|  |  |
| --- | --- |
| C语言 | 摘要  [通过迷人的摘要吸引您的读者。它通常是文件的简短摘要。当您准备好添加内容时，只需单击此处并开始键入。]  3316065997@qq.com  C |

目录

[常见错误及重要知识 3](#_Toc134208506)

[变量 4](#_Toc134208507)

[Switch—case语句（循环控制） 0](#_Toc134208508)

[逻辑条件的判断 1](#_Toc134208509)

[基础数据类型 2](#_Toc134208510)

[Struct与Union结构体变量 3](#_Toc134208511)

[Enum枚举结构体变量 5](#_Toc134208512)

[链表 6](#_Toc134208513)

[整数相关 8](#_Toc134208514)

[double与float 9](#_Toc134208515)

[Scanf与printf 10](#_Toc134208516)

[Scanf与缓冲区 11](#_Toc134208517)

[位操作符 12](#_Toc134208518)

[运算符 13](#_Toc134208519)

[函数 14](#_Toc134208520)

[常用函数 17](#_Toc134208521)

[数组 18](#_Toc134208522)

[字符 20](#_Toc134208523)

[指针 21](#_Toc134208524)

[指针2 22](#_Toc134208525)

[Malloc动态内存分配 23](#_Toc134208526)

[字符串初见 24](#_Toc134208527)

[字符串输入输出 25](#_Toc134208528)

[字符串函数string 26](#_Toc134208529)

[字符串数组 27](#_Toc134208530)

[文件的输入输出 28](#_Toc134208531)

[其他（符号变换器） 30](#_Toc134208532)

[#include与typedef 31](#_Toc134208533)

[自定义头文件 32](#_Toc134208534)

[Dev-c++ 33](#_Toc134208535)

[VS 34](#_Toc134208536)

# 常见错误及重要知识

1. scanf中常常忘记&,for循环结束时候i的值大一。Scanf错误的读到了下一个字符，可能是前面多了一个scanf多读入了一个字符。
2. 所有的变量都是一数字存在内存中的，故加减改变变量的值再按不同标识符输出可得到合适的结果。（Eg：字符0机内码为48，‘0’-48 %d 输出后得到数字0）
3. 用计数器cnt 时，for结束时会比实际大一但是由于重0开始数。数字是按一开始数的正常值。使用cnt应该注意在哪个位置要对应的值。
4. 素数的最快计算方法：性质1，大于5的素数总在6X的前后俩个。

性质2.和数必有一个因子小于sqrt(number)。示例见C典型

4. 定义格式中指针变量前面的“\*”，表示定义的变量的类型为指针型变量，“\*”不是指针变量的一部分。指针变量名不含\*号

指针变量保存的是地址而不是值，用解运算符\*可以得到指针指指向的值，该值可参与运算，运算可修改内存中保存的值

5.#define PI 3.1415926使程序中的所有的PI的值都是3.1415926

6.数组越界时：调试会弹出Program received signal SIGSEGV, Segmentation fault.窗口

7. 浮点数的比较尽量只用<>比较并用fabs(a-b)<0.0001比较fabs（）函数在math.h中，作用是返回一个浮点数的绝对值。

8.指针只能指向指针类型对应的变量, [Error] invalid initializer(无效的初始化，str2[6]='\0'）。

9. 二数组初始化：int a[3][3]={{0}}未标明为0，int a[3][3]={{2}}不是初始化全为2

10. 实现功能的自定义函数，要包含实现功能所需的头文件。数组名是一个指针常量

11. memset (数组名, 初始值, 数组数×类型字节大小)，memset()函数的头文件是<string.h>。可以批量初始化。

12. 函数const char \*为参数可接受char\*为实参的调用，%6g 输出保留6位有效数字。

13. 函数的实参可以是常量变量表达式，但是要有确定的值传递给形参。

14. 数组要开大一点防止数组越界，特别是字符数组要避免缓冲区溢出（即用scanf %s读入数据太多了，将多余的数据写到了不该写的地方,然后就会错的莫名其妙的，输出正确的一直WAWAWA）

15. 函数有部分数据可以进去，部分数据调试时卡在那里下不去，可能是自定义的函数出错遇到那个数据时导致循环变成死循环。

16.函数exit（）；退出程序。

17.在使用malloc函数要保证安全，

p=(line\*)malloc(sizeof(line));

if(p==NULL){

printf("内存空间不足");exit(0);

}

18.洛谷中的换行符为/r/n

# 变量

1. char是字符型变量，用单引号表示字符面量（‘a’’b’）,scanf和printf中用%c来控制输出。
2. 变量是保存数据的地方,变量的名字只能由数组字母下划线组成（不可用数字开头、含关键字）。
3. 一行可定义两个变量，以逗号隔开。Eg：int prite,cnt;
4. 输入函数scanf eg：scanf(“%d”,&price);
5. 明示常量：count AMOUNT=100;（count为修饰符意为不变的，以大写提醒是常量）
6. Double是双精度浮点数，float是单精度浮点数。
7. Printf(“ ”);输出空格 printf(“\n”)换行。

Const与#define：const定义的是常变量具有变量的性质（有类型，占用内存），#define是预处理命令在编译前已处理为一个常量，#difine PI 3.1415926使程序中的所有的PI的值都是3.1415926

1. 变量未重新赋值前值不变，可用变量名用其值来运算。（int a=c+10;(正确的只要先定义了 C 并给C一个值)）
2. 浮点数的比较尽量只用<>比较并用fabs(a-b)<0.0001比较fabs（）函数在math.h中，作用是返回一个浮点数的绝对值。

# Switch—case语句（循环控制）

1. 示例 ：switch(type )

{

case 1（常量int/char）:

printf (“”);

break;

case 2:

printf (“”);

break;

case 3:

case 4:

printf (“”);

break;

default:（都不满足条件）

printf (“”);

break;

}

1. Switch-case是基于计算的表达式type是定义在外常量int，根据这个type的值来选择case的语句。（可使用多重case标签,如上3 4对应一个printf）
2. Break；可跳过循环。
3. goto out;

语句1；

语句2；

out;

可跳过其内语句。

1. 嵌套的if-else 是有限的，规定最少有127层。栈深是有限的故递归不能无限进行。
2. Continue：continue可跳过本轮continue所在的循环。除Switch所有的循环都可以用，continue会使程序跳出剩余的部分，while和for是跳过本轮循环。
3. Switch与if else的选择：根据范围选择的if简单，根据变量类型选择无法用Switch，但是Switch生成的程序通常运行较快。

# 逻辑条件的判断

1. ！运算符的优先级很高，比乘法运算符还高，与递增运算符的优先级相同，只比圆括号的优先级低。
2. &&和||都是序列点：序列点前先计算，如有++值会相应的变化，序列点前测试失败就不再向下测试（&&）。
   1. if (number != 0 && 12/number == 2)

printf("The number is 5 or 6.\n");如果number的值是0，那么第1个子表达式为假，且不再对关系表达式求值。这样避兔了把0作为除数。许多语言都没有这种特性，知道number为0后，仍继续检查后面的条件

2.2 while (x++ <10 &&x+y<20)实际上，&&是一个序列点，这保证了在对&&右侧的表达式求值之前，已经递增了x。

# 基础数据类型

1. 整数

Char（一字节）、int（一个寄存器大小）、short、long、longlong

1. 浮点数

float ,double,long double

1. 逻辑

Bool 要用#include <stdbool.h>的头文件，之后可使用bool和true代替1,false代替1.。

1. 指针
2. 自定义
3. 用const修饰的变量“不可”修改，const离谁近谁被固定。

const int a = 5; //a = 6; // error: assignment of read-only variable ‘a’

int \*p;

p = (int \*)&a; // 这里报警可以通过强制类型转换来消除

\*p = 6;//最后const还是被修改了。

7.

# Struct与Union结构体变量

1. 定义：struct eg:struct student

{成员列表 {int number;

char name[20];

char sex;

struct student \*p;//定义了指向struct student结构体变量的指针

}变量名列表; }student1,student2;而不是另一个结构体变量

1. scanf("%d %s %c",&student1.number,student1.name,&student1.sex );//

数组名不要用&，因数组名直接是地址。

1. char\* a="avavav";//没有字符串型变量，但是可以用字符指针保存字符串。

printf("%s\n",a);//指针字符串输出，对应的是字符型指针名。

1. 结构体变量可以向普通变量一样使用，但要加上一个结构体变量名前缀,但是只能对最底层的结构体变量做（即结构体嵌套要用.到最底层才能进行运算）。Eg：student1.number=2+3;
2. 同类的结构体变量可以相互赋值，可以对结构体变量取地址。对指向结构体变量的指针解指针后⬄原来指针指向东西。（printf("%d",（\*p）.number);）
3. 结构体数组在最后定义变量时改为数组即可。
4. 为了直观和方便（\*p）.number可用p->number;代替，数组a[i]⬄(\*p)故结构体数组只可以用a[i].date

若P指向一个结构体变量下面三种表达等价：

* 1. stu.成员名（如stu.number）
  2. (\*p).成员名（如(\*p).number）
  3. P->成员名（如p->number）

1. (p++)->num 先求得p->num值，然后使p指向下一个元素。
2. 结构体变量传入函数方式：
   1. 用结构体变量成员作为实参，属于值传递。
   2. 用结构体变量作为实参，属于值传递。函数调用时要占用内存空间和时间开销大，一般不用。
   3. 用指针做实参。
3. Union:用同一段内存保存不同类型的变量，但是只能保存一个值，占的内存为最大的变量的大小。Eg：union date

{

int a;

char b;

};

10.1.在stuct中嵌套union，可以一个union保存两个不同类型的变量。输入时a.b.c

11. typedef struct{

union{

char a;

int b;

}date;

}int\_char;

main()

{ int\_char[i].date.a;

return 0;

}

1. 定义一个结构体变量时可以将类型定义为 Elemtype 后要使用时用typedef重定义即可得到合适的类型。Eg；typedef struct{

ElemType date[];

}stack;（使用时加上typedef int ElemType即可将结构体中的数据类型变为int型）

# Enum枚举结构体变量

1. 枚举是一种用户定义的数据类型，它用关键字enum以如下语法来声明：

enum 枚举类型名字（名字0, ，名字n};  
枚举类型名字通常并不真的使用，要用的是在大括号里的名字，因为它们就是就是常量符号，它们的类型是int，值则依次从0到n。

1. 如：enum colors { red, yellow, green}; （声明了一种类型，用时eg：enum colors t= red;）

就创建了三个常量，red的值是0，yellow是1，而green是2可用来遍历数组很方便。当需要一些可以排列起来的常量值时，定义枚举的意义就是给了这些常量值名字。可以指定值如enum colors { red=1, yellow, green=5};

1. 虽然枚举类型可以当作类型使用，但是实际上很少用

如果有意义上排比的名字，用枚举比const int方便

枚举比宏（macro）好，因为枚举有int类型

# 链表

1. 链表在内存上是不连续的，故需要两个域，指针域和数据域，比指针多要内存（指针域），好插入，难查找。
2. 分类：静态列表和动态列表，单向列表（尾元素指向NULL），双向列表，循环列表，单向循环列表和双向循环列表。获得一个元素就相当于获得整个列表。
3. 有头结点：头结点不保存任何元素，头结点始终是第一个元素。插入时如有头结点不用考虑在第一个之前插入的情况。
4. struct eg:struct student

{成员列表 {int number;

char name[20];

int next;

struct student \*p;//定义了指向struct student结构体变量的指针

}；只是定义了一个struct student型的变量，未实际分配空间，只有定义了变量才实际分配空间。

1. 使用typedef可对结构体变量重命名，便于书写。Eg：typedef struct LinkNode

{

int data;

struct LinkNode \*next;

}Node;

后续可以用Node代替struct LinkNode

1. 静态链表：

struct LinkNode

{

int data;

struct LinkNode \*next;

};

void test()

{

struct LinkNode node1 = {10,NULL};

struct LinkNode node2 = {20,NULL};

struct LinkNode node3 = {30,NULL};

struct LinkNode node4 = {40,NULL};

node1.next = &node2;

node2.next = &node3;

node3.next = &node4;//让一存入二的地址，二入三的地址。

//历遍链表，定义一个辅助指针变量

struct LinkNode \*p2=&node1;

while(p2){

printf("%d ",p2->data );

p2=p2->next;

}

}

1. 动态列表

#include <stdio.h>

main()

{

test();

}

struct Student \*creat()

{

struct Student \*head,\*p1,\*p2;

int n=0;

p1=p2=(struct Student)malloc(sizeof(struct Student));//开辟第一个节点

scanf("%d %d",&p1->num,&p2->score);//读入第一个数字

head=NULL;

while(p1->num!=NULL){//如果读入值不为0

n+=1;

if(n==1)head=p1;//如果是第一个数据

else p2->next=p1;//不是第一个，将第一个的地址保存在p2中

p2=p1;//保留前一个地址，第二轮时上一步将第三个地址存入第二个变量的指针中

p1=(struct Student)malloc(sizeof(struct Student));//开辟下一个节点，地址存入p1中

scanf("%d %d",&p1->num,&p2->score);//读入下一个数据

}

p2->next=NULL;//

return (head);//

}

1. 链表常见错误：
2. 引用了一个未定义的指针。
3. 使用了一个无意中修改了的指针（原因可能是有多个指针指向了同一个节点）
4. 结构体变量定义也有作用域：在main函数中定义结构体变量只可以在{ }中用，故常常定义成全局结构体。
5. 双向链表节点的删除：p->prior->next=p->next; p->next->prior=p->prior;（两个连续指向指向了后第二个节点）

# 整数相关

1. 负数的表示： 补码 如-1（char8字节）为11111111，因为11111111+00000001=0；故 -1；补码是11111111。
2. 数的范围：对于一个字节（8位）可表示为00000000~11111111即-128~127 00000000为0

11111111~10000000为-1~-128

00000001~01111111为1~127

同样-a的补码是0-a,实际是2\*\*n-a(n位数)，在机内一样看待不一样则不一样。

1. 一般默认为补码，如果想让他看作纯二进阶 在类型前加上unsigned 。
2. 整数输入输出：int或大于int{（longlong）
   1. %d : int
   2. %u: unsigned
   3. %ld: long long
   4. %lu: unsigued long long

double与float

1. float字长是32有7个有效数字，double字长是64有15个有效数字（7位以外不准）

|  |  |  |
| --- | --- | --- |
| 类型 | Scanf | Printf |
| float | %f | %f,%e |
| double | %lf | %f,%e(%e用科学计数法) |
| Long long | %lld | %lld |
| char | %c | %c |
| 字符串(char数组) | %s | %s |
| Short |  |  |

1. 输出精度：在% 与f间加上.n可指定输出小数点后n位。
2. 范围与精度：浮点数在计算时是由专门硬件部件实现的。

# Scanf与printf

1. 7.scanf输入正确会返回输入的数的个数，利用这一点可以做很多事情。

* + 1. while (scanf("%lf%d", &x, &exp) == 2)//获取两个值成功

{

xpow = power(x,exp); //处理两个值

printf("%.3g to the power %d is %.5g\n", x, exp, xpow);

printf("Enter next pair of numbers or q to quit.\n");

}

* + 1. while (scanf("%d", &x) == 2)

{ **;//跳过整数输入**  }

1. 字符串可以直接读入字符数组中，不要用& eg：scanf(“%s”,pet);
2. 打印%要用%%，因为只有一个%编译器会认为漏掉了一个转换字符。
3. 转化说明：%d==%i %p（打印指针） %s（字符串） %X(x)（无符号十六进制整数） %u（无符号数） %a(A)（十六进制数） %e(E)（e计数法） %g(G)（根据值的不同自动选择%f%e） %o(无符号八进制数)
4. 转化说明修饰符：%m.nf之间打印长度为m个字符保留n位小数。
   1. 用h表示无符号，hh表示singed char类型。
   2. 用l和ll表示long和 longlong，eg：%lud
   3. 用L表示long double，eg：%Lf
   4. 用t两个指针差值的类型，用z表示size-t类型的值。
   5. 负号表示左对齐，+表示输出显示符号，空格表示若是正为空格为负是负号。#显示数据类型，0填充长度。用\*\*标识长度eg：printf(“\*scanf \*”);out: \*scanf \*
   6. %6g 输出保留6位有效数字。

# Scanf与缓冲区

1. 缓冲区又被称为缓存，是[内存](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98&spm=1001.2101.3001.7020)空间的一部分，也就是说在内存空间中预留了一定的存储空间，这些存储空间用来缓冲输入或输出的数据，这部分存储空间就叫做缓冲区，分为输入缓冲区和输出缓冲区。（协调了高速的CPU与低速的输入输出）
2. 缓冲区刷新的条件：（scanf以删除的方式读入数据）
   1. 标准输入输出是行缓冲，其满刷新即一行满了（1024个字节）才刷新
   2. 遇到 '\n'会刷新
   3. 调用fflush()函数
   4. 程序结束
3. 缓冲区的类型

全缓冲：当填满标准I/O缓存后才进行实际的I/O操作，全缓冲的典型代表是对磁盘文件的读写。

行缓冲：当在输入和输出中遇到换行符时或满一行（1024个字节）才会刷新，即把数据从缓冲区中取出来。

不带缓冲：不进行缓冲，标准出错情况stderr是典型代表，这使得出错信息可以直接尽快地显示出来。

1. Scanf在输入时要与函数里的完全一样才可以输入（scanf("d=%d", &d);）但是整数与浮点数特殊，否则只会留在缓冲区里，等待下一个输入函数。
2. scanf对不同类型输入的处理方式（**空白符（空格符、制表符、回车符**））

5.1. scanf()函数在以a=%d形式读取数据时，不会跳过空格，换行，Tab这些空白字符，在遇到这些空白字符或其他不符合要求的数据时会读取失败。

5.2. 如果数据不符合输入格式，则scanf()函数会读取失败，不再寻找输入缓冲区中后面的数据，不再等待用户输入新的数据，(即该scanf()函数调用结束)，但不符合格式的数据依然保留在缓冲区中。

5.3.scanf()函数在以%c形式读取数据时不会跳过字符

5.4.scanf()函数在以%s形式读取数据时会忽略前面的空白字符，而且默认的分隔符是所有的空白字符。若要读取一行，可用gets()函数。字符数组至少要比字符串长一，a[100]有效合法存储空间是99.

5.5.对于整型和浮点型数据的输入，也就是说"%d"类型的输入，scanf默认的分割符是所有的空白字符(空格(space)，回车(enter)和指标符(table)都行)。也就是说如果一个scanf函数中出现scanf(“%d%d”,&x,&y)，那么用任何一个空白字符来分隔两个整数a,b的值，变量a,b都可以接收到正确的输入。另外，要注意的是，scanf对于数字输入，会忽略输入数据项前面的空白字符。如：因在于在读取第一个字符串后，缓冲区中剩余一个换行符，在读取字符串之前会跳过所有的空白字符，之后scanf会发现此时缓冲区已经为空，从而需要再次等待用户输入。

6. 缓冲区的刷新：**fflush(stdin)是全部清空，g**etchar()是清空一个字符，如果缓冲区还有多余的字符则会被正常使用。

# 位操作符

1. 按位与&：参与运算的两个值，转换成2进制，如果两个相应位均为1时，结果位才为1，否则为0.例如 6&8=0，转换成二进制00000110 & 00001000 = 00000000
2. 按位或 |：参与运算的两个值，转换成二进制，如果两个数的相应位有1，结果位就为1。例如 6 | 8=14，转换成二进制00000110 | 00001000 = 00001110
3. 按位异或 ^：参与运算的两个值，转换成二进制，如果两个数的相应为相同，为0，否则为1.例如：6^8=14，转换成2进制：00000110 ^ 00001000 = 00001110

异或应用：交换两个数的值：a=a^b;b=b^a;a=a^b;

1. 左移(<<)、右移(>>)运算符：左移是对二进制位进行移动，例如1的二进制位为00000000000000000000000000000001，左移一位后，1到了倒数第二位，末位补0，即00000000000000000000000000000010. 右移相对特殊，不同的编译器有不同的情况，具体分为两种：1.算数右移（符号位不变）2.逻辑右移（符号位补0）。
2. 按位取反 ~：取反运算符～为单目运算符，其功能是对参与运算的数的各二进位按位求反。即二进制1变0，0变1。
3. ：

# 运算符

1.sizeof eg：sizeof（double）；可计算占内存大小（单位是字节，一字节为8比特）。

2. sqrt() ; 求平方根。{要用应该加上#include <math.h>的头文件}。

2.1 rand（）；召唤一个随机数；int number = rand()%100+1{召唤一个100以内的随机数}

3.逻辑运算：3.1.结果只有0和1

3.2.&&为和 ||为或 ！为非

3.3.短路：逻辑运算从左到右计算如左边已能决定结果就不做右边了；

4.条件运算符：count=(count<20)?count-10:count+10; 即if（count<20）则count-10反之count+10；

5.优先级：（运算符是动作，算子是参与运算的东西）

|  |  |  |
| --- | --- | --- |
| 优先级 | 运算符 | 综合性 |
| 1 | （） | 从左到右 |
| 2 | ！+ - ++ -- | 从左到右(单目的+-) |
| 3 | \*\% | 从左到右 |
| 4 | + - | 从左到右 |
| 5 | < > <= >= | 从左到右 |
| 6 | == != | 从左到右 |
| 7 | && | 从左到右 |
| 8 | || | 从左到右 |
| 9 | += -= \*= \= %= | 从左到右 |

* 1. 不要做嵌入式赋值，不利于阅读易产生错误。Eg：int car=1+(b=a)

1. %是取余计算eg：352%100=52；
2. 复合赋值total += 5 ⬄total=total+5 (先算右边在用结果来和左边的来运算)

Total \*= sum⬄total=total\*sum

1. 递增递减运算符：“++” “--”是单目运算符接算子必为变量，++在前先加后用，++在后先用后加。
2. 1.0\*sum可将sum转换为浮点数；

**10.复合运算符 |=、&=** ：|=：如A|=B等价于A=A|B，即A和B先进行按位或运算，然后把结果赋A &=：如A&=B等价于A=A&B，即A和B先进行按位与运算，然后把结果赋给A。

# 函数

1. 函数const char \*为参数可接受char\*为实参的调用。

#include <stdio.h>

void sum(int begin,int end) （void为返回类型sum为函数名后（int，int）为参数表用，

{ 分开两个参数，在函数定义（）后不能加； }

int i;

int sum=0;

for(i=begin;i<=end;i++){

sum +=i;

}

printf("%d到%d的值是%d\n",begin,end,sum);

}

main()

{ sum(1,10);

sum(20,30);

sum(35,45);

return 0; 注：return可返回一个值，值也可是变量ret。

}

1. 函数调用：函数名(参数)；注：即使没有参数也需要（）如果有参数要有一定的顺序。
2. Return可返回值。也可出现多个return但为单一出口只用一个。

#include <stdio.h>

int max(int a,int b)

{

int ret=0;

if(a>b){

ret=a;

}else{

ret=b;

}

return ret; 等价于return（ret）；加括号是为了提高可读性。

}

main ()

1. 为避免冗杂通常先声明，后定义。

#include <stdio.h>

int max(int a,int b);声明函数头加**；**变成函数原型，原型中可无ab但要有类型。

main ()

{

int x;

x=max(5,3);

printf("%d\n",x);

return 0;

}

int max(int a,int b)

{

int ret=0;

if(a>b){

ret=a;

}else{

ret=b;

}

return ret;

}

1. 每一个函数有自己的一个变量空间，变量空间独立存在变量在空间中存在离开则变量消失。故函数只能传入值运算后输出值。调用函数时的参数为实际参数（常量变量表达式,但是调用时要保证有确定的值），定义的函数中的为形式参数，函数调用时将实参值赋给形参Eg：max（4,a+b）; const char\*p=add((const char\*)a,(const char\*)b,len);//只是限定了在函数中不被改变，在函数中为不可变量
2. 本地变量

本地变量是定义在块内的，可以定义在函数的块内也可以定义在语句的块内。

甚至可以随便拉一对大括号来定义变量

程序运行进入这个块之前，其中的变量不存在，离开这个块，其中的变量就消失了

块外面定义的变量在里面仍然有效

块里面定义了和外面同名的变量则掩盖了外面的

不能在一个块内定义同名的变量

本地变量不会被默认初始化

参数在进入函数的时候被初始化了 {}内为一个块

1. 没有参数时（函数不需要return，类型定义为void）
   1. void m（void）；无参数,只执行任务不返回值。定义函数的类型是返回值的类型。
   2. void m（）；参数未知，编译器会自己猜一个，故不要这样写。（函数类型为返回值的类型）
2. C语言的函数定义是相互平行的，可以嵌套调用但是不可以嵌套定义。嵌套调用（定义的函数1调用定义的函数2，还有定义的函数1内调用函数1（递归调用））是直接调用即可，无需声明（主函数中声明过就行）。
3. 数组作为函数参数：
   1. 数组具体的一个元素可以作实参不可作形参（不可能为一个数组元素分配内存，数组在内存中占一段连续的内存），形参是函数调用是临时分配内存的，函数调用时是值传递，从实参到形参，单向值传递。Eg：float sum(float[],int n);(传入数组和数组大小)。**二维数组做形参先要给出数组大小 ，数组传递时给出首地址即可。**
   2. 数组传入函数时，传入的是数组的地址（即是数组首元素的地址），max(a,N);之后可以用a[i]=0;
4. 函数模块化：非必要不使用全局变量

内聚性强：功能单一（仅通过实参—形参来与外界联系）。

耦合性弱：与其他函数影响小，eg：ab\_str[i]避免与全局变量重名冲突。

1. 用户区

|  |
| --- |
| 程序区 |
| 静态存储区 |
| 动态存储区 |

1. 局部变量（定义在函数体内的变量）：

局部自动变量（auto）：一般默认为自动变量（对数据定义指定两种属性：**数据类型**和**存储类型**）

静态局部变量：static int c=13;函数调用结束时保留原值

1. 计存器变量：register int s；（寄存器读写很快，优秀的编译器会自动将用的多的变量放到寄存器中）。
2. 全局变量：
   1. 函数体外定义在要用时声明。（extern）

main() {

int a=1,b=4,c=3;

**extern** int A,B,C;（声明，将全局变量的作用域拓展到从此开始）

printf("%d",max2(a,b,c));

return 0;

}

int A,B,C;(定义三个全局变量)

* 1. 一个C程序可以由多个源程序文件组成的，在一个文件中可以用另一个文件中定义的全局变量，可以用extern来拓展到另一个文件中。

extern A;//将其他文件中定义的全局变量A作用域拓展到本文件中。

int sum(int a,int b)

{

sum=a+b+A;

}

**Static** int A;//仅限于本文件使用的外部变量

1. 内部函数与外部函数：

**Static** 类型名 函数名（参数表）；//仅作用与本文件

**Extern** 类型名 函数名（参数表）；//可供其他文件调用，但调用前要声明。

# 常用函数

1. 字符函数：
   1. put(字符数组str)：输出字符数组
   2. gets(str):输入字符数组
   3. strcat（str1,str2）;将字符2连接到字符1之后
   4. str(n)cpy(str1,str2); 将字符2连复制字符数组1之中。
   5. strcmp(str1,str2);比较字符串1与字符串2，结果为0 - +。
   6. strlen(字符数组)；统计字符串长度。<string.h>
   7. strlwr(str1);转换为小写。

1.8 strupr(str1);转换为大写。

2. 其他函数：

2.1.rand()rand和Float()

2.2. 快速排序：常见的qsort写法：void qsort(s,n,sizeof(s[0]),cmp);，#include <stdlib.h>

第一个参数是参与排序的数组名-首元素地址（也可以写成s[i]或者\*s）。

第二个参数是参与排序的元素个数。

第三个参数是单个元素的大小，sizeof（s[0]）就获得了s[0]的元素大小。

第四个参数是一个函数，定义qsort排序规则的函数。

2.2.4int cmp(const void\*pa,const void\*pb)

{

return \*(int\*)pa-\*(int\*)pb;//升序

//return \*(int\*)pa-\*(int\*)pb;降序

}

1. memset (数组名, 初始值, 数组数×类型字节大小)，memset()函数的头文件是<string.h>。可以批量初始化。
2. exit(0);终止程序，包含在stdlib.h头文件中。
3. double log(double x);计算log以e为底的X的函数值。

double log10(double x); 计算log以e为底的X的函数值。（换底公式：logxN=logaX/logaN）(math.h)

1. `int abs(int x);（求绝对值函数，math.h）

Double fabs(double x);

Floor(double x);向下取整函数

Fmod(double x,double y);求x/y的余数。

Exp（double X）；求e的x次方的值。

Pow(double x,double y)；求x的y次方的值。

# 数组

1.数组是一种容器（放东西的东西），特点是其中所有的元素具有相同的数据类型;，一旦创建，不能改变大小，数组中的元素在内存中是连续依次排列的）a[2][3][4]两页3行4列

2.定义int number[100]; number[1]=5;(对number中第二个值赋值，数组中从0开始计)

3.编译器不会检查下标，如果下标越界访问可能导致程序有时会崩溃（调试都会崩溃）。

4. 数组的大小可用变量，即读入一个数用这个数作为数组的大小。

5. int a[10] = {  
[0]=2,.[2]=3,6,};  
用[n]在初始化数据中给出定位

没有定位的数据接在前面的位置后面其他位置的值补零

也可以不给出数组大小，让编译器算

特别适合初始数据稀疏的数组

1. 大小计算 sizeof(a)/sizeof(a[0])
2. 数组作为函数参数时，要用另一个参数传入数组的大小不能直接在【 】中给出。

Eg：int sea（int key,int a[ ],int length）

不能再用sizeof来计算数组元素个数，因为传入是函数的数组的地址。

1. 二维数组：int a[3][5];

遍历初始化数组for (i=0;i<3;i++){

for(j=0;j<5;j++){

a[i][j]=i\*j;

}

}

8.1.或者直接给出eg：int a[ ][5]={

{1,2,3,4,5},

{2,5,4,6,4},

};

列数必须给出，int a[3][3]={{2}}不是初始化全为2。

9.数组和指针

9.1.有一个数组int a[10]; a（数组名）和&a[0]都表示数组的首元素的地址。

9.2. a+2==&a[2] 表示了相同的地址。 (结合第一条)

\*(a+2)==a[2]表示了相同的值。

9.3p++表示让指针指向下一个元素，在数组中p+1是P的下一个元素的地址。

int sump(int \* start, int \* end)其中while循环可压缩成一句total += \*start++;(结合9.1)

{

int total = 0;

while (start < end)

{

total += \*start; // add value to total(把所有的值加起来)

start++; // advance pointer to next element（让指针指向下一个元素）

}

return total; }

1. 初始化：int a[3][4]={{1},{0,6},{0,0,3}};未赋值的自动化为0。
2. 形参数组的值改变会导致实参数组值的改变，因数组名传递就是地址的传递。
3. 除作为函数参数的数组名由编译器转换为指针，其余情况数组名就是数组名。
4. 数组名是地址，在内存中依次排列，编译后首地址是确定的值（数组值为地址，指针为保存地址的变量），数组引用的a[i]总是被编译为\*(a+i)来处理。（a[i]⬄\*(a+i)恒成立）
5. 字符数组保存大数字时，a[0]保存是最高位后依次递减，进位加法和乘法时要考虑从小位到大位及进位后数组是否会越界。

# 字符

1. 字符类型：char用单引号表示字符面量如’A' ‘R’
2. Char的输出入用%c
3. 混合输入：scanf(“%d %c”,&I &c)读完整数及整数后空格在读下一个。

与 scanf(“%d%c”,&I&c)读完整数直接读下一个。

1. Char是整数类型可进行计算。Eg：’a’+’A’-‘a’=A
2. 逃逸字符：不能打印出来的字符

|  |  |  |  |
| --- | --- | --- | --- |
| 字符 | 意义 | 字符 | 意义 |
| \b | 退格 | \” | 双引号 |
| \t | 到下个制表位 | \’ | 单引号 |
| \n | 换行 | \\ | 反斜杠本身 |
| \r | 回车 |  |  |

注:\t是确定一个位置让其如表格一样整齐

1. 类型转换：6.1自动类型转换：运算符两边不一致会自动转换为较大的类型。

6.2强制类型转换：6.2.1：（类型）值eg：（int）10.2 注意安全因小不能表大

6.2.2：类型转换优先级高于四则运算。

7. char \*a;//定义一个字符指针变量a

a="I love China";// 将字符串中的第一个元素的地址赋给a

1. Char \*a;字符型指针，保存的不是字符串而是字符串第一个元素的地址。
2. 要保证数组完全化为0，最好用for循环。const char\*可接受char\*型的变量，反之不行。

# 指针

1. 运算符& : 获取变量地址。分配地址时是自上而下的，先定义的反而地址较高。
2. 指针就是保存地址的变量。Eg：int \*p; 或int\* p；, 打印时用的是%p。
3. Void f(int \*p);
4. {

\*p = 34;

}

{在被调用时得到了一个变量的地址，在函数里可以通过指针访问外面的变量。}

1. 5.1指针应用1：函数返回多个值，某些值就只能通过指针返回，传人的参数实际上是需要保存带回的结果的变量。

void swap(int \*pa,int \*pb)

{

int t=\*pa;

\*pa=\*pb;

\*pb=t;

}

调用时为 swap(&a,&b);

5.2指针应用2：函数需要返回多个值用指针带回，传入的参数实际上是需要带回的变量。

5.3指针应用3：函数返回运算的状态，指针返回值，动态内存分配。

6.数组变量是特殊的指针

数组变量本身表达地址，所以int a[10]； int\*p=a， / /无需用&取地址。

但是数组的单元表达的是变量，需要用&取地址

a== &a[0]

[ ]运算符可以对数组做，也可以对指针做:

P[0]⬄a[0]

\*运算符可以对指针做，也可以对教组做：

\*a=25;  
数组变量是const的指针，所以不能被值  
• int al] <.==> int\* const a=...

7. 指针与const

7.1.指针是const，一但得到了一个地址，指针的地址不可修改。

const int\* p1=&i;

int const\* p2=&i; 判断那个被const的标志是\*在前指针不可修改。

int \*const p3=&i;

8. 指针运算：

8.1.可加减一个整数。

8.2.递增递减和指针加减。

8.3. \*p++{取出P所指的数据，完事后将P移动到下一个位置；常用于数组类的连续空间操作。}

8.4可比较eg：< >

# 指针2

1. 0地址：当然你的内存中有0地址，但是0地址通常是个不能随便碰的地址。  
   所以你的指针不应该具有0值  
   因此可以用0地址来表示特殊的事情：返回的指针是无效的，指针没有被真正初始化（先初始化为0)

NULL是一个预定定义的符号，表示0地址有的编译器不愿意你用0来表示0地址。

1. 指针类型转换：void\*表示不知道指向什么东西的指针

计算时与char\*相同（但不相通)

指针也可以转换类型 ，int \*p= &i; void\*q=(void\*)p;

这并没有改变p所指的变量的类型，而是让后人用不同的眼光通过p看它所指的变量我不再当你是int啦，我认为你就是个void!

1. int \*p=NULL定义了一个空指针；
2. 有一个数组int a[10]; a和&a[0]都表示数组的首元素的地址。
3. 定义格式中指针变量前面的“\*”，表示定义的变量的类型为指针型变量，“\*”不是指针变量的一部分。指针变量名不含\*号
4. 指针变量保存的是地址而不是值，用解运算符\*可以得到指针指指向的值，该值可参与运算，运算可修改内存中保存的值。

7.\*解引用符：可以用\*来访问指针指向的值。

8. 指针应用：动态内存分配，int\*p=(int\*)malloc(sizeof(int));//动态分配了一个整性的变量的内存，并将其地址赋给指针p。

9. void型指针：可以接受任意型的指针但是不可以，对其进行解引用和四则运算。

# Malloc动态内存分配

1. Malloc函数（头文件为#include <malloc.h>和#include <stdlib.h>）

int \*a = (int\*)malloc(n\*sizeof(int));

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

int number;

int\* a;

int i;

printf("输入数量：");

scanf("%d",&number);

// int a[number];

int\*a =(int\*)malloc(number\*sizeof(int)); 用malloc 申请了number个int大小的空间。double\*a =(double\*)malloc(number\*sizeof(double));

for (i=0;i<number; i++ ) {

scanf("%d",&a[i]);

}

for ( i=number-1; i>=0; i--){

printf("%d ", a[i]);}

free(a); 归还申请的空间，只能归还申请来的空间的首地址。

return 0;

}

* 1. malloc申请的空间大小是以字节为单位，返回结果是void\*需要将类型转换为自己需要的类型。
  2. malloc如果申请失败则返回0或NULL。
  3. free（）是配套使用的，如不归还运行速度会下降最后崩溃。

常见的free的错误：申请了没free—>长时间运行内存逐渐下降  
新手：忘了  
老手：找不到合适的free的时机  
free过了再free，地址变过了，直接去free

1. realloc重新分配动态存储空间：

函数原型：void \*realloc(void \*p,unsinger int size);

# 字符串初见

1. 以0（整数0）结尾的一串字符

0或’\0’是一样的，但是和’0’是不同

0标志字符串的结束，但它不是字符串的一部分

计算字符串长度的时候不包含这个0

字符串以数组的形式存在，以数组或指针的形式访问。更多的是以指针的形式

string.h里有很多处理字符串的函数

eg：char word[] = {‘a’,’s’,’b’,’\0’};

1. 由“”括起来的是字符串常量，常量即不可修改。 Eg：”hellow”
2. 在C语言中两个字符串中如果没有任何东西会自动连接起来，但也可在用末尾加上\另一个的开头去除“来连接。
3. C的字符串是以字符数组的形式存在的，可历遍数组。
4. 如果构造字符串用数组，处理用指针。
5. Char\*可以是字符串，但不一定是字符串。
6. 字符0与'\0'是不一样的，机内码是0和48。

# 字符串输入输出

1. scanf(“%s”,string); scanf读入一个单词（到空格，tab，回车为止），但这种是不安全的，因不知道内容的长度。
2. 安全输入：在%与s之间加上数字表示允许读入的最大字符数量。Eg：scanf(“%7s”,string);
3. 常见错误： char \*string;  
   scanf("%s", string);  
   以为char\*是字符串类型，定义了一个字符串类型的。变量string就可以直接使用了，由于没有对string初始化为0，所以不一定每次运行都出错
4. 空字符串：char buffer[100]="";  
   这是一个空的字符串，buffer[0]==’\0’   
    char buffer[]=“";  
   这个数组的长度只有1!
5. 单子符输入输出：
   1. Int putchar(int); 向标准输入中写入一个字符而不是一个int的四个， 返回类型是int返回写了几个字符，EOF（-1）表示写失败。Unix是输入Ctrl+D结束。
   2. Int getchar(void); 从标准输入中读出一个字符，返回是它读的字符也是int，int是为了返回EOF（-1）表示标准输入结束了。Win是输入Ctrl+Z结束。
6. Shell(缓冲区)：是一个程序输入输出的数据先在缓冲区，按下回车才进入程序。Putchar and getchar都是从shell中读出或写入。

# 字符串函数string

1. 头文件为#include <string.h> ,包含了多个函数（strlen strcmp strcpy strcat strchr strstr）。
2. Strlen: size-t strlen(const \*char) 返回长度，不含’\0’。

char line[]="hellow";

printf("%lu\n",strlen(line));

1. Strcmp: int strcmp(const char \*s1,const char\*s2); 比较两个字符串返回0相等 1S1较大 -1S2较大。【printf("%d\n",'a'-'A');输出结果是32，实际上是机内码之差】
2. Strcpy：char \* strcpy(char \*restrict dst, const char \*restrict srç);

把src的字符串拷贝到dst（注意是后拷贝到前面），restrict表明src和dst不重叠（C99)、返回dst为了dst能参与其他的程序来。

1. Strchr与strrchr：char \* strchr(const char \*s, int c);从左找

 char \* strrchr(const char \*s, int c);从右找

返回NULL表示没有找到，非NULL则找到。

1. 字符串中找字符串  
   char \* strstr(const char \*s1, const char \*s2);（返回查找到的首字符的地址，）

char \*strcasestr(const char \*s1, const char \*s2);忽略大小写。

1. Strlwr（str1）：将含有的小写字母转化为大写，strupr(str): 将含有的大写字母转化为小写。

# 字符串数组

1. Char \*\*a a是一个指针指向另一个指针，那一个指针指向一个字符串。
2. Char a[][] a是一二维数组包含n个字符串。
3. Int main(int argc,char const\*argv[])

Argv[0]是命令本身，当使用Unix的符号链接时反应符号链接的名字。

4. memset (数组名, 初始值, 数组数×类型字节大小)，memset()函数的头文件是<string.h>。可以批量初始化。

5. 函数const char \*为参数可接受char\*为实参的调用，%6g 输出保留6位有效数字。

6. 数组要开大一点防止数组越界，特别是字符数组要避免缓冲区溢出（即用scanf %s读入数据太多了，将多余的数据写到了不该写的地方,然后就会错的莫名其妙的，输出正确的一直WAWAWA）

# 文件的输入输出

1. 文件：数据文件（供程序运行时使用）与程序文件（.c .obj(目标文件) .exe ）
2. 输入输出称为流（数据流），dat（数据文件） bmp（图形文件）
3. 文件根据组织形式可分为ascll文件（又称文本文件txt file）与二进制文件(映像文件image file)，以01储存和以字符存储的区别。
4. 在stdio.h头文件中提供了文件的结构体变量不同编译器可能提供的结构体变量不完全相同，但是都以FILE为名。FILE \* fp定义了一个指向FILE结构体的指针。
5. 打开文件的函数 FILE \* fopen（文件名，文件使用方式）；eg：fopen（“f1”,”r”）;(打开文件名为f1的文件以“r”的方式)。带**b**打开二进制文件，不带为文本文件。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 使用方式 | 处理方式 | 打开文件不存在时 | 打开文件存在时 |
| 1 | r | 只读（文本文件） | 出错 | 正常打开 |
| 2 | w | 只写（文本文件） | 创建新文件 | 文件原内容丢失 |
| 3 | a | 追加（文本文件） | 创建新文件 | 在文件原有内容后面追加 |
| 4 | r+ | 读/写（文本文件） | 出错 | 正常打开 |
| 5 | w+ | 写/读（文本文件） | 建立新文件 | 文件原有内容丢失 |
| 6 | a+ | 读/写（文本文件） | 建立新文件 | 在文件原有内容后面追加 |
| 7 | rt  （同1） | 只读（文本文件） | 出错 | 正常打开 |
| 8 | wt  （同2） | 只写（文本文件） | 创建新文件 | 文件原内容丢失 |
| 9 | at  （同3） | 追加（文本文件） | 创建新文件 | 在文件原有内容后面追加 |
| 10 | rb | 只读（二进制文件） | 出错 | 正常打开 |
| 11 | wb | 只写（二进制文件） | 建立新文件 | 文件原有内容丢失 |
| 12 | ab | 追加（二进制文件） | 建立新文件 | 在文件原有内容后面追加 |
| 13 | rt+  （同4） | 读/写（文本文件） | 出错 | 正常打开 |
| 14 | wt+  （同5） | 写/读（文本文件） | 建立新文件 | 文件原有内容丢失 |
| 15 | at+  （同6） | 读/追加（文本文件） | 建立新文件 | 在文件原有内容后面追加 |
| 16 | rb+ | 读/写（二进制文件） | 出错 | 正常打开 |
| 17 | wb+ | 写/读（二进制文件） | 建立新文件 | 文件原有内容丢失 |
| 18 | ab+ | 读/追加（二进制文件） | 建立新文件 | 在文件原有内容后面追加 |

1. 关闭数据文件int fclose（fp）(关闭是撤销文件缓冲区和文件信息区，会先将缓冲区的数据输出到磁盘)，成功返回0否则返回-1
2. 顺序读取数据
   1. Fgetc(fp)读取一个字符，失败返回-1，fgetc⬄getc因为在stdio.h中已用define定义《#define getc(fp) fgetc(fp) 》
   2. Fputc(fp)输出一个字符，失败返回-1，fputc⬄putc

Scanf(“%s”,filename);//读入文件名

Getchar();//消除最后的换行符

Exit(0);//终止程序

* 1. feof(fp)有两个返回值:如果遇到文件结束函数feof（fp）的值为非零值否则为0。
  2. 字符串的读写：
     1. Char \* fgets(char \* str, int n, fp);//从 fp中读出长为n-1的字符串加上\0，存放到str中，成功返回str的地址，失败NULL,若在读完n-1个字符前遇到换行，EOF，读入即结束，并将\n作为一个字符读入。
     2. Int fputs(char \*str,FILE\*p);将字符串（不含最后的\0）输出到fp指向的文件中。成功0失败EOF
     3. Fprintf( fp,格式字符，输出列表)；//但是两者都要将机内码和二进制

Fscanf(fp, 格式字符，输入列表);//数据相互转化，经常数据转化不可用

* + 1. size\_t Fread（buffer，size，count，fp）;buffer存放读入数据的地址，size字节数，count数据项，fp文件指针，返回读到的基本单元个数。 Fwrite（buffer，size，count，fp）;buffer存放读入数据的地址，size字节数，count数据项，fp文件指针。

1. 随机读写数据（系统为每一个文件设置了一个文件标记）
   1. Void Rewind（）将文件标记指向文件开头，
   2. Int Fseek(fp,位移量，起始点);eg:fseek(fp,50L,1);
   3. Long Ftell( fp );返回当前文件标记位置。Eg：a = ftell( fp1 );
   4. Int Ferror(fp);//检查输入输出函数是否出错，无错返回0；
   5. Clearer(fp);重置ferror函数的值，否则文件输入输出函数出错，时的值一直在。重置后才可进行下一次检测。

# 其他（符号变换器）

1. 打印空格printf(“ ”)。If（islower（ch））{如果是小写字母才处理}
2. If-else 中可单独使用if（条件）{ 语句 } 。
3. 通常使用count（简写cnt） 作为计数器。
4. 条件&&为和，||为或。
5. 八进制与十六进制：以0开始是8进制，以0 x开始是16进制。%o进制用于8进制printf，%x进制用于16进制printf。（ox大小写对应数值中的字母的大小写）
6. While（条件）{循环}；
7. Do{ 循环} while(条件)；先做后判断。
8. If（）{ } else{ }

级联为if（）{ }

Else if（）{ }

1. For(i=1;i<=23;i++){ 循环 } 一初始化二判断三递增。
2. 可用while(){a=-a}来实现正负号轮换，a=-1.

# #include与typedef

1. 使用尖括号< >和双引号" "的区别在于头文件的搜索路径不同：

使用尖括号< >，编译器会到系统路径下查找头文件；而使用双引号" "，编译器首先在当前目录下查找头文件，如果没有找到，再到系统路径下查找。

1. Typedef可以重命名类型名，简而言之就是按定义方式定义变量将变量名换成新名字在前面加上typedef 即可。eg：typedef int ABS(将int重命名为ABS)，eg：typede struct node{int date ;struct node\*next}NODE
2. 使用typedef可以提高程序的可移植性，如将以int为4字节的程序移植到int为2字节的机器上，通常要将所以的int替换为long，只要将typedef int integer换为typedef long integer即可。
3. #include 的处理过程很简单，就是将头文件的内容插入到该命令所在的位置，从而把头文件和当前源文件连接成一个源文件，这与复制粘贴的效果相同。
4. #include <point.h>包含了点坐标。
5. #include “max.c”将max.c源文件链接到当前源文件。

自定义头文件：

1.#ifndef **\_**MYHEAD\_H（自定义头文件\_是占位符，格式需要）

#define \_MYHEAD\_H

int max(int a,int b);

#endif

2.#include <stdio.h>主函数

#include "myhead.h"

main()

{

printf("hi\n");

int a=9,b=8;

printf("%d",max(a,b));

return 0;

}

3.#include <stdio.h>实现功能的自定义函数，要包含实现功能所需的头文件。

int max(int a,int b)

{

if(a<b){

a=b;

}

return a;

}

# Dev-c­­++

1. 选中变量，右击加A可设为查看。
2. Ctry+滑轮可改变字体大小。
3. //注释一行 /\* \*/注释中间的一切。
4. 代码的调试：4.1. 用特殊值检验如0 1 -1
   1. 在合适的位置插入printf输出中间值。
   2. 或printf (“hr \n”)证明运行至此。
5. 四种项目：Windows Application 是窗体应用，运行的时候显示窗体  
    Console Application 是控制台应用,运行的时候是显示控制台  
    Staic Library是静态链接库，用来创建一个静态库。  
    DLL是动态链接库，用来创建一个动态库。

Empty Project是空项目，空项目里面是完全空的，所有头文件，库文件，源文件等等都要自己写。

库⽂件是将⼀系列的源⽂件进⾏编译打包，形成的⼆进制⽂件包，其中封装着函数接⼝，在编程中可以由其它源⽂件调⽤。库⽂件可分为静态库和动态库两种。静态库与动态库的区别静态库的代码在编译的过程中已经载入到可执行文件中。动态库的代码在可执行程序运行时才载入内存，在编译过程中仅简单的引用。静态库链接的时候把库直接加载到程序中。动态库链接的时候，它只是保留接口，将动态库与程序代码独立，这样就可以提高代码的可复用度和降低程序的耦合度。静态库在程序编译时会被连接到目标代码中，程序运行时将不再需要该静态库。动态库在程序编译时并不会被连接到目标代码中，而是在程序运行是才被载入，因此在程序运行时还需要动态库存在。

# VS

1. 在vs 中，打开项目 -> 属性 -> C/C++ -> 预处理器 -> 预处理器定义中添加 \_CRT\_SECURE\_NO\_WARNINGS 这个预定义。
2. 一个新建项目下往往会有四个文件夹：  
   外部依赖项：头文件依赖的头文件，是外部库而不是自己定义的。  
   头文件：.h文件。函数声明、宏定义、结构体定义等。  
   源文件：.c或.cpp文件，函数实现，变量定义。新建项时在这里右键添加。  
   资源文件：放一些会用到的图片。