srand((unsigned)time(NULL));

/\* 生成 10 个随机数 \*/

for (i = 0; i < 10; i++)

{

// 生成实际的随机数

j = rand();

cout << "随机数： " << j << endl;

}

system("PAUSE");

return 0;

}

1. 数组

数组是一个固定大小的相同类型元素的顺序集合。所有的数组都是由连续的内存位置组成。最低的地址对应第一个元素，最高的地址对应最后一个元素。数组大小一旦确定那么便不可修改，后续会有可修改大小的类数组，以后用到再说。

**数组的声明**

类型 数组名[元素数量]

例如：double arr[10];一个含有十个double类型元素的数组

**数组的初始化**

double arr[4] = {10.0, 2.0, 30.0, 4.0};//元素个数不得超过四

double arr[] = {10.0, 2.0, 30.0, 4.0};//元素个数自动生成

**数组的访问**

数组名[下标]

注意：下标是从零开始的，使用时不要越界

**数组的赋值**

数组名[下标]=值，注意：值要与数组的类型一样

如：arr[0] = 10;

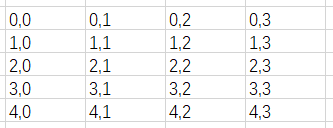
**二维数组**

格式：

类型 数组名[元素数量] [元素数量]

double arr[4][5]

二维数组可以近似看做一张表格，4代表了每一行有四个元素，5代表了每一列有5个元素，如图所示



a[0]={11,22,33,44}

a[1]={11,22,33,44}

a[2]={11,22,33,44}

a[3]={11,22,33,44}

a[4]={11,22,33,44}

1. 字符串

字符串实际上是使用 **null** 字符 '\0' 终止的一维字符数组。

**创建字符串的三种方式**

char strA[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };

char strB[6] = "Hello";

string strC = "Hello";

字符串只能在创建时整体赋值，创建之后不能整体赋值。但可以使用指针整体赋值。

**用char数组创建字符串是c语言遗留下来的风格，使用时不能完全按照c的风格使用，在不同的c++标准中对char的支持不同！！！**

1. 指针

**指针**是一个变量，其值为另一个变量的地址，即内存位置的直接地址。指针可以直接操纵地址中的数据去间接修改变量的值。**指针**既然是变量，那么一定有大小，指针的大小根据系统位数而定，32位为4字节，64位为8字节，同时也跟创建的文件有关，如果在64位系统里创建32位程序，那么指针的占用4个字节。必须在使用指针存储其他变量地址之前，对其进行声明。

* 1. **两个有关运算符**

&：取地址运算符

\*：指针运算符

* 1. **创建指针**

类型 \*指针名；

int \*ip;

* 1. **指针赋值**

因为指针只能存放地址，所以应对变量取地址，然后赋给指针。

注意：不能将数字，字符等非地址值赋给指针，也不能将不同类型的指针和变量进行赋值

先定义时赋初值

Int a=10;

int \*ip=&a;

先定义，在赋地址

Int a=10;

Int \*p;

P=&a;

* 1. **使用指针对变量进行控制**

指针可以直接操纵地址中的数据去间接修改变量的值。

举例：

#include "stdafx.h"

#include <iostream>

using namespace std;

int main()

{

int \*p; //定义指针

int a = 10;

p = &a; //把a的地址赋给指针p

cout << \*p << endl;

\*p = 20; //通过指针p修改指针a地址中的数据，从而实现对a的修改

cout << \*p << endl;

system("PAUSE");

return 0;

}

* 1. **野指针**

当一个指针没有任何指向时便称为野指针，这样的指针不知指向哪里，是一种常见且会造成严重后果的问题。为此，我们应该使指针实时有所指向，可以将指针赋为空。这在后来的对象销毁中尤为关键。

如：

Int \*p=NULL;

p = NULL;

* 1. **值传递与按地址传递**

在函数中使用普通变量只能进行值传递，无法对传进来的变量进行修改

如：

#include "stdafx.h"

#include <iostream>

using namespace std;

void func(int a,int b)

{

int step = a;

a = b;

b = step;

}

int main()

{

int var\_1=2;

int var\_2 = 4;

func(var\_1, var\_2);

cout << var\_1 <<","<<var\_2<< endl;

system("PAUSE");

return 0;

}

**使用指针作为函数参数**

使用指针作为形参，可以实现按地址传递，从而实现修改变量的功能

如：

#include "stdafx.h"

#include <iostream>

using namespace std;

void func(int \*a,int \*b)

{

int step = \*a;

\*a = \*b;

\*b = step;

}

int main()

{

int var\_1=2;

int var\_2 = 4;

int \*p, \*q;

p = &var\_1;

q = &var\_2;

func(p, q);

cout << var\_1 <<","<<var\_2<< endl;

system("PAUSE");

return 0;

}

* 1. **int\*p[2]与 int (\*p)[2]**

int \*p[2]，“[]”的优先级别高，所以它首先是个大小为2的数组，即p[4]；剩下的“int \*”作为补充说明，即说明该数组的每一个元素为指向一个整型类型的指针。

举例：

int \*p[2];

int a = 10;

p[0] = &a;

int (\*p)[2]，首先，(\*p)表明p是一个指针，指向一个一维数组，该数组有4个元素，元素的类型是int。指向数组的指针就是说一个指针指向了整个数组，也就是说这个指针指向的是整个数组的空间，此时对这个指针进行自加操作得到的地址就是向后移动整个数组长度的地址。注意举例中访问数据的方式。

举例：

int (\*p)[3];

int a[3] = { 1,2,3 };

p = &a;

cout << p[0][0] << endl; //输出结果为1

1. **指针与数组**

一维数组各元素在内存中是由连续的一块内存单元组成的。数组名就是这块连续内存单元的首地址，每个数组按类型不同分别占不同的内存单元，比如char类型占一个内存单元，float占四个内存单元。

例如：

**int** a=[10],\*p;

p=&a[0];

因为数组名是数组连续内存单元的首地址，所以a=&a[0],

也就是说，p=&a[0]与p=a是等价的

1. 使用指针操作数组元素

int a[10];

int \*p=a; //\*p此时指向数组a的首元素，即p=&a[0]

\*p=1; //即a[0]=1

1. 指针的加减

指针会根据指针类型来决定加减所移动的内存单元，这使得指针可以操作数组和字符串（字符串就是个数组，从c语言习惯的char类型字符串就可以看出来，string只不过进行了封装和改进，它的实现方式还是用数组的方式）。注意指针不要越位，造成野指针！

int a[10] = {0,1,2,3,4,5,6,7,8,9,};

int \*p=a; //\*p此时指向数组a的首元素，即p=&a[0]

++p; //此时p指向的是a[1]

cout<<\*p<<endl; //打印a[1]的值

cout<<\*p++<<endl; //打印a[1]的值

cout<<\*++p<<endl; //打印a[3]的值

1. begin、endl函数

begin函数返回数组首元素的地址，end函数返回数组尾元素下个元素的地址。在使用时需要引入头文件<iterator>。

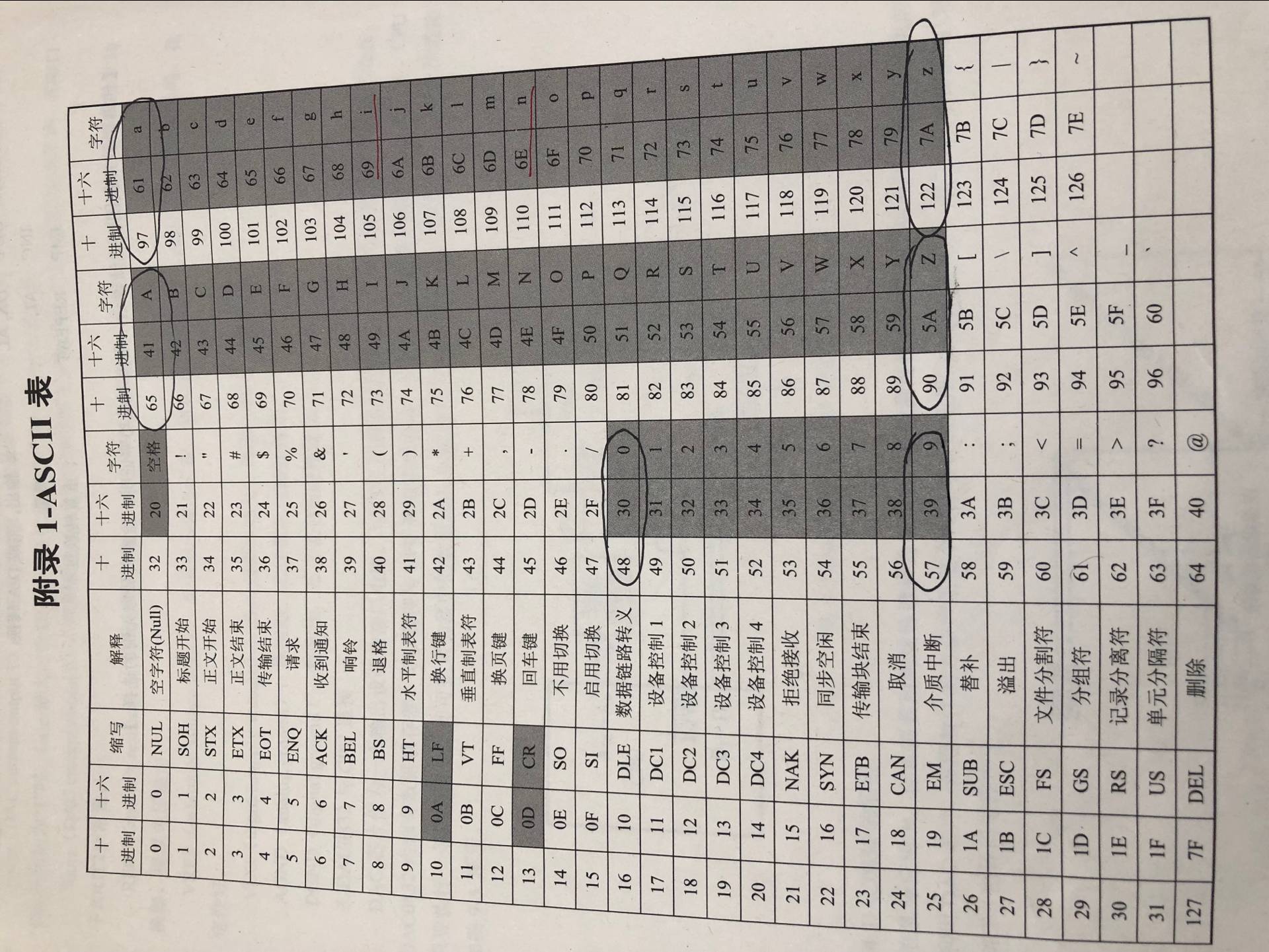
举例

Int a[3];

Int \*p=begin(a); //等价于&a[0]

Int \*q=end(a-1); // 等价于&a[2]

1. **Ascll码表**



**重点记忆黑色的部分**

**十进制 32 代表 空格**

**十进制 48 代表 0**

**十进制 57 代表 9**

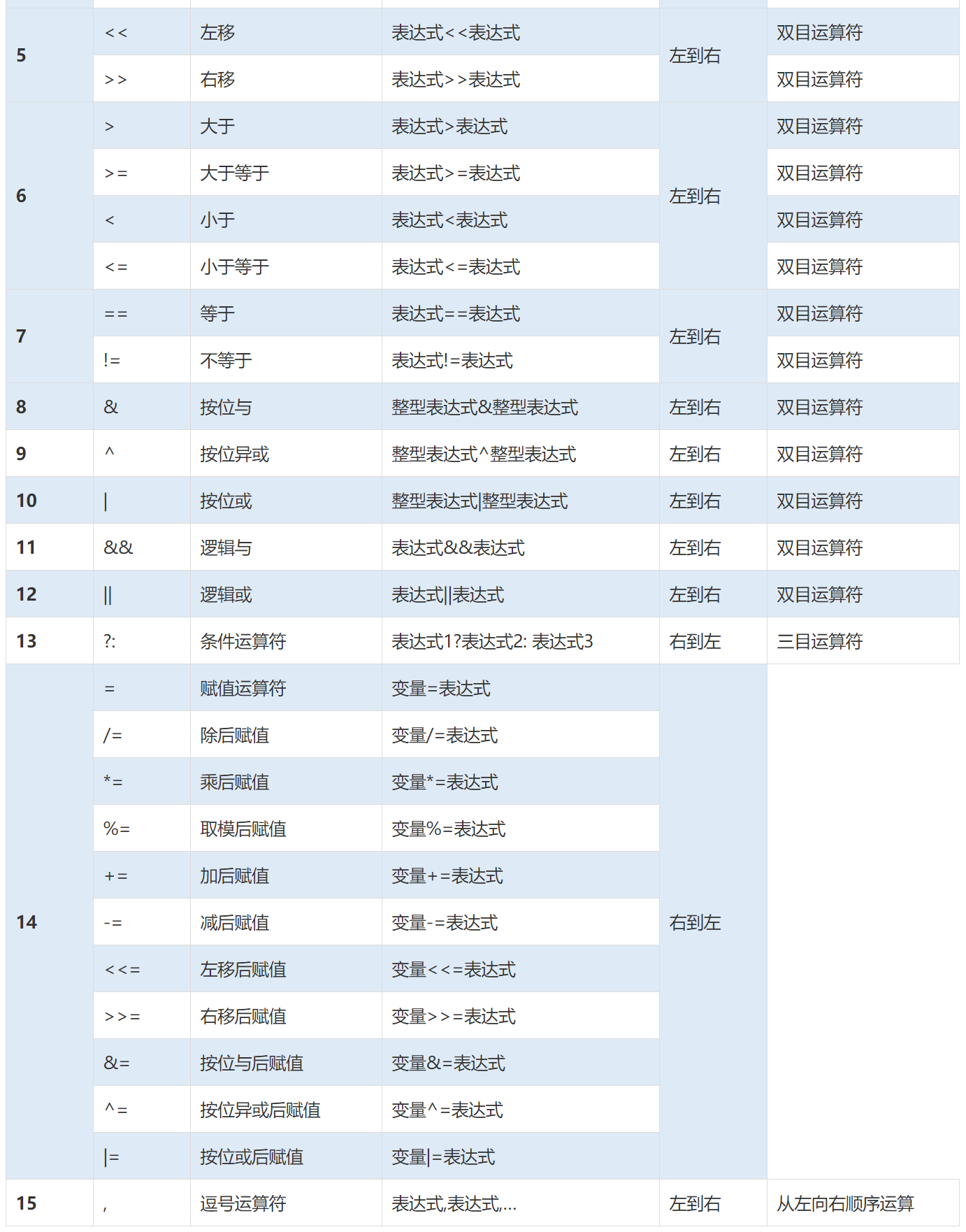
**十进制 65 代表 A**

**十进制 90 代表 Z**

**十进制 97 代表 a**

**十进制 122 代表 z**

1. Ascll码表

****