C 语言复习笔记
2024/1/7

第一章 预备知识

1.3.1 计算与进制

十进制转二进制：整数除二取余，从下往上（无误差）
小数乘二取整，从上往下（除个别有限运算外，有误差，常规定一定精度）

二进制转十进制：直接按位权展开（无误差）
10.11B=1*2+0*1+1*0.5+1*0.25=2.75

二进制转十六进制：四个二进制位对应一个十六进制位，即 0000~1111 对应 0~F【十六进制：0~9，A(a)~F(f)（10~15）】

二进制转八进制：三个二进制位对应一个八进制位
11 101.110 $1_{(2)}$=011　101.110 100=35.64

写法：一般只能在汇编程序语言中直接书写二进制数字，以字母 B 结尾；十六进制数在汇编语言中以 H 结尾
c 语言通常用前缀 0X 或 0x 表示其后是十六进制数，用 0nnn 表示整型八进制数

1.3.2 计算机体系结构

1.3.3 信息与编码

计算机本质上只能识别二进制数据，将信息表示为二进制数据的过程叫编码(coding，现常用来指代编程)

1.字符编码：

美国国家标准学会(ANSI)制定基于拉丁字母的 **ASCII**(美国信息交换标准)，包含 95 个可显示字符与 33 个控制字符（后来扩充至 256 个）

位 1bit 1b 最小数据单位 信息编码与传输中常用
字节 1byte 1B=8b 信息存储常用
2^8=256，**每个 ASCII 码占 1B**
注：K=2^{10}，M=K^2=2^{20}，G=K^3
Unicode 2 字节 中日韩

2.数字编码：

有符号整数编码-**补码**

① 增添符号位前缀 1 负/0 正（原码）
② 负数除符号位外按位取反（反码），再加 1
非负整数的原码/反码/补码一致

例-14==1000 1110→1111 0001→1111 0010

**应用：利用 ASCII 码差值转换大小写/字符串转数

第二章 数据与运算

2.1 数据与数据类型

1）常量

*编译器直接根据数值确定类型,如 80int，1.85double
*整型八进制(Octal)常量以 0 开头 十六进制 0x/0X
*浮点型常量
十进制小数形式:符号、数字、小数点；整数和小数部分可以省略一个 如-.123, 123., 0., .0
指数形式：整数(或小数)e(E)整数；e 前后必须有数字！e 后必须为整数 如 123.4e-5

2）变量 内存中一小块被命名的存储空间，用于数据存储与操作

***先声明后使用** 数据类型+标识符

注：

① 变量赋初值时
int a,b,c=a=b=0;//T
int a=0,b=0,c=0;//T
int a=b=c=0;//F 没有声明 bc

② 标识符：用于给变量、函数、结构体等命名，由数字字母下划线组成，且不能以数字开头，合法的用户标识符**不能含 C 语言关键字**（32 个；数据类型/流程控制/存储类 /struct union/sizeof/typedef/enum） main 可以的

3）数据类型：区别在于占据字节多少+编码方式
关注其特征 编码方式 取值范围 内存大小 可以施加的运算方式

1.整型

*溢出

例一 short int -32768~32767
short s=32767;
s+1=-32768

例二 不引入第三个变量实现两个变量值的交换：
a=a+b; 或 a=a*b;
b=a-b; b=a/b;
a=a-b; a=a/b;
但这种方法可能导致溢出，故不安全

2.字符型 char 1B
存储 ASCII 码表中的字符编码即英文字符集
实质上是单字节整数

类型	格式符（字符）	字节数	取值范围
unsigned char	%c	1	0 to 255
char	%c	1	-128 to 127
signed char	%c	1	-128 to 127

字符型常量 单引号括起来的一个字符
转义字符 '\'开头的特殊字符，如'\n'回车,'\\'打出单

斜杠，'\"'双引号，**'\ddd'1 到 3 位八进制数** ASCII 码代表的字符（如'\101'表示'A'），**'\xhh'1 到 2 位十六进制数** ASCII 码表示的字符

字符型变量 算术运算

printf("a; %c,%d",a,a);//输出 a: a, 97

字符串常量 双引号括起来的字符序列

串尾：ASCII=0，转义形式'\0'，不显示，不计入有效长度，但占 1B （sizeof("")==1）

getchar()函数逐个读取字符，但 getchar()前后要用 while(getchar()!='\n'); 来清空缓冲区

如：输入 123456 时，键盘将"123456\n"写入到缓冲区，scanf 会读取"123456"，那么缓冲区就还剩下"\n"。继续往下走，getchar 就会将"\n"读取

3.浮点型

关注精度(有效数字位数)&取值范围

优点：数据范围大 缺点：精度有限→判断两个浮点数相等的结果可能是错的→定义误差范围

if(c<3.14+epsilon && c>3.14-epsilon)

*浮点运算为保证精度优先使用 double 输出为"%lf"

派生类型

2.2 运算符与表达式

表达式：用运算符把操作数连接起来，并符合 C 语言语法规则的式子，常量，变量，函数调用也属于表达式；表达式的类型即表达式的值的类型，取决于操作数的类型以及它们所做的运算

例：

a+2.5*b-6/c+'z'

混合类型**算术运算**，值为 **double 类型**

!((a<1e-6) && (b==(c+sqrt(f))))

逻辑运算的结果是'逻辑值'

C 语言用**整型**代表逻辑类型

1.主要运算符的**运算顺序**

单目>双目，运算>关系>逻辑>三目(选择)>赋值

***/% > +- && > || (<,<=,>,>=) > (==,!=)**

逗号运算符优先级最低，自左向右结合

exp1,exp2 逗号表达式的值是 exp2 的值

x=(a=3,6*a); //a=3, x=18

x=a=3,6*a; //a=3, x=3

误区：优先级最高的一定先算吗？

NO，e. g. **逻辑运算的短路效应**

此外，后置--++的优先级最高，但其操作数值的变化在当前表达式中不体现出来

2.操作数

*个数：一元 二元 三元

*类型： %取余必须是整型或字符型，**运算结果符号与左操作数相同**

*数据类型转换 赋值表达式的类型就是左操作数的类型，运算表达式自动转换为范围大的；强制类型转换，如(float)x/y，但 x 本身的值/类型不变

3.结合方向

***右结合少**：单目、赋值、三目(? :)

例题

int a=3,b=4;表达式 a=(b>=a>=2)?1:0 的值为 0

(b>=a)值为 1，1>=2 值为 0

4.其他要注意的：

①自增自减赋值的操作数*必须是**左值**!

```
int i=3, j, a, b=1, c=2;
j = ++i;        /* i=i+1, j=i, i=4, j=4 */
j = i++;        /* j=i, i=i+1, i=5, j=4 */
j = -i++;       /* -(i++), i=6, j=-5 */
j = i++*2;          /* (i++)*2, i=7, j=12 */
a = (b+c)++;        /* 非法!!! */
a = 34++;           /* 非法!!! */
j = ++i++;          /* 非法!!! */
```

b+c,i++都是表达式，不是左值!

数组名不能作为左值!

i=-i++; /***同一表达式中对同一个变量进行多次修改并且没有使用序列点，会导致未定义行为（undefined behavior）**。i=-i++的结果无法确定，在不同的编译器或不同的环境下，i 的最终值可能会有所不同*/

② sizeof 操作数可以是常数/变量/类型名/表达式

sizeof(type) //求出该类型内存字节数

sizeof(char)==1 字节

sizeof 表达式 //可以不写但最好写括号； 但**表达式本身不被求值**！如 sizeof(i++)，i 值不变

```
char *p="hello";
sizeof(p);//地址一般是 4/8B，由电脑决定
sizeof(*p)==1//h 的内存，char 1 B
```

③ 逻辑运算符

逻辑表达式数据类型是 int,真 1 假 0

但 c 语言中非零值都会被判断为真

短路效应 *可能对值产生影响

④ ==相等 =赋值!!!! 别再错了我求求你

int a=4; a/=2;　　//a=a/2=2;
int a=4; a%=2;　　//a=a%2=0
2a 不是合法标识符
2a+1 不是合法表达式 2*a+1 才行
⑤+-作正负号时为单目运算符，右结合且优先级（2）与前置++，前置--，*，&，!，sizeof，（类型）一致

2.3 输入与输出

C 语言没有专门的输入输出语句，而是用函数接受输入、输出结果

printf 格式控制

%格式符	含义说明
d, i	有符号十进制整数，正数不输出符号
o	无符号八进制整数，不输出前导0
x, X	无符号十六进制整数，不输出前导0x或0X
u	无符号十进制整数
f	小数形式单、双精度数，缺省精度为6位小数
e, E	以规范化指数形式输出浮点数，缺省精度为6位小数
g, G	以f(F)和e(E)两种格式中较短的一种输出
%	输出一个%
.nf	n是正整数，表示小数点后保留n位
-	输出内容左靠齐（默认右靠齐）

scanf("格式串",地址列表) 注意：
①格式字符串中的普通字符须按相应位置输入
　　scanf("i=%d",&i);//输入 i=3
②多个输入项之间用一个或多个空白字符(空格/回车/Tab)分割 不能用逗号等
③如规定了最大宽度，但输入数据超出宽度，则按宽度截取
　　scanf("%3d",&i);//输入 12345 回车 i==123
例
```
int i, j;
char c;
scanf("%d%d", &i, &j);
/* "2 3" or "2<TAB>3" or "2<Enter>3" */
scanf("i=%d, j=%d", &i, &j);
/* "i=2, j=3" or "i=2, j=3" */ 有空格则 j 出错
scanf("%d%c%d", &i, &c, &j);
/* "2 a 3" => i=2, c=' ', j 出错 */
 /* "2a 3"  => i=2, c='a', j=3 */
scanf("%d",&n);//取地址运算符&
printf("%.2f",sum)//保留两位小数
//四舍五入？保留有效数字？
```

2.4 关系运算

== != >< <= >= 二元
操作数几乎可以是任意类型，
关系表达式的结果只有假 0/真 1，且为 int 类型

2.3.2 逻辑运算

！ && ||
操作数几乎可以是任意类型，结果假 0 真 1
特殊性（**短路效应**）：&&||（&&）||）连接的表达式从左往右算，**确定结果真假后立即停止运算**
&& 左假不再计算
|| 左真不再计算

De Morgan 定理(!进出，运算符取反)
!(a&&b) == (!a)||(!b)
!(a||b)==(!a)&&(!b)
!((a&&b)||c)==(!a||!b)&&!c

2.5 循环与数组

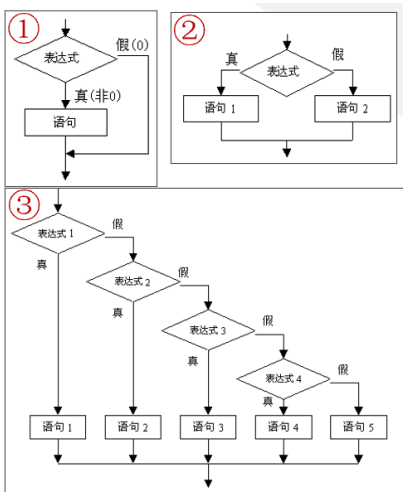
声明的类型和使用的类型相似
int *p
char a[1][4]
double (*p)[10]// 行指针，*p 的类型是 double[10],(*p)[i]的类型是 double
double *p[10]//指针数组,a[i]类型为 double*,*a[i]类型为 double*

第三章 控制结构

3.1 分支结构

3.1.1 if 语句

if(条件){语句;}
else if(){}
else{}
if 语句的嵌套注意 if-else 配对关系：else 总是与其上面距离最近且同一层级的 if 配对；使用{}明确限定 if-else 的配对关系



3.1.2 **switch 语句**
switch(整型表达式)
{

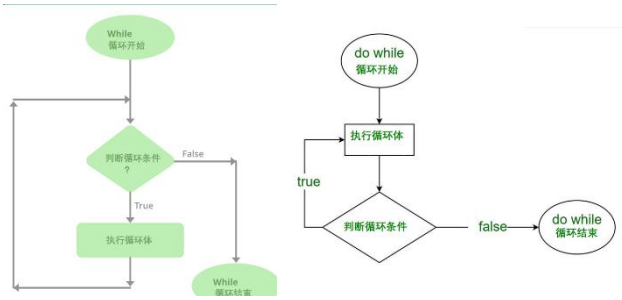
case 常量表达式 1：语句；break；
…//case 的先后顺序不影响结果
default: 语句；break；//default 可以不写**}**
例题：switch-case 判断 GPA 分段问题
level=(score>=95) + (score>=90) + (score>=85)+…
“打怪升级“ 每个 level 对应一个 case 不必写 100 个 case

3.2 循环结构

死循环：循环条件永远为真

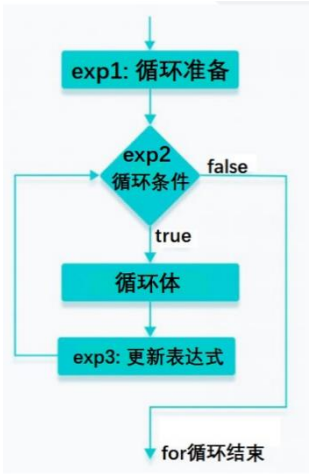
3.2.1 while 语句 & do while 语句

while(表达式)
{语句；}//循环体
do
{语句}while(表达式);//先执行循环体再判断条件



for 语句

for(exp1;exp2;exp3) //exp1;
{语句 4;} //while(exp2){语句； exp3;}
1→2→ 4→3→2→4→3→2…

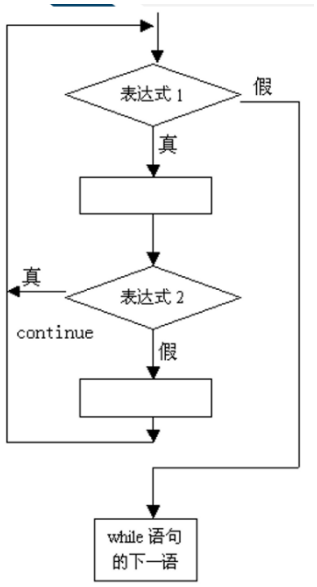


注：exp1 初始化 i,
exp2 循环条件, 值非 0 即执行循环体,
exp3 更新 i
123 都可以省略,
但分号不能省

3.3 break & continue （是语句!

break
作用：
① **用于 switch 语句中跳出 switch 语句**
② **用于循环体中跳出当前循环，一般有条件 if 控制；如果有多层循环存在，break 只跳出它所在的那一层循环语句，不影响外层循环执行**
③ **不能用于循环语句和 switch 语句之外的任何其他场合**

continue
作用
① **仅结束本次循环（跳过循环体中 continue 之后的语句，转到循环条件的判定）**
② **通常加上 if 条件控制**
③ **仅用于循环语句**



重要例题:
1. **判断素数**
   注: 因子遍历到 sqrt(m)即可
2. **求Π(利用 arctanx 的 Maclaurin 公式)**
3. **求最大公约数和最小公倍数**
   辗转相除/更相减损/穷举

$$gcd(a,b)=\begin{cases} a & b=0 \\ gcd(b,a\%b) & b\neq 0 \end{cases}$$

$$lcm(a,b)=a*b/gcd(a,b)$$

4. **求斐波那契数列**

第四章 数组 批量数据的组织与处理

4.1 一维数组
4.2 多维数组
4.3 字符型数组
4.4 排序与查找算法

4.1 一维数组

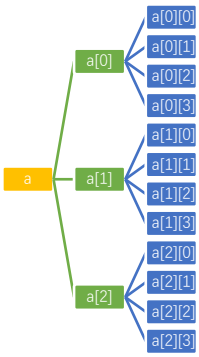
概念：一组同类元素的集合
类型可以是基本类型/派生（构造）类型

统一的数组名加下标可以确定数组中的元素
*数组存储在连续的存储区，元素之间是相邻的
一维数组定义：类型 数组名[常量表达式]
说明:

- 常量表达式表示数组的大小(元素个数)，长度是固定的。
- 表达式中不能包含变量，否则为 VLA（可变长度数组，C99 允许 VLA，VC++不支持)
- 数组作为一个整体与数组元素是两种不同的数据类型。**数组名代表了数组的起始地址。**
- 数组元素引用 数组名[下标] **下标 0~N-1**

一维数组的初始化
初始化—在**定义数组**的时候赋初值
① 对数组整体赋初值 int a[2]={1,2};
可以不指定数组长度，由初值个数决定数组大小
② 只给部分元素赋初值 int a[2]={1}//1,0
未指定初值的其余元素均为 0;
但不初始化时，认为元素是随机数
更常用的是定义之后，循环赋值
一维数组应用
筛法求素数
冒泡排序法

```
int a[5]={1,3,2,4};
int i,j,m;
for(i=0;i<5;i++)
{
    for(j=0;j<4-i;j++)
    {
        if(a[j]<a[j+1])
        {
            m=a[j];
            a[j]=a[j+1];
            a[j+1]=m;
        }
    }
}
```



4.2 二维数组
类型 数组名[常量表达式][常量表达式]
可以看作特殊的一维数组--元素是一维数组的数组
按行存放，先行后列
char a[3][4]; /* 3 行 4 列*/
char a[3,4]; /* 逗号表达式
以最后的值为准 a[4]*/
引用：int a[m][n]
a[2][3]==a[2*m+3]

初始化
① **分行给二维数组赋初值**
int n[3][4]={ {1,2,3,4},
{5,6,7,8},
{9,10,11,12}};
② **对部分元素赋初值**
int a[3][4]={{1},{2,3},{10}};//剩下都是 0
③ **给二维数组赋初值时第一维长度可省略**
但无论有无初值都要用{}分开，以此确定第一维长度
应用：矩阵 转置/乘法

4.3 字符数组
char a[10]; **看作字符串常量**
→ 字符数组应包含字符串结束标志'\0'
初始化：char c[]={'c', '+', '+', '\n' }; //不安全
char c[]={'c', '+', '+', '\n' , '\0' }; //OK
char c[5]={'c', '+', '+', '\n' }; //OK
char c[] = "c++\n"; //OK, 双引号表示字符串 自带串尾\0
用字符数组名引用字符串变量

1. char str[20];
   scanf("%s",str); //不用&，不能能含空格/缩进/回车，否则截断
   printf ("%s",str);
2. 二维数组
   char names[3][10] = {
   "Tom",
   "Jacky",
   "John" };
   printf(" %s ", **names[1]**); //输出 Jacky

字符串处理函数

1. puts(str) 输出字符串，到'\0'为止并换行
gets(str) 输入字符串，可输入空白字符，回车结束
fgets(str,n,stdin) 可以读取换行符，最多 n-1 个字符
/*超出的部分还在缓冲区，最好用 getchar()吃掉*/
2. strcat(st1,st2) 把 2 接 1 后面
   strcpy(st1,st2) 把 2 赋值给 1
   strcmp(st1,st2) st1>st2 返回值>0;= =0;< <0
   strlen(str) 字符串实际长度|最常用

scanf("%s",str)//不能有空格
gets(str) fgets(str,MAX_SIZE,stdin)
getchar()//每次输入一个字符，灵活但**需要手动加上串尾'\0'

4.4 排序和查找算法

选择排序 每次找一个最大的，放在所遍历序列之首

```
int a[5]={1,3,2,4};
int i,j,m,max;
for(i=0;i<5;i++)
{
    max=a[i];
    for(j=i;j<5;j++)
    {
        if(max<a[j])
        {
            m=a[j];
            a[j]=max;
            max=m;
        }
    }
    a[i]=max;
}
```

```
int a[5]={1,3,2,4};
int i,j,m,max;
for(i=0;i<5;i++)
{
    max=i;
  for(j=i;j<5;j++)
  {
    if(a[j]>a[max])
      max=j;
  }
  if(max!=i)
  {
    m=a[i];
    a[i]=a[max];
    a[max]=m;
  }
}
```

插入排序 把新元素插入有序列

1.开始时，将第一个元素看作已经排好序的有序表;

2.排序过程使用双层循环，外层循环针对除了第一个元素之外的所有元素，将其插入到前面的有序表，

3.内层循环**根据当前待插入元素大小向前查找插入位置，并进行比较和移动**

```
#include<stdio.h>
void insertionSort(int *a,int n);
int main()
{
    int arr[]={1,9,10,2,5,3};
    int i,n=sizeof(arr)/sizeof(arr[0]);
    insertionSort(arr,n);
    for(i=0;i<n;i++)
     printf("%d ",arr[i]);
}
void insertionSort(int *a,int n)
{
    int i,j,key;
    for(i=1;i<n;i++)
    {
        key=a[i];
        for(j=i-1;j>=0 && key<a[j];j--)
        {
            a[j+1]=a[j];
        }
        a[j+1]=key;//注意是j+1，可以手推一下
    }
}
```

顺序查找

二分查找

```
int a[N]={ … … };
int   i=0, j=N-1,mid, x;
scanf("%d", &x);
while(i<=j) //即数组不为空
{   mid= (i+j)/2;
    if   (x== a[mid]) break;
    else if (x>a[mid]) i= mid+1;
            else j=mid-1;
 }
if(i>j)     printf("没找到 x");
else printf("x 位于%d "， mid);
```

第五章 函数 ***【c 语言的基本单位】***

C 函数特点

1. 函数可实现程序结构的模块化
2. 一个程序由一个或多个源程序文件组成，一个源程序文件可包括一个或多个函数
3. **C 程序的执行从 main 函数开始，并回到 main 函数结束**
4. 函数之间可以相互调用，或调用自身
5. 程序中各函数**分开定义，不嵌套**

5.1 函数定义

形式

类型 函数名**(参数声明)**

{ //函数体开始

函数内局部变量定义

语句

} //函数体结束

关于函数“类型”

- 指函数**返回值的类型**
- 若省略此项，则认为返回类型是 **int**
- 若无返回值，应定义类型为 **void**

“参数声明”

- **类型** 形参名，**类型** 形参名，…
- **调用时顺序对应、类型匹配**

函数体:

- 局部变量定义和语句部分
- 类型、参数、函数体内容都可省略:

dummy( ) //函数类型默认为 int，（）不可省略

{ } //无操作

函数可递归调用，不可递归声明（没这个东西）

5.2 函数调用

1.一般形式 函数名(实参表列)

说明 ①即使没有实参，**（ ）**也不能省略；多个实参之间用逗号分隔

②实参与形参**按顺序对应，类型匹配**

③**实参的求值顺序不唯一**（取决于具体编译器的实现），所以不要使用如下调用形式:

printf("%d, %d", i, i++);

④**调用之前先声明!**

定义也有声明作用（定义在前，调用在后，也不必声明）；库函数的声明包括在头文件里，不需要在源程序声明

2. 函数调用的方式

函数语句 **printf("%d\n", i);**

函数表达式**(c + d * max(a,b)) % e**

作为函数参数 **gcd(max(a,b), min(c, d))**

说明【函数调用】可以看作表达式，

和函数返回值类型相同的表达式一样使用；

函数调用还完成了函数中定义的操作。

相当于 int m=max(a,b);**m 参与了接下来的运算**

3.函数的返回值的说明

return 语句

形式

return; 或 **return *expression*;**

功能 函数返回语句，结束函数调用，返回到主调函数；如果需要，还可以带回函数**返回值**

说明

① 函数返回值的类型在函数定义中指定

② 函数返回值通过函数中的 return 语句带回
若函数无返回值，则可以没有 return 语句

③ **return 语句后的表达式类型，应与函数返回值类型相同，或可以兼容赋值；两者类型不同时，自动做类型隐式转换**

④ 函数需要返回值时，若缺少 return 语句，或 return 语句未带返回值，则**返回一个不确定值**

⑤ 若要返回多个值，可以用指针带回，return 一般返回的是状态（如 1/0），便于 main 判断

4.参数传递

形参：**定义**函数时，函数名后()中的参数

实参：**调用**函数时，函数名后()中的参数

① 实参可以是常量、变量、表达式、函数调用

② 实参和形参必须类型相同或赋值兼容

③ 实参和形参是不同的变量

具有不同的存储空间、生存期和作用域

参数传递概念

1.参数传递是**值的传递**--**实参数据**对形参的“值传递”(赋值)

2.单向传递

只能把实参的值传递给形参

- 不能把形参的值传递给实参!

2. 对形参的值所作的改变也不会带回给实参!

5.3 数组作为函数参数

1.数组元素作为函数参数

数组元素作为参数与一般变量参数是一样的，传递的是**数组元素的值**

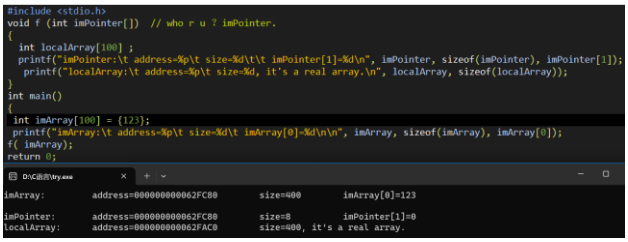
2.数组名作为函数实参

C 语言中的数组名代表了数组在内存中存放的起始地址，所以 数组名作为函数的参数 传递的是**数组的起始地址**，即传递数组“**地址**”/指针值。

说明:

（1）用数组名作为函数的参数，必须对实参数组先定义； 形参必须说明成与实参数组类型一致的数组。

（2）形参数组和实参数组的数组名可以不一样，数组的大小也可以不一致。由于传递的是地址，所以形参地址和实参地址一样，形参数组和实参数组在内存中实际上是**共用了同一块连续的地址空间**。(或者说形参数组并不是真的数组，形参数组名实际上是个指针变量)



int swap(**int a[ ]**)//指针，实际操作对象是实参，所以成功修改

数组长度 int n=sizeof(a)/sizeof(a[0]);

也可以 int sum(int *a, int n)

5.4 递归函数

一个自己调用自己的函数称为递归函数（自调用函数）。分为直接递归和间接递归

设计递归函数的两个要点：

① 初始条件(递归返回条件)

② 递推关系：每一步利用前一步的结果，计算模式相同。

例题:

例 2: 用递归方法求 factor(n)=n!。

factor(n)= 1, (n=1)

factor(n)= n*(n-1)!, (n>1)

```
long factor( int n);
int main()
{  long  n,  y;
   scanf("%ld", &n);  // n>0
   y=factor(n);
   printf("%ld! =%ld\n", n, y);
}
long factor(int n)
{
   long  f;
   if ( n==1)     f = 1;
   else
        f = factor(n-1) * n;
   return(f);
}
```

例 3. 递归求最大公约数

```
#include "stdio.h"
int main()
{
   int   a=0, b=0, x;
   printf("输入两个正整数: ");
   while (a <= 0 || b <= 0)
      scanf("%d%d", &a, &b);
    x = gcd(a, b);
   printf(" gcd =%d\n", x);
}
int gcd(int a, int b)
{
   if (a % b == 0)
       return (b);
   else
       return ( gcd(b, a % b) );
}
```

汉诺塔问题

```
//主函数
int main()
{   int h;
   printf("\ninput number:\n");
   scanf("%d",&h);
   printf("tower disks %d\n",h);
   hanoi(h, 'A', 'B', 'C');
}
//递归函数
 void hanoi(int n, int x, int y, int z)
{   if(n==1)
     printf("%c-->%c\n", x, z);
   else
   {
     hanoi(n-1, x, z , y);
     printf("%c-->%c\n",  x, z);
     hanoi(n-1,y,x,z);
   }
```

递归函数小结:

① 每一层的函数调用都有自己的局部变量（含形参），在栈内存的不同层。

② 每次函数调用都会有一次返回。

③ 位于递归调用前的语句在每层函数调用时的执行顺序 和 递归调用 次序相同。

④ 位于递归调用后的语句在每层函数调用时的执行顺序 和 递归调用 次序相反。

⑤ 递归函数中**必须包含可以终止递归调用的语句**。

5.5 常用库函数

<math.h>

fabs(float) abs(int)//类型不对会强制类型转换

<stdlib.h> <time.h>

srand(time(NULL)) rand()

malloc calloc free realloc

第六章 变量

6.1 局部变量与全局变量

变量的作用域：变量有效的代码**空间**范围

- 变量在作用域才能被引用
- 根据作用域可以把变量分为**局部变量**和**全局变量**
- **在源码上即可确定作用域**

变量的生存期：变量在内存中有效的**时间**范围

- 变量在生存期内才存在，可多次出现，例如函数多次被调用时
- 生存期体现在程序执行时，和变量的存储类型有关，分静态和动态存储

局部变量：

- **函数或复合语句内部定义的变量**
- 说明:

① 作用域为当前函数体或复合语句范围

② 不同函数内的局部变量可以重名：它们是不同的变量，作用域不同，不会相互干扰

③ 函数形参也是局部变量的一种

④ 若多个同名局部变量作用域重叠，则最内层复合语句中定义的局部变量有效

全局变量(外部变量)

函数之外定义的变量

说明

① 作用域从变量定义处到当前源程序文件结束

② 全局变量可以被本文件的函数所共用

③ 增加函数间传递数据的渠道

④ 若全局变量与局部变量重名，则**局部变量**在作用域内屏蔽全局变量

⑤ 应限制使用过多全局变量

不利于模块化：多处赋值、并发问题

命名冲突可能性增加

不利于代码的复用

6.2 变量的存储方式

动态存储方式：程序运行期间根据需要动态分配存储空间；每次调用函数时，给该函数中的局部变量和形参**分配**存储空间，函数返回后**释放**空间。

函数形参、自动变量等

静态存储方式：程序运行期间分配固定的存储空间

全局变量全部存放在静态存储区中

变量的存储类型

1.auto 自动类型—自动分配和释放存储空间

【局部变量默认，auto 可省略】

未初始化的自动变量值一般视作随机值/garbage value

2.register 寄存器类型—存放在 CPU 中，存取快
只能动态分配存储空间
【只有自动变量和形参可以定义为寄存器类型】
实际存储类型是否为寄存器取决于系统环境
3.static 静态存储类型—静态全局/局部
static 全局变量，限定引用范围为当前文件
static 局部变量 在整个程序运行期间其存储空间都不释放，其值不会丢失。在函数调用结束后，静态局部变量虽仍然存在，但已经不在变量的作用域内，所以其他函数不能引用它。

- 静态局部变量在编译时赋初值（仅一次）
- 若不对静态局部变量赋初值，则初值为 0

4.extern 外部变量—扩展外部变量的作用域
【未用 static 声明的外部变量】
在同一文件内：在某变量定义之前要使用它，则声明外部变量。
在不同文件中：在一个文件中引用其他文件定义的外部变量，可以声明外部变量

- **全局变量默认都是外部变量**，定义一般放在所有函数之前

区分变量的声明和定义
声明：对已定义变量的说明，不分配存储空间
定义：定义一个新的变量，并分配存储空间
定义有声明的功能，但声明不能代替定义

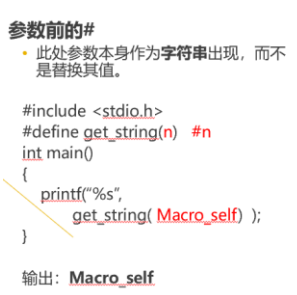
6.3 内部函数和外部函数
外部函数 extern ：
【C 函数默认都是外部的】---相互之间可见，且可以互相调用 extern 在定义时可省略，其他文件调用时声明不可省略 extern

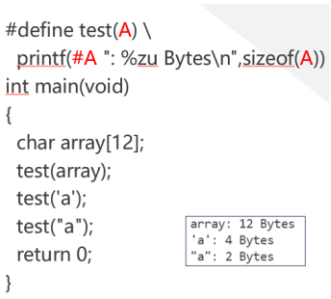
内部函数/静态函数 static:
如果要限制 C 函数的外部属性，即限制互相之间的“可见性”，可使用 static 说明
内部函数只能在所在文件中使用， 类似于静态外部变量的作用范围

第七章 编译预处理

7.1 预处理
在**编译之前**对源程序进行“预处理”
预处理**命令不是 C 语句，末尾没有分号**
不是 C 语言本身的组成部分
不必位于源程序所有语句之首
预处理命令**不在可执行文件中执行**
预处理命令**都以#开始，必须单独占据一行**
包括宏定义、文件包含、条件编译等
举例
#define MAX 1000
#include <stdio.h>

7.2 宏
C 语言中的宏是可代替一组语句或表达式的一个符号名称，当源程序中需要写这些语句或表达式时，可以用该名称代替。
1.无参数的宏定义
一般形式 **#define** 标识符 字符串
说明
① 把源程序中相应标识符替换为#define 定义的字符串仅是简单的字符串替换，没有任何语法正确性检查
② 有效范围从定义处开始，到文件结束，或者由 **#undef** 命令终止作用域
③ 行末没有分号，如有分号将会被一起替换
④ 宏定义中的替换串为空也是合法的
例如： #define DEBUG
2.系统预定义宏
3.带参数的宏定义
一般形式 **#define** 宏名**(**参数表**)** 字符串
说明
① 宏定义仅是把参数作为字符串做简单替换，而不做任何运算求值和语法检查
② 宏名与参数表的括号之间不应有空格
③ 应注意参数替换后可能出现的语法错误和意料之外的运算
④ 应在宏定义内容及其中参数两边加上括号
例： #define max(A, B) ((A) > (B) ? (A) : (B))





7.3 文件包含
7.4 条件编译

第八章 指针

8.1 指针的概念

指针：即地址，指针是对内存地址（一个 32 位或 64 位(根据体系结构)**无符号整数**）的一种形象化的称呼。

8.2 指针变量和指针运算

① 变量的指针和指针变量

变量的指针: 是内存中存储某个变量的**存储单元**（可能有多个字节）的**首地址**

变量的**数据类型**决定所占用存储单元字节数

指针变量: **存放指针的变量--**指针(地址)是一个数据，可以用另一个变量来存放

a) 指针变量可以通过**间接访问运算**去读写它所指向的对象

b) 指针运算和它**所指向数据的类型**相关

② 指针变量的定义

一般形式 基类型 * 指针变量名;

说明 "**基类型**"表示该指针**指向的数据的类型**

可定义基类型为 **void** 类型的指针变量

//指向对象的类型还不知道，使用时再强制转换即可

定义形式 **void *p;**

说明：定义一个指针，但不指定它指向的数据类型

不能通过***p** 引用它指向的数据

void*指针可以与其他任何类型的指针相互赋值和比较(关系运算)，而不需要显式的强制类型转换

常作为函数形参和返回值的类型

③ 指针运算符和取地址运算符

指针运算符* 获得指针指向的内存数据

又称"间接访问运算符" (Indirection Operator)

a) 单目运算符，自右向左结合，优先级较高（2）

b) 操作数为具有指针(地址)意义的值

取地址运算符& 获得操作数的地址

可以认为*与&互为逆过程

a) 单目运算符，自右向左结合,优先级较高（2）

b) 操作数应为各种类型的内存变量、数组元素、结构体成员等，不能是表达式、常量、寄存器变量

④ 指针变量的引用

1.在定义了指针变量之后，要"**先赋值，后使用**"

否则是**野指针，**其所存储的值是不确定的，不应引用，否则是危险的

int a,*p=a;//正确

int a,*p; *p=a;//错误！相当于调用了指针

int a,*p; p=&a;//正确

2.当 **p** 是一个合法的指针，***p** 可以视同与 **p** 的基类型相同类型的变量，用法相同

p1=p2;//p1 p2 指向同一对象

*p1=*p2;//p2 指向的值 赋值给 p1 指向的值

3.NULL: 系统预定义的宏，表示**空指针**。

一般可实现为 #define NULL ((void *) 0) 或 #define NULL ((char *)0）具体实现取决于编译器。

在定义指针变量的时候，如果没有确切的地址可以赋值，则**初始化为 NULL** 是好的编程习惯。

⑤ 指针做函数的参数

参数传递

仍遵循"单向值传递"的规则，这里所传递的值是**指针类型的值**（地址）

单向：指针型**实参变量的值不会因形参而改变**

但是，如果形参指针进行**间接访问运算**，可以修改它所指向的内存数据，也是实参所指向的数据。

用途

借助指针类型参数，函数可以间接访问指针对应的地址单元

可用于处理批量的数据如数组。

可以使函数"返回"多个值（不是 return 途径所以"返回"加上双引号）

⑥ 指针的运算

- **运算类型**

① 算术运算：加、减、自增、自减

p q是指向同一数据集合(数组)的指针

n是整型变量 **(注意避免数组越界)**

运算方式	说 明
p + n	**p**之后第**n**个元素的地址
p-n	**p**之前第**n**个元素的地址
p++	**p**作为当前操作数，然后后移一个元素
++p	**p**后移一个元素，然后作为当前操作数
p--	**p**作为当前操作数，然后前移一个元素
--p	**p**前移一个元素，然后作为当前操作数
p-q	表示**p**和**q**两者之间的元素个数

②关系运算：所有关系运算

条件 ： p,q 是指向同一数据集合(数组)的指针；
否则无意义//因为数组占一段连续内存
运算方式：p<q、p<=q、p==q、p!=q、p>=q、p>q
p<q:判断 **p** 所指元素是否在 **q** 所指元素之前
其他运算的含义与上述类似
③赋值运算：一般赋值、加赋值、减赋值
条件：**p,q** 是指向**同一数据类型**指针，**n** 是整型数据
有意义的赋值方式：**p=q**、
p=q+n、**p=q-n** (要求 **q** 指向数组)
p+=n、**p-=n** (要求 **p** 指向数组)
注意避免数组越界
④指针的运算还包括：
指针运算 *
对指向数组的指针的下标运算 [ ]
对指针变量的取地址运算 &
对指向结构体的指针的指向成员运算 ->
除上述运算方式(包括约束条件)外的其他运算都没有意义；无意义的指针运算不一定会出现语法错误，但可能造成危险的后果

8.3 指针与数组

1. 指针与数组的关系
int a[10];
数组名 a 表示数组的首地址，即 a[0]的地址，a==&a[0]；但不能作为左值，不能放在赋值号左侧，不能被修改，所以可视为“**常量指针**”

2. 指向数组的指针

①**char a[10], *p;** ② **char a[10],*p=&a[0];**
p = &a[0];
③**char a[10], *p;** ④**char a[10], *p=a;**
p = a;

可以通过这个指针变量 p 引用所有的数组元素
引用数组元素的方法：
下标运算符**[]**， 例如 **a[i]** **p[i]**
指针运算符*， 例如 ***(a+i)** ***(p+i)**
注意
① **a+i**(或 **p+i**)不是简单的在 **a**(或 **p**)表示的地址值上简单的加 **i** 个字节，而是加上 **i 个基类型**所需的地址偏移量，即加上 **i*sizeof(int)**
② 指针值可以改变，如 **p++**为下一元素的地址
a++是非法操作，数组名不能被赋值
p=a+i;
***(p++)** 与 **a[i++]** 等价
***(++p)** 与 **a[++i]** 等价

③防止下标越界

```
p = a;                  /* 指针需要先赋值 */
while (p<a+10)          /* 指针在数组范围内移动 */
   scanf("%d", p++);    /* 每输入一次，指针向后移动一个int */
p = a;                  /* 指针重新指向数组起始地址 */
for (i=0; i<10; i++)
   printf("%d", p[i]);  /* 指针使用了[ ]访问元素， 如同 *(p+i) */
```

3. 数组用作函数参数
4. 指向二维数组的指针

① **指针 a**：二维数组名，基类型为一维数组
a+1 偏移量为 1 行 (1*sizeof(基类型))
② **指针 a[i]、*(a+i)**：一维数组名，基类型为**元素的类型 int**，**a[i]+1** 或***(a+i)+1** 偏移量为一个元素
③ 第 **i** 行 **j** 列的**元素**：下面表达式均表示元素的**值**
a[i][j] == *(*(a+i)+j)
***(a[i]+j) == (*(a+i))[j]**
注意 **a** 和***a** 都是指针，但是基类型不同
注意 ***(a+i)**和 ***a+i** 的区别

用指针变量操作二维数组
基类型为数组**元素类型**的指针变量--指向二维数组的元素 **char * p;**
根据一维数组元素和二维数组元素的对应关系，可以访问所有的二维数组元素

基类型为**一维数组类型**的指针变量--指向二维数组的行
int a[M][N];
int (*p)[N] = a; //初始化
//或者 int (*p)[N] ； p = a;
则 **p[i]、*(p+i)、a[i]** 都表示数组的第 **i** 行首地址，即 **&a[i][0]**
p[i][j]、*(*(p+i)+j)、*(p[i]+j)、(*(p+i))[j]都表示二维数组元素 **a[i][j]**
csdn：“ (p[n]+m)这里要注意，使用二维数组地址时，*(p+i)与 p[i]与 p+i 是等价的，都是首地址。可能有同学会奇怪，可能认为 * (a+i)与 a+i 不应该等价呀，a+i 是地址，那 *(a+i)应该是该地址上储存的内容呀。对于这个问题，因为二维数组比较特殊，这个地方特别注意一下就行。”*

8.4 通过指针处理字符串

1) **指针变量指向字符数组（字符串变量）**

- 用法同前述“指向数组的指针”，数据类型为字符型；
- 注意字符串的特点：数组内容可能不“满”，串尾标志'\0'；

```
char a[ ]="abc";        // 用串常量初始化，数组大小：4   (abc\0)
char a[4]={'a','b','c'}; // 效果同上
char b[ ]={'a','b','c'}; // 数组大小：3 (没有串尾，易错，不要这样做！ )

char *p=a;  //或者 char *p=&a[0];
printf("%s", p); // 输出abc
*p='A';        // p[0]='A'; 修改数组的内容。
printf("%s", p); // 输出Abc
```

2) 指针变量指向字符串常量

- 字符串常量在内存中以“匿名”字符数组形式存储
- 字符串常量有自己的地址(指针)，"abc" 被编译器看作 char *
  可以把字符串常量的地址赋给字符指针变量，如：
  char *p; p="abc"; （注意，是把指针赋值给p，p指向串，不是把字符串内容赋值给p ）
- 防止对串常量的修改（属于Undefined Behavior, C99），需要修改时应使用数组
  - 串常量所在的存储区一般被编译器设置为“只读”（read only）（常量不允许被修改），写操作可导致异常中断
- 为了安全，建议把指向串常量的指针定义为 const char * p = “abc”；

3) **const关键字**（类型限定符）

用于定义一个“常量”（不可修改的变量），例如：

```
const float   _pi = 3.14;
int const     size = 10;
```

- 定义时必须同时初始化；
- 然后对PI和size赋值则是非法的。

字面常量或简称字面量（literal）

- 3.14 ， 10 ， ‘a’， “abc”

- const用于限定**指针所指向的对象**不可修改：
  - const float *ptr2pi = &_pi;
  - *ptr2pi = 3.1416; //error!
  - ptr2pi =NULL; // OK

指向的地址可以改变，_pi不能改变

- const用于限定**指针变量**不可修改：
  - int a=1,b=2;
  - int * const pa = &a;
  - p=&b; // error!
  - *p=2; // a=2; OK

指向的地址不能改变，但a可以变

字符数组和字符指针变量比较

1.定义：

char astr[]=”Good morning!”;//串常量内容复制到数组中，**数组内容可修改**

char *pstr=”Good morning!”;//pstr 指向串常量，串常量一般不可修改

定义数组的时候分配可**存放字符串内容**的存储空间

定义指针变量时只分配一个**存放指针**的存储空间

2.赋值：

① 可以用串常量对数组赋初值；不能直接对数组名赋值：

```
char a[4] = “abc”;   //OK
char a[4];   a=“abc”;   //错.
```

可以用库函数 strcpy(a, “abc”); //OK

②对指针变量可以用字符串常量或字符数组名赋值，但都只是**指向**字符串

```
const char * p=“123”;   p=“abc”; //p 变了
char name[]=“C++”;      char *q;
q=name;   *(++q)='P'; *(++q)='P';
```

// q 指向字符串变量 name 并把内容改为”CPP”

③ char str[50], *r;

scanf(“%s”, str); //OK. 输入字符串，存入数组 str

scanf(“%s”, r) ; //错误！ r 未初始化，是野指针。

字符串指针作函数参数：

```
//统计字符串中小写字母的数量：
int na2z( const char s[] )
{
  int count = 0, i;
  for( i=0; s[i] != '\0'; i++)
    if( s[i]>='a' && s[i]<='z' )
      count++;
  return count;
}
```

```
//或者写成如下形式，没有区别：
int na2z( const char *s )
{
  int count = 0;
  for( ; *s ; s++ )
    if( *s>='a' && *s <='z' )
      count++;
  return count;
}
```

```
void str_cpy(char *t, char *s)
{
   while(*t++=*s++); /* 逐个字符复制 */
}
int  main()
{
   char *str1="C Language",  str2[20];
   str_cpy(str2, str1);
   puts(str2);  /* 输出C Language */
}
```

常用操作：

搜索串尾 while(*s) s++;　　while(*s++);

复制串 while(*to++=*from++);

8.5 指针与函数

1. 函数的地址

程序执行时，函数代码也驻留在内存， 函数的地址（指针）就是这段代码的开始地址 —称作函数的入口地址，也称作该函数的指针。

2. 指向函数的指针变量

可以用通过该指针变量调用函数

定义：int （*p）(int, int)；

赋值：p=max;

c=(*p)(a,b);//c=max(a,b);

```
//梯形法求一元函数定积分：
double integral( double (*f)(double) ,
                 double a,     double b )
{
   double s, h;
   int n=100, i;
   h = (b-a)/n;
   s = ( (*f)(a)+(*f)(b) ) / 2.0;
   for(i=1; i<n; i++)
      s += (*f)(a+i*h);
   return s*h;
}//call back函数回调 涉及不同函数的调用
 （有点像宏）
```

```
#include <stdio.h>
#include <math.h>
int  main()
{
   double y1, y2, y3;
   y1 = integral(sin, 0.0, 1.0);
   y2 = integral(cos, 0.0, 2.0);
   y3 = integral(exp, 0.0, 3.5);
   printf("%lf\n%lf\n%lf\n", y1,y2,y3);
}
```

3.返回指针的函数

max(a,b)

```
int * f(int *px, int *py) // 返回整型指针
{
   return *px>*py?px:py; //较大数的地址
}
int main()
{
   int a=2, b=3, c=9;
   *f(&a,&b) = c; //赋值给a和b中较大的数
   printf("%d\n", b);  // 输出9
}
```

```
int * f(int *a, int *b) //返回整型指针
{
return *a>*b?a:b;
//比较的是a[0] b[0]的大小
}
int main()
{
   int i, a[]={1,2,3,4}, b[]={5,6,7,8};
   for (i=0; i<4; i++)
      printf("%d\n", f(a,b)[i]);
```

TYPE *create(int n);//链表创建函数

8.6 指针数组和多重指针

1） 字符串数组：

一般用**二维字符数组**存放多个字符串

char lang[][10] ={“Java”，“**Python”,**“C++”，“HTML”，“SQL”}

2）基类型为一维数组的指针变量

char lang[][10] ={“Java”，“Python”，“C++”，“HTML”，“SQL”};

char （* **p**)[10] = lang;

// p 为指针变量，基类型为 char [10]，指向数组

p == lang; *p == lang[0]

3） 指针数组： 类型 *数组名[长度];

可以指向多个字符串。

char *parray[10] ={“Java”，“Python”， “C++”，“HTML”，“SQL”};

//小括号不必写，[]优先级比*高；

指向一维数组的指针 char (*p)[10]则需要

// parray 为数组名，是指针数组，元素类型为指针 char *，需要**循环赋值**

4）指向指针的指针

基类型为指针类型的指针。例：

char **p; p= parray;

5）命令行参数---main 函数的参数：

int main(void); //无参数

int main(int argc, char *argv[]);

int main(int argc, char **argv);

说明

argc 为命令行参数的个数；

argv 为命令行参数字符串“数组” -- 指针变量，基类型为 char *；

命令行参数包括文件名本身，例如：执行命令行：

a.exe arg1 arg2 arg3 参数个数 argc==4

main 函数怎样得到这些参数？ -- 由操作系统向程序传递参数

命令行参数也叫程序参数，也可以通过命令行之外的其他方式传递，如在 IDE 中设置。

6）分析复杂类型定义形式的方法

从标识符开始，逐层分析其意义

按运算符优先级和结合方向的顺序进行

可能涉及的运算符包括

()自左向右结合：改变结合顺序；或声明一个函数，向外一层函数返回值类型声明

[]自左向右结合：声明一个数组，**向外一层是数组元素类型声明**

*** 自右向左结合：**声明一个指针类型，**向外一层是指针基类型声明**

5 左右法则

这是一个简单的规则，允许解释任何声明。具体解释如下：

从最内侧括号开始阅读声明，向右走，然后向左走。当遇到括号时，方向应相反。一旦括号中的所有内容都解析完，就跳出来。然后继续，直到整个声明被解析。

左右规则的一个注意点：第一次开始阅读声明时，必须从标识符开始，而不是从最内在的括号开始。

举例：

```
int * (* (*fp1) (int) ) [10];
```

按以下步骤开始庖丁解牛：

```
从标识符开始 -------------------fp1
右边是 ），左右向左看，找到了*---fp1是一个指针
跳出()后遇到了(int) ------------指针指向了一个函数，函数中有一个int参数
向左看，找到了 * ---------------该函数返回指针
跳出(), 向右看，找到了 [10] ----返回的指针指向含有10个元素的数组
向左看，找到了 * ---------------数组元素都是指针
在向左看，找到 int -------------每个数组元素指向int类型
```

8.7 动态内存分配及其指针

<stdlib.h>

void *p1=malloc(1024);

char *p2 = (char*)malloc(n*sizeof(char));

double *p=(double*)calloc(n, sizeof(double));

free(p);

realloc(p,50);//重新分配

动态二维数组：

①int *p=(int*)malloc(n*n*sizeof(int));当成一维

②参考下述“生命游戏”代码

```
int main()
{
    int n,g;//设置网格大小和进化代数
    scanf("%d%d",&n,&g);
    char **pcell=(char**)malloc(n*sizeof(char*));
    int i;
    for(i=0;i<n;i++)
    {
        pcell[i]=(char*)malloc((n+1)*sizeof(char));
        /*考虑'\0'所以多一个；注意每个一级指针的地址都
        是动态分配的，不是连在一起的，不能作为一个整体传进函数
        所以定义void evolution(int n,char **lifeMatrix,int g);*/
    }
    evolution(n,pcell,g);
    for(i=0;i<n;i++){
        free(pcell[i]);
    }
    free(pcell);
    return 0;
}
```

分配内存注意：类型、大小(考虑‘\0’)、释放

小结

定　义	含　义
int i;	定义一个整型变量i
int *p;	p为指向整型数据的指针变量
int a[n];	定义整型数组a，它有n个元素
int *p[n];	定义指针数组p，它有n个指向整型的指针元素
int (*p)[n];	p为指向含有n个元素的一维数组的指针变量
int f();	f为返回整型值的函数
int *p();	p为返回值为指针的函数，该指针指向整型数据
int (*p)();	p为指向函数的指针，该函数返回一个整型值
int **p;	定义一个指向指针的指针变量

第九章 结构体

9.1 结构体的定义和引用

1. 什么是结构体

① 结构体允许用户根据自己的需要建立数据类型，是用户自定义的“构造类型”

② 相对于独立的变量，结构体可表示一组相关数据的集合

③ 相对于数组元素的单一类型，结构体可以定义不同类型的成员

2.定义结构体类型

```
struct student
{
    int stunum;
    char name[20];
    float examscore;
    float labscore;
    float totalmark;
} ;
```

①一般作为全局的定义放在所有函数前

②这只是结构体**类型**的定义，类型名是 struct student

③不能赋初值

注意分号

④成员类型也可以是结构体，但不可以直接或间接递归嵌套

```
typedef struct {
        char author[20];
        char tittle[50];
        float price;
} BookInfo ; //类型名

BookInfo  book1, book2;
```

3.定义结构体变量

定义类型的同时定义变量（全局），开头的 student 可以省略

```
struct student
{
    int stunum;
    char name[20];
    float examscore;
    float labscore;
    float totalmark;
} stutable[10];
```

main 函数中定义如下

```
int main()
{
    struct student stutable[10];
```

初始化：顺序类型对应，部分初始化则剩余成员获 0 值

4. 结构体成员引用

一般形式

结构体变量名.成员名

成员运算符 .

具有**最高的优先级，自左向右结合**

说明：

结构体成员和同类型的变量用法相同

若成员类型又是一个结构体，则可以使用若干个成员运算符，访问最低一级的成员：

- 结构体类型变量之间可以直接相互赋值 ---整体赋值 (stu1=stu2)

实质上是两个结构体变量相应的存储空间中的所有数据直接拷贝

包括复杂类型在内的所有结构体成员都被直接赋值，如字符串、结构体类型等

- 函数的实参和形参可以是结构体类型，并且遵循实参到形参的传递规则:单向、传值
- 为了提高程序的效率，函数的参数多使用结构体类型指针

```
struct student stu;
scanf("%f", &stu.score);
stu.num = 12345;
stu.birthday.month = 11;
stu.score = sqrt(stu.score) * 10;
strcpy(stu.name, "Mike");
/*name是数组名，不能直接对数组名赋值，所以调用strcpy;定义char *name则可以直接赋值*/
printf("No.%d:", stu.num);
```

9.2 结构体数组

可用于表示 二维表格

9.3 指向结构体的指针和链表

1.指向结构体的指针

定义、使用与其他基本类型指针类似

可以使用**指向运算符->**引用指针所指向的结构体的成员：结构体指针->成员名

->具有最高的优先级，自左向右结合

若 struct student stu, *p=&stu;

则 stu.num、 (*p).num、 p->num 等效

2.指向结构体的指针用法

注意运算符的结合方向和优先级

struct student stu[10], *p=stu;

++p->num; /* 同++(p->num); */

p++->num; /* 同(p++)->num; */

(++p)->num; (p++)->num;

3.结构体的应用—链表

链表的操作

创建：头插/尾插

遍历链表

删除/插入节点

排序：插入排序

倒置

```
Node* reverseLinkedList(Node* head) {
    Node* current = head;
    Node* previous = NULL;
    Node* next = NULL;

    while (current != NULL) {
        next = current->next;  // 保存后续链表信息

        current->next = previous;  // 将当前节点的下一个节点指向previous

        previous = current;  // 更新previous指针为当前节点
        current = next;  // 将current指针向后移动一位
    }

    return previous;  // 返回新链表的头节点
}
```

先写好大体，注意考虑几个特殊情况下的操作：

链表为空表 (head==NULL)

链表只有一个结点

对链表的第一个结点进行操作

对链表的最后一个结点进行操作

最后一个结点的 next 指针应为 NULL

9.4 共用体

各成员共享一段内存

```
union Data   //定义类型
{
      int    i;
      char ch;
      float   f;
} a ;       //定义变量
union Data   b, c;   //定义变量
```

整体赋值时可以整体引用结构体变量，

但共用体变量不能整体引用

特点：

① 成员共享内存：只能放一个成员的值

② 共用体变量初始化只能对一个成员进行（初值表只有一个常量）

③ 最后一次赋值的成员有效（覆盖了以前所有成员的值）

④ 共用体变量的地址及其各成员地址都是同一地址

⑤ 同类型共用体变量之间可互相赋值

⑥ 共用体变量可作函数参数

⑦ 可用作结构体成员；共用体成员也可以是数组或结构体。

第十章 文件

10.1 文件概述

文件：文件是一组相关数据被作为一个整体而存储，通常存储在**外存**以便在计算机关机时仍能保留。计算机**读文件时将它从外存复制到内存，写文件时则从内存传送到外存。**

C 语言中的文件： 流（stream）

C 语言把文件看作一个**字节的序列**

C 语言对文件的存取是**以字节为单位**的

文本文件 ：字符序列构成的文件，存储字符（文本）的 ASCII 码。内容一般以行为单位（以\n 划分行）。

二进制文件：按数据（如 int、float、double 或其他复杂类型）在内存中的二进制形式直接存储。

注：EOF 并不是文件的内容，文件中没有 EOF“文件尾”标记

10.2 文件指针

FILE *fp;

FILE 是 typedef 定义的结构体别名，所有文件的操作都需要 FILE 类型的指针。

注意不要写成 struct FILE

10.3 文件操作

步骤

①FILE *p;

②打开文件

FILE *fopen (char *filename, char *mode);

filename 要打开的文件名，可含路径名

返回值：成功则返回指向被打开文件的指针，出错返回空指针 NULL（0）

```
if((fp=fopen(“student.txt”,a+)==NULL)
  printf(“cannot open this file!”);
```

文件打开模式：

打开模式	描　　述
r	只读，打开已有文件，不能写
w	只写，创建或打开，覆盖已有文件
a	追加，创建或打开，在已有文件末尾追加
r+	读写，打开已有文件
w+	读写，创建或打开，覆盖已有文件
a+	读写，创建或打开，在已有文件末尾追加
t	按文本方式打开 (缺省)
b	按二进制方式打开

③读写文件数据

函数	功能	函数	功能
fputc	输出字符	fprintf	格式化输出
fgetc	输入字符	fscanf	格式化输入
putc	输出字符	putw	输出一个字
getc	输入字符	getw	输入一个字
fwrite	输出数据块	fputs	输出字符串
fread	输入数据块	fgets	输入字符串

1.int fputc(int c, FILE *fp);//成功返回输出的字符，失败 EOF
int fgetc(fp);
例：文件复制
while((c=fgetc(fp1))!=EOF)　fputc(c,fp2);
//同 getchar，fp 会自动移动到下一个字符
fclose(fp1);　fclose(fp2);

2.fprintf(fp, “分数是：%d\n”, m);
//写入文件，就是 printf 多个 fp
fscanf(fp,”格式控制串”，地址);

3.fputs(char *s, FILE *fp);
字符串结尾的‘\0’不会输出到文件,也不会自动添加换行符

4.文件的定位：
fp 位置指针指向当前读写的位置
每次读写文件，位置指针都会相应移动
可以通过相关函数强制修改位置指针：
rewind 函数 rewind(fp);回到开头
fseek 函数 fseek(fp，偏移量，起始位置);
偏移量 **long** offset; 所以写 100L，-10L 这样的
起始位置
SEEK_SET　(0):文件开始
SEEK_CUR　(1):文件当前位置
SEEK_END　(2):文件末尾
ftell 函数 ftell(fp);成功则返回当前文件位置，否则-1L

④关闭文件
fclose(fp);成功返回 0，出错返回 EOF（-1）

后记

从 22 届开始，期末考试减少了纯语法知识的考察，但程序阅读量提升。
23 届期末卷子由基础知识、阅读代码、改错、**画流程图**、手写代码等题型组成，建议在考前集中练习 1-2 天，回顾一下本学期编写过的重要程序，比如最大公约数、排序、链表。

基础知识：结合 ppt 和课本一起看，做总结的目的也是便于快速回顾。结合题目，基本没问题。考试遇到不会的也不用害怕，选择题分值很小。

代码阅读：刘老师发的练习资料我基本上做完了（编程题写了大致），实测刷题可以很有效的在短期内提高程序阅读速度。不过我平时也会帮同学 debug，虽然更多时候是两个人一起头痛。

画流程图：感觉是为考而考的送分题，课本上有讲，看看就行。

手写代码：一锤定音的重头戏，很多同学都写不完。不过掌握重要代码（建议考前默写，以防眼高手低）之后，提前 15 分钟写完是没问题的。考的不难---没有用到很高雅的数学思维，但**要看出问题的本质还是需要一定的积累和随机应变。**

程设考试其实并不能反映出你编程能力的高低，但只要努力想拿高绩并不难。祝学业顺利。

于宛扬
2024/1/19