知识整理

- 结构化程序设计：分而治之、自顶向下
- 流程控制
- 语句级控制：顺序结构、分支结构、循环结构
- 单位级控制：函数
- 算法描述：自然语言、伪代码、流程图

在C语言中，一行可以书写多个语句。T

在C语言中，一个语句可以书写在不同行上。T

if (表达式)
　语句1
Else　　其中的语句1、语句2只能是一条语句。
　语句2

while语句的一般形式如下，其中的循环体语句只能是一条语句。
while (表达式)
　循环体语句

程序与数据一样，共同存储在存储器中。当程序要运行时，当前准备运行的指令从内存被调入CPU中，由CPU处理这条指令。这种将程序与数据共同存储的思想就是目前绝大多数计算机采用的（　冯•诺伊曼　）模型的存储程序概念。

下列(　c　)不是C语言所表现出来的不足之处。
A.数据类型检查不严格
B.表达式容易出现二义性
C.可移植性差
D.不能自动检查数据越界

C程序中定义的变量，代表内存中的一个存储单元。T

编译→语法错误，调试→逻辑错误

若变量定义为double x;，则x % 2是符合C语言语法的表达式。F

为了检查以下if-else语句的两个分支是否正确，至少需要设计3组测试用例，即x的取值至少有三组（小于15的数、15和大于15的数）。T
```
if (x <= 15){
  y = 4 * x / 3;
} else{
  y = 2.5 * x - 10.5;
}
```
对于判断条件`x <= a`，需要测试的核心用例包括：
- 小于 a 的数（如a-1）：验证左区间逻辑；
- 等于 a 的数（如a）：验证临界值逻辑；
- 大于 a 的数（如a+1）：验证右区间逻辑。

`for (int I = 1; i > 0; i++) {}`
整数溢出导致循环变量从正数变为负数，从而让循环条件为假退出
```
for (int i = 0; i > 0; i++) {
  // 不是死循环：循环体和 i++ 都不会执行
}
// 情况1：条件写反（i >= 0）
for (int i = 0; i >= 0; i++) {
  // 死循环：i从0开始不断递增，永远满足i >= 0
}

// 情况2：条件恒为真（如1）
for (int i = 0; 1; i++) {
  // 死循环：条件永远为真
}

for (表达式1; 表达式2; 表达式3) {
  循环体语句;
}
```
- **执行表达式 1（初始化）**：仅在循环开始时执行一次。
- **执行表达式 2（条件判断）**：
  - 若结果为**真（非 0）**：执行循环体语句 → 执行表达式 3（递增 / 递减）→ 回到**表达式 2**再次判断；
  - 若结果为**假（0）**：直接退出循环，后续步骤都不执行。

如果变量已经正确定义，则表达式fahr ++ 与fahr + 1等价。 F

C程序中，用一对大括号{}括起来的多条语句称为复合语句，复合语句在语法上被认为是一条语句。

```
int i;
double product;
product = 0;//应该是1
for (i = 1; i <= 20; i++){
  product = product * i;
}

int i;
double sum;
sum = 0;
for (i = 2; i <= 4; i = i + 2){
  sum = sum + 1/i;
}//sum不是0.75而是0

int i, sum;
for (i = 1; i <= 10; i++){
  sum = 0;
  sum = sum + i;
}//sum=10
```

C语言中，数字字符'1'的值（ASCII码）就是数字1。F（49）
C语言中，大小写字母'A'，'B'，'C'，…，'Z'，'a'，'b'，'c'，…，'z'的ASCII码按升序连续排列。 F

执行语句putchar('5');后，在屏幕上显示的输出结果是'5'。F
`putchar()`的参数可以是**字符常量（如`'5'`）**、字符变量，或对应字符的 ASCII 码值（如`'5'`的 ASCII 码是 83，`putchar(83)`也会输出5）

C语言中，以下（ ）是合法的字符常量。

A.'A'B.'z'C.'0'D.'5'E."a"
ABCD单引号，一个字符。字符可以是 ASCII 码表中的**可打印字符**（字母、数字、标点符号等）或**不可打印的转义字符**（如`'\t'`、`'\0'`等）　s是 ASCII 可打印字符），符合字符常量规则

```
flag = 1;
denominator = 1;
item = 1.0;
pi = 0;
while(fabs(item) >= 0.0001){
    item = flag * 1.0 / denominator;
    pi = pi + item;
    flag = -flag;
    denominator = denominator + 2;
}//while循环结束item的绝对值小于0.0001，累加到pi的最后一项item的值满足item的绝对值小于0.0001
```

- **自动变量（局部无 static）**：未赋值时是**随机垃圾值**，不是 0，使用前必须手动赋值；
- **全局变量 / 静态变量**：未赋值时会被编译器**自动初始化为 0**；

实参与其对应的形参各占用独立的存储单元

在函数调用Func(exp1，exp2+exp3，exp4*exp5)中，实参的数量是3。

函数是一个完成特定工作的独立程序模块，包括库函数和自定义函数

08是正确的整型常量。 F
- 以0开头的数字序列，会被编译器识别为**八进制数**；
- 八进制数的有效数字只有0~7

`!x<==>x==0`

`~(~2<<1)==5`

设字符型变量x的值是064，表达式 ~ x ^ x << 2 & x 的值=0333
位运算优先级（从高到低）：~（按位取反）> <<（左移）> &（按位与）> ^（按位异或）> |（按位或）。

以下定义了一个一维数组str，该数组可以存放81个字符型数据。T
char str[81];
认为`char str[81];`只能存 80 个字符 → 错误，仅存储字符串时受`'\0'`限制，存储普通字符可存 81 个

对二维数组初始化时，经常采用如下分行赋初值的方式，即在定义数组时，把初值表k中的数据依次赋给第k行的元素。T
类型名　数组名[行长度][列长度] = {{初值表0}, … {初值表k}, ……};

选项（ ）与以下字符数组定义等价。
static char s[6] = {'H', 'a', 'p', 'p', 'y', '\0'};
A.static char s[6] = {'H', 'a', 'p', 'p', 'y',};字符数组初始化时，若指定数组长度且初值表长度 < 数组长度，**未赋值的元素会自动填充`'\0'`**
B.static char s[6] ="Happy";
C.static char s[6] ={"Happy"};
D.static char s[6] = {'H', 'a', 'p', 'p', 'y', 0};

```
char array [ ]="China";
数组占用字节数：6sizeof
字符串有效长度：5strlen
```

可以在赋值语句中通过赋值运算符 "=" 对字符数组整体赋值F
```
// 错误示例
char arr[10];
Arr = "hello"; // 直接赋值数组名，非法
```
正确的赋值方式只能是逐个字符赋值，或使用`strcpy`函数

指针

以下选项中，对基本类型相同的指针变量不能进行运算的运算符是（+）。
优先级：()（括号）> ++（自增）> *（解引用）；**后置自增++的优先级高于前置自增++和解引用*。**
后置自增++是**右结合**，前置自增++和解引用*是**左结合**；
(*p)++:解引用后自增（修改指针指向的变量值，后置）
++*p :解引用前自增（修改指针指向的变量值，前置）
"p++ =="(p++):先执行p++（指针的后置自增）：先返回p的**原值**（指向a的地址），再将p的地址值加 1（指向下一个内存地址，如a的地址是 0x100，则p变为 0x104，因 int 占 4 字节）
字符串常量在内存中的存放位置由系统自动安排。

```
char s[ ]="china";
char *p;
p = s;
```
数组s的长度和p所指向的字符串长度相等F
- 数组s的长度：6（包含`\0`，`sizeof(s)=6`）；
- P指向的字符串长度：5（`strlen(p)=5`，不包含`\0`）；

```
char s[ ] = "language", *p = s;

while( *p++ != '\u') {
    printf("%c", *p – 'a' + 'A');
}输出 ANGU

  char str[]= "abc\0def\0ghi", *p = str;
    printf("%s", p+5) ;输出ef

char s[20]= "abcd" ;
char *sp = s ;
puts(strcat(sp+1, "ABCD"+1)) ;输出bcdBCD
```

结构

```
Struct ex {//关键字 //结构类型名
  int x;   // 结构体成员1
  float y;  // 结构体成员2
  char z ;  // 结构体成员3
} example;  //结构变量名 定义结构体类型的同时，声明了一个名为example的结构体变量
```
C语言把结构的定义看成一条语句
1、单独定义：struct ex{}; struct ex stu;
2、混合定义：struct ex{}stu;
3、无类型名定义：struct{}stu;

一个结构类型变量所占的内存空间是其各个成员所占内存空间之和。sizeof（struct student）/sizeof（stu）
不同类型的结构变量之间也可以直接赋值。 F

C语言中结构类型变量在程序执行期间（ ）。
A.
所有成员一直驻留在内存中

结构数组
结构数组是结构与数组的结合体，与普通数组的不同之处在于每个数组元素都是一个结构类型的数据，包括多个成员项。 T
结构指针
```
struct student stu={},*p;
p=&stu;
```
成员访问：单个结构体变量 std.name　p->name　(*p).name
　　结构体数组　std[0].name　(p+i)->name　(*(p+i)).name
　　(std.name❌ (*p)->name❌)
　　(如struct student* p）只能指向struct student类型的变量，不能指向结构体成员（如int*类型的&stu[0].age）。
输入输出：scanf("%d",&std[0].age) scanf("%s",std[0].name) scanf("%d",&(p->age)scanf("%d", &(*p).age)
->优先级高–> 优先级 > 前置++ > * 解引用

```
struct stu{
  int x;
  int *y;
} *p;
int dt[4] = {10, 20, 30, 40};
struct stu a[4] = {50, &dt[0], 60, &dt[1], 70, &dt[2], 80, &dt[3]};

int main( )
{
  p=a;
  printf("%d,", ++p->x);等价于 ++(p->x)
  printf("%d,", (++p)->x);
  printf("%d", ++(*p->y));p指向a[1]++(*(p->y))


  return 0;
}
struct {
  int len;      ++p->str中的++加在str上　(*p)->str++中的++语法错误
  char *str;
} *p;
struct {
  int a; int *b;
} s[4], *p;
int i, n = 1;
for(i = 0; i < 4; i++){
    s[i].a = n;
    s[i].b = &s[i].a;
    n = n + 2;
}
p = &s[0];
printf("%d\n", ++*p->b);
p++;
printf("%d,%d\n", (++p)->a, (p++)->a);  /* 输出数据之间没有空格分隔；按从右向左的顺序求解表达式 */
```

- 第一个 printf：修改 s [0].a 为 2，输出 2；
- p++：p 指向 s [1]；
- 第二个 printf：先算右侧 (p++)->a（取 3，p→s [2]），再算左侧 (++p)->a（p→s [3]，取 7），输出 7,3；
- ++p->a等于 ++（p->a）

函数与程序结构

函数的定义不可以嵌套，但函数的调用可以嵌套
以下描述错误的是(D)。
A.函数调用可以出现在执行语句中
B.函数调用可以出现在一个表达式中
C.函数调用可以作为一个函数的实参
D.函数调用可以作为一个函数的形参

以下函数调用中，实参的个数为（ 2 ）。
func((exp1,exp2),(exp3,exp4,exp5));

凡是函数中未指定存储类别的局部变量，其隐含的存储类型为(A)。
A.自动（auto）
B.静态（static）
C.外部（extern）
D.寄存器（register）

在一个C源程序文件中，若要定义一个只允许本源文件中所有函数使用的全局变量，则该变量需要使用的存储类别是D。
A.extern
B.register
C.auto
D.static

将一个函数说明为static后，该函数将 (B)。
A.既能被同一源文件中的函数调用，也能被其他源文件中的函数调用
B.只能被同一源文件中的函数调用，不能被其他源文件中的函数调用
C.只能被其他源文件中的函数调用，不能被同一源文件中的函数调用
D.既不能被同一源文件中的函数调用，也不能被其他源文件中的函数调用

以下正确的叙述是（A ）。
A.宏替换不占用运行时间，只占用编译时间
B.在程序的一行中可以出现多个有效的预处理命令行❌**//每个预处理命令必须独占一行**
C.使用带参数宏定义时，参数的类型应与宏定义时一致❌
D.宏定义不能出现在函数内部❌

C语言的编译系统对宏命令的处理（B ）。
A.在程序运行时进行的
B.在对源程序中其他语句正式编译之前进行的
C.在程序连接时进行的
D.和C程序中的其他语句同时进行编译的

下面说法中正确的是（）。
A.若全局变量仅在单个C文件中访问，则可以将这个变量修改为静态全局变量，以降低模块间的耦合度
B.若全局变量仅由单个函数访问，则可以将这个变量改为该函数的静态局部变量，以降低模块间的耦合度
C.设计和使用访问动态全局变量、静态全局变量、静态局部变量的函数时，需要考虑变量生命周期的是**普通局部变量（栈区）、动态分配变量（堆区，malloc/free）**，而非静态存储类变量
D.静态全局变量使用过多，可那会导致动态存储区（堆栈）溢出 //栈溢出通常是局部变量过多、函数递归过深；堆溢出是 malloc 申请过多内存

普通全局变量	整个程序（所有.c 文件）	程序运行全程	静态存储区（.data/.bss）
静态全局变量	仅定义它的.c 文件	程序运行全程	静态存储区
静态局部变量	仅定义它的函数内部	程序运行全程（仅第一次初始化）	静态存储区
普通局部变量	仅函数 / 代码块内部	函数调用时创建，结束销毁	栈区（堆栈）

C语言的编译预处理功能包括文件包含、宏定义、条件编译

递归：1递归出口2递归式子P249
递归调用算法与相同功能的非递归算法相比，主要问题在于重复计算太多，而且调用本身需要分配额外的空间、传递数据和控制，所以时间和空间开销通常都比较大。
递归并不节省存储器的开销，因为递归调用过程中必须在某个地方维护一个存储处理值的栈。
分治法
宏定义
#define 宏名 宏定义字符串
（#表示在编译预处理起作用，不是真正c语句，不用逗号）
宏名常用大写字母（不是必须），中间不能有空格，宏名和宏定义字符串空格连接

指针进阶

命令名 命令行参数（也可以没有参数）
`int main(int arc, char *argv[]) argc`接受命令行参数个数（包括命令）`argv`接收参数（包括命令）

系统根据输入的命令行参数的数量和长度自动分配存储空间
`int *p[5];`
指针数组（重点）
P 是一个数组，包含 5 个元素，每个元素是 `int*` 类型（指向 int 的指针）。
`int (*p)[5];`
数组指针
P 是一个指针，指向一个 “包含 5 个 int 类型元素的数组”。

`*pt[0] = *(pt[0])`
对于以下程序，能够正确表示二维数组 t 的元素地址的表达式是（　）。
```
int main(void)
{
  int k, t[3][2], *pt[3];

  for ( k = 0; k < 3; k++) {
      pt[k] = t[k];
      }

  return 0;
}
```
A.&t[3][2]
B.*pt[0]/*t[0][0]的值
C.*(pt+1)/t[1][0]的地址 pt+1指t[1]的地址
D.&pt[2]

a[i][j]*(a[i]+j)*(*(a+i)+j)

```
int a[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
int (*num)[4] = a; // 第一步：初始化num，让它指向a的第一行（a是a[0]的地址，类型匹配）

// num[1] 等价于 *(num+1)，指向a的第二行（完整的4int数组）
printf("num[1] 指向的地址：%p\n", num[1]);  // 输出a[1]的地址（即&a[1][0]）
printf("num[1][2] = %d\n", num[1][2]);    // 输出7（a[1][2]，这才是正确访问）
```

小测错题补充

Else总是和之前与其最近的不带else的if配对
Sizeof（int）为2，int类型数据可以表示的最大值为
$2^{15}-1$
- 2 字节 = 16 位二进制
- 最高位是符号位（0 表示正数，1 表示负数），剩余 15 位表示数值。

最大值计算逻辑
正数的补码与原码一致，15 位数值位全部为 1 时，数值达到最大

if(I<<10)
For语句里面++i和i++结果一样
int k，flag；

if (k=0) flag=0；
else flag=1；//k的值为0，flag的值为1

if (k=1) flag=0；
else flag=1；//k的值为0，flag的值为0

int I=1;
while(i>0){i++;printf("%d",i);//不会进入死循环

char a；
a='\65';T
a=65;T
a=6+'A';T
a="A";F

数组名作为参数，函数调用时是将实参数组的首地址传给形参

Char str[10];str="string";
char str[7]={'s','t','r','i','n','g'};

-k++<==>-(k++)

～(a^-a)<=>0

Int a[3]={1,2,3},b[2],*p;

 a[3]={1,2,3},b[2],*p;
p=a+1;
b=p;数组名是常量不能被赋值

for(; ;) 死循环

void fun(int x) // 形参x，局部于fun函数
{
  static int y; // 静态局部变量y，作用域仅限fun函数
  // ...
  return;
}

int z; // 全局变量z，整个程序都可访问
void main()
{
  int a,b; // 局部变量a、b，仅限main函数
  fun(a);
  // ...
}
- **直接访问就是利用变量的地址直接访问。**F
- **直接访问（直接寻址）**：直接通过**变量名**访问变量的值，编译器会自动把变量名映射到对应的内存地址，无需手动操作地址；
- **间接访问（间接寻址）**：才是通过**变量的地址**（指针）来访问变量的值，需要手动解引用指针。

结构体成员的类型必须是基本数据类型。F
还能是数组、指针、其他结构体，甚至指向自身的结构体指针

当变量的存储类型缺省时，系统能默认变量的存储类型为auto类型，
分配在静态区
- **存储类型缺省的局部变量**：默认是auto类型（自动变量），但存储在**栈区**（而非静态区）；
- **存储类型缺省的全局变量**：默认是extern类型（外部变量），存储在**静态区**；

下列选项中正确的语句组是（）
A:
char s[8]; s={"beijing";
B:
char *s=NULL; s={"beijing";s 是数组名（地址常量），不能被赋值
C:
char s[8]; s="beijing";
D:
char *s=NULL; s="beijing";s = "beijing"; 是给指针变量 s 赋值，让它指向字符串常量 "beijing" 的首地址（指针是变量，可被赋值）

函数的声明
已知函数的定义如下：
```
double F(char letter, int step)
{
    return ((toupper(letter) - 'A' + step % 26 + 26) % 26 + 1) / 26.0;
}
```
下列函数声明中，_____ 是正确的。
- A. double F(char, int);
- B. F(letter, key);
- C. double F(char letter, int key);
- D. double F(char s, int k);

下列程序的输出结果为：g=4,g=3,k=6　1 分。
```
#include <stdio.h>
int fib(int g)
{
    switch(g){
        case 0: return 0;
        case 1:
        case 2: return 2;
    }
    printf("g=%d,", g);    /* 中间没有空格 */
    return fib(g-1) + fib(g-2);
}

int main(void)
{
    int k;

    k = fib(4);
    printf("k=%d\n", k);    /* 中间没有空格 */

    return 0;
}
```

当字符型变量ch取何值（或何范围）时输出"YES"
```
char ch;
scanf("%c",&ch);
if(ch<='A'&&ch>='Z')
printf("NO");
else printf("YES");
```
- A. 大写字母A~Z
- B. 任何字符
- C. 除大写字母以外
- D. 大写字母B~Y

If条件始终为假

下面程序段的运行结果是
"2*4*4*8*"　1 分
```
char a[]= "12345678", *p;
int i = 0 ;

p = a ;
while(*p != '\0'){
  if(i % 2 == 0){
    *p='*' ;
  }
  p++;
  i++ ;
}
puts(a);
```

下面程序段的运行结果是　6 6 7　1 分。
```
int i, j, k = 5, *p;

p = &k;
i = ++(*p);
j = (*p)++;
printf("%d %d %d", i, j, k) ;   /* 输出的数字之间有一个空格 */
```
只能向指定文件写入一个字符的函数是
fputc | fputc()　1 分　。

```
int x = 5, y = 7;
void swap ( )
{
      int z ;

      z = x ;  x = y ;  y = z ;
}
int main(void)
{
      int x = 3, y = 8;
      swap ( ) ;
      printf ("%d,%d \n", x , y ) ;

      return 0 ;
}
```
输出 3 8

请写出程序段的输出结果：
#1*1*24*4　3 分
```
int n=0;
while(n<=4)
switch(n)
{
case 0:printf("#");
default:printf("%d",++n);
case 2:printf("*");
case 1:printf("%d",n++);
}
```

对于定义 `char str[] = "abc\000def\n"`（注：其中0为数字零），求字符串str的长度len和数组str的大小size，len和size分别是：
- A. 12，13
- B. 3，11
- C. 3，9
- D. 3，4

若有以下定义，
```
struct data
{
    int x;
    int y;
}data1;
int *p;
```
则使p指向data1中的成员x，正确的赋值语句是（）。
- A. p=&a;
- B. p=&(data1.x)
- C. p=data1.x;
- D. *p=data1.x;

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

int main(void)
{
    char str[] = "abc\000def\n";

    printf("sizeof(str)=%d\n", sizeof(str));
    printf("strlen(str)=%d\n", strlen(str));

    return 0;
}
```
str = "abc\000def\n"
str[0] = 97 'a'
str[1] = 98 'b'
str[2] = 99 'c'
str[3] = 0 '\000'
str[4] = 100 'd'
str[5] = 101 'e'
str[6] = 102 'f'
str[7] = 10 '\n'
str[8] = 0 '\000'

1. 定长字符数组
```
char str[10] = "abc";
strlen(str); // 3 (统计到'a'/'b'/'c'，到 '\0' 停止)
sizeof(str); // 10 (数组总内存，和实际存储内容无关)
```
2. 不定长字符数组
```
char str[] = "abc";
strlen(str); // 3 (不计末尾自动补的 '\0')
sizeof(str); // 4 (包括 'a'/'b'/'c'/'\0' 共4字节)
```
3. 无 '\0' 的 char 数组
```
char str[3] = {'a','b','c'};
strlen(str); // 随机值 (找不到 '\0'，越界读取)
sizeof(str); // 3 (数组固定内存，不管 '\0' 有没有)
```

1. 字符串长度（strlen）：strlen会从字符串起始位置计数，遇到第一个\0（空字符）就停止。题目中"abc\000def\n"里的\000是八进制转义的空字符，因此strlen(str)只统计a、b、c，结果为3。

2. 数组大小（sizeof）：sizeof计算整个数组的字节数，需包含所有字符（包括空字符和转义符）。数组内的字符依次为：a、b、c、\0、d、e、f、\n、\0（字符串末尾自动补的空字符），共9个字符，每个char占1字节，因此sizeof(str)结果为9。

下面的程序段输出是（）。
```
short int a;
int b = 65536;

a = b;
printf("%d\n", a);
```
- A. 65536
- B. 0
- C. -1
- D. 1

若变量已正确定义，以下while循环正常结束时，累加到pi的最后一项item的值满足
```
flag = 1;
denominator = 1;
item = 1.0;
pi = 0;
while(fabs(item) >= 0.0001){
  item = flag * 1.0 / denominator;
  pi = pi + item;
  flag = -flag;
  denominator = denominator + 2;
}
```
- A. item的绝对值小于0.0001

实验题

```
#include<stdio.h>
int main(void){
    char ch;
    int word=0;
    while((ch=getchar())!='\n'){
        if(ch==' '){word=0;}
        else if(word==0){
            word=1;//如果单词首字母为大写则
            //告诉我第二个字母也要大写
            if(ch>='a'&&ch<='z'){ch=ch-'a'+'A';}
        }
        putchar(ch);
    }
    return 0;
}
```

（六）常用算法
以下算法针对本大纲中列出的各种数据结构
1.分类（排序）算法
冒泡、选择、插入
2.检索（查找）算法
(1) 无序数据序列的查找（见遍历算法）
(2) 有序数据序列的查找：二分法
3.遍历算法
(1) 一维数组和二维数组的遍历
(2) 文件的遍历
4.简单的数值计算方法：如多项式函数的计算
5.其它基本算法：如进制转换

```
    for(i=0;i<n-1;i++){
        int index=i;
        for(int j=i+1;j<n;j++){
            if(stu[index].birth>stu[j].birth){
                index=j;
            }
        }
        temp=stu[index];
        stu[index]=stu[i];
        stu[i]=temp;
    }
```
顺序排序：index！！！！否则就是拿i比较

```
void bubble(int a[], int n){
    int i,j,t;
    for(i=0;i<n;i++){
        for(j=0;j<n-i;j++){
            if(a[j]>a[j+1]){
                t=a[j+1];
                a[j+1]=a[j];
                a[j]=t;
            }
        }
    }
}
```
冒泡排序

```
#include<stdio.h>
int main(void){
    int n,i;
    scanf("%d",&n);
    int *students=(int*)malloc(n* sizeof(int));
    for(i=0;i<n;i++){
        scanf("%d",&students[i]);
    }
```
不确定n，动态分配内存

```
#include<stdio.h>
int main(void){
    int a,b;
    scanf("%d%d",&a,&b);
    double s=(a-b)*100.0/b;
    if(s>=30){printf("Exceed %.0f%%. License Revoked",s);}
    else if(s>=10){printf("Exceed %.0f%%. Ticket 200",s);}
    else{printf("No");}
    return 0;
}
```
`%d` → `%.0f`：适配 `double` 类型，`.0` 确保输出无小数位

```
    int y,m,d,k=0;
    scanf("%d/%d/%d",&y,&m,&dd);
    int k,a[2][13]={{0,31,28,31,30,31,30,
    if((y%4==0&&y%100!=0)||y%400==0){
        k=1;
    }else{k=0;}
    int day=0;
    for(int j=0;j<mm;j++){
        day+=a[k][j];
```
a[2][13] 不是1

```
int main()
{
    int n;
    struct Student stds[20];
    scanf("%d", &n);
    //#1_begin------------
    getchar();//换行符留在输入缓冲区
    //#1_end==============
    for(int i = 0; i < n; i++)
    {
        char str[20];
        gets(str);
        int pos = 0;
        while (
            str[pos]>='A'&&str[pos]<='Z'||str[
            //#2_begin------------
            //#2_end==============
        ) {
            stds[i].name[pos] = str[pos];
            pos++;
        }
        stds[i].name[pos] = '\0';
        //#3_begin------------
        strcpy(stds[i].num,str+pos);
        //#3_end==============
    }
```
getchar()

```
    for(int i = 0; i < n; i++)
    {
        if(
            strlen(words[i])>26&&strcmp(words[i]+strlen(words[i])-2,"ed")==0
        )
        {
            "ed' 是非法的字符常量（C 语言中单引号只能包裹 1 个字符），必须改成双引号的字符串 "ed"，否则编译器会直接报错
```

```
char* trim(char *str)
{
    char *left = str;
    char *right;
    //#1_begin------------
    right=str+strlen(str)-1;//换行符留在输入缓冲区
    //#1_end==============
    while(
        //#2_begin------------
        *left==' '||*left=='\n'
        //#2_end==============
        );
    {
        left++;
    }
    while(
        //#3_begin------------
        *right==' '||*right=='\n'
        //#3_end==============
        );
    {
        right--;
    }
    *(right + 1) = '\0';
    //#4_begin------------
    return left;//换行符留在输入缓冲区
    //#4_end==============
}
```
如果返回str返回的还是原字符串起始地址

```
#include<stdio.h>
int main(void){
    int n;
    scanf("%d",&n);
    int a[8][8],i,j;
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            scanf("%d",&a[i][j]);
        }
    }
    int flag=1;
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            int flag1=1,flag2=1;
            for(int k=0;k<n;k++){
                if(a[i][j]==a[i][k]){
                    flag1=0;break;
                }
            }
            for(int k=0;k<n;k++){
                if(a[i][j]==a[k][j]){
                    flag2=0;break;
                }
            }
            if(flag1==1&&flag2==1){printf("%d %d",i,j);return 0;}
            else{flag=0;}
        }
    }
    if(flag==0){
        printf("NONE");
    }

    return 0;
}
```
运行错误：循环条件i，j混淆

找鞍点 Flag!!!!!

```
int n,i,j,k,num[1000],sum[1000]={0};
scanf("%d",&n);
for(i=0;i<n;i++){
    scanf("%d",&num[i]);
}
int count=0;
for(i=0;i<n;i++){
    for(j=i+1;j<n;j++){
        int flag=0;
        for(k=0;k<count;k++){
            if(num[i]==num[j]==sum[k]){
                flag=1;break;
            }
        }
    if(flag==0){sum[count]=num[i]+num[j];count++;}

    }
}
printf("%d",count);
```
外层遍历“待检查的元素”，内层遍历“已处理过的元素”，count记录处理元素

```
#include<stdio.h>
int f(int m,int n);
int main(void){
    int m,n,a,b;
    scanf("%d%d",&m,&n);
    if(m>n){a=f(m,n);}else{a=f(n,m);}
    b=m*n/a;//最小公倍数
    printf("%d %d",a,b);
    return 0;
}
int f(int m,int n){
    if(m%n==0){
        return n;
    }else{
        return f(n,m%n);
    }
}
```

易错点

变量名和数组名不能重名

`%5d` 补的是空格，而 `%05d` 才是补 0

`puts()` 输出字符串的核心要求是：字符串必须以 `'\0'`（空字符）结尾

单个字符： ch=getchar() /scanf ("%c", &ch)
字符串：1、一个个读取字符：`int i=0;`
`while((arr[I]=getchar())!='\n') i++; arr[i]='\0';`
2、整个读取： 不能带空格scanf("%s", str);（输入 "hello world" 仅读 "hello"）/
可以带空格scanf("%[^\n]", str);（输入 "hello world" 全读）
/可以带空格fgets(str, sizeof(str), stdin)

- scanf/gets：是否允许"空格"
- printf/puts：printf 可输出转义字符；puts 将'0' 转换为'\n'

输出 24

八进制
n= log(x)/log(8)：计算x的八进制最高位的位数（如八进制71的n=1，因为8^1=8）。
high = x / 8^n：取八进制的最高位数字。
Low = x % 8：取八进制的最低位数字。
= x % 8：取八进制的最低位数字。
若high + low != 8则 break，否则：
去掉最高位：x = x % 8^n；
去掉最低位：x = x / 8；
计数count++。
最终返回count==2（即恰好完成两次上述操作）。

字符	含义	ASCII 码值（十进制）	二进制	用途
'\0'	空字符（Null）	0	00000000	字符串结束标志，标记字符串的结束位置
'0'	数字字符 0	48	00110000	表示数字 0 本身（不是 0）

十进制	无	0–9	8, 10, 123
八进制	0	0–7	07, 010 (对应十进制 8)
十六进制	0x/0X	0–9, a–f/A–F	0x8, 0XA, 0X10