目录

[目录 1](#_Toc486759475)

[C语言期末复习 3](#_Toc486759476)

[第一部分：变量和数据类型 3](#_Toc486759477)

[算法的基本概念 3](#_Toc486759478)

[标识符的命名规范 3](#_Toc486759479)

[字面值的表示 3](#_Toc486759480)

[输入输出的格式 4](#_Toc486759481)

[练习 5](#_Toc486759482)

[第二部分：运算符和表达式 5](#_Toc486759483)

[逗号表达式 5](#_Toc486759484)

[赋值表达式 6](#_Toc486759485)

[逻辑表达式 6](#_Toc486759486)

[关系判等表达式 7](#_Toc486759487)

[算术表达式 7](#_Toc486759488)

[条件表达式 7](#_Toc486759489)

[位运算表达式 8](#_Toc486759490)

[练习：编写一个程序，让其以二进制的形式输出 9](#_Toc486759491)

[第三部分：条件、循环 9](#_Toc486759492)

[条件语句 9](#_Toc486759493)

[循环语句 11](#_Toc486759494)

[小测： 13](#_Toc486759495)

[第四部分：指针 14](#_Toc486759496)

[指针变量的声明和含义 14](#_Toc486759497)

[多重指针的声明及其等价形式 15](#_Toc486759498)

[几种特殊的指针 15](#_Toc486759499)

[指针的运算 16](#_Toc486759500)

[第五部分：数组 16](#_Toc486759501)

[数组的定义、初识化和元素的引用 16](#_Toc486759502)

[二维数组的表示 17](#_Toc486759503)

[二维数组得顺序存储 18](#_Toc486759504)

[指针操作数组 18](#_Toc486759505)

[练习 18](#_Toc486759506)

[第六部分：函数 22](#_Toc486759507)

[函数的定义和声明 22](#_Toc486759508)

[函数的调用（在写函数时，尽量使用函数原型） 23](#_Toc486759509)

[一维数组作形参的过程 23](#_Toc486759510)

[在数组中实现元素的查找。 24](#_Toc486759511)

[在数组中实现排序 24](#_Toc486759512)

[动手练习：请定义一个函数实现数组的元素的逆置 25](#_Toc486759513)

[编程求n的阶乘 26](#_Toc486759514)

[生存周期和作用域 27](#_Toc486759515)

[声明作用域 27](#_Toc486759516)

[存储类型说明符 28](#_Toc486759517)

[程序在内存中的分布区域 28](#_Toc486759518)

[指针函数和函数指针 29](#_Toc486759519)

[字符串的比较以及排序 29](#_Toc486759520)

[第七部分：预处理和宏 30](#_Toc486759521)

[文件包含 30](#_Toc486759522)

[对象宏（无参宏） 31](#_Toc486759523)

[函数式宏（有参宏） 31](#_Toc486759524)

[条件编译 32](#_Toc486759525)

[第八部分：结构体 32](#_Toc486759526)

[结构体 32](#_Toc486759527)

[动态链表的构成： 33](#_Toc486759528)

[动态链表 34](#_Toc486759529)

[链表程序 35](#_Toc486759530)

[第九部分：字符串函数 41](#_Toc486759531)

[字符串函数 41](#_Toc486759532)

[1、strlen 41](#_Toc486759533)

[2、strcpy 42](#_Toc486759534)

[3、strcat 43](#_Toc486759535)

[4、strcmp 44](#_Toc486759536)

C语言期末复习

## 第一部分：变量和数据类型

### 算法的基本概念

1. **算法是解决问题的方法**
2. **算法的基本特征**

* **有穷性**
* **确切性**
* **有效性**
* **一个算法有零个或多个输入**
* **一个算法有一个或多个输出**

### 标识符的命名规范

1. **变量名由数字、字母和下划线组成**
2. **字符不能开头**
3. **不能使用C语言的关键字，例如：extern static case等**
4. **C语言的标识符区分大小写**
5. **命名应当有意义**

### 字面值的表示

**前缀表示进制，后缀表示类型。**

**整型字面值**

* **如果整型字面值以0x或0X开头，则是十六进制表示。**
* **如果整型字面值以数字 0 开头，则是八进制表示。**
* **否则是十进制表示。**
* **编译器根据字面值后缀决定类型，例如：L、U、LL、UL、ULL**

**浮点型字面值**

* **科学计数法：5e3**
* **自然数计数：5.3**
* **可采用十进制或十六进制表示（限于C99）**
* **使用f，F，l，L是编译器得到字面值类型，默认为double**

**字符字面值**

* **转义字符：用来表示很难输入的字符**
* **字符转义：’\n’、’\t’等**
* **数字转义：可用八进制或十六进制来表示字符。**
* **十六进制表示：’\xdd’,d为十六进制字符**
* **八进制表示：’\ddd’,d为八进制字符**

**字符串字面值**

* **字符串字面值是由一对双引号括起来的字符序列（以’\0’结束）。**

### 输入输出的格式

**&（对于scanf一定不能忘记加上取地址的符号）**

**%d（以十进制的形式展示出来）**

**%c（以字符的形式展示出来）**

**%s（以字符串的形式展示出来，知道遇见’\0’时结束）**

**%f（对于它来说，输入的时候，若是double形式，则需要用%lf，输出的时候，float和double都可以使用%f）**

**getchar() putchar()输入或输出单个字符 gets() puts()输入或输出字符串**

### 练习

#include<stdio.h>

int main(void)

{

char str[]="abcd";

printf("%s",str+2);//结果是cd

char str[]="abc\0d";

printf("%s",str+2);//结果是c

char str[]="abc\0d";

printf("%s",str);//结果是abc

puts(str);

return 0;

}

#include<stdio.h>

int main(void)

{

printf("a\b");//结果为空

return 0;

}

## 第二部分：运算符和表达式

### 逗号表达式

1. #include<stdio.h>
2. **int** main(**void**)
3. {
4. **int** a=3,b=5,z;
5. z=a>b,a+b;//在这个运算过程中，逗号的运算优先级比赋值运算的优先级要低，因此应该先算赋值运算，再算逗号运算
6. printf("%d",z);
7. z=(a>b,a+b);//在这个运算过程中，赋值的运算优先级比特殊运算符的优先级要低，因此应该先算特殊运算，再算赋值运算运算，在特殊运算中，逗号运算的结果是输出逗号后面的数。
8. printf("%d",z);
9. **return** 0;
10. }

### 赋值表达式

**赋值运算符的优先级只比逗号运算符的优先级高，而比其他的要低。**

#include<stdio.h>

#include<stdlib.h>

int main(void)

{

int x=3;

int y=5;

printf("%d",y\*=x+3);//结果是30

system("pause");

return 0;

}

### 逻辑表达式

**在此运算符中，有一个短路特性：**

**（表达式1）&&（表达式2）如果表达式1为假，则表达式2不会进行运算，即表达式2“被短路”。**

**（表达式1）||（表达式2）如果表达式2位真，则表达式2不会进行运算，即表达式2“被短路”**。

1. /\*
2. 与运算符(&&)和或运算符(||)均为二元运算符。具有左结合性。非运算符(!)为一元运算符，具有右结合性。
3. “&&”和“||”低于关系运算符，高于赋值运算符，且&&的优先级高于||的优先级，“!”高于算术运算符。
4. \*/
5. #include<stdio.h>
6. #include<stdlib.h>
7. **int** main(**void**)
8. {
9. **int** a=3,b=2,c=1,d=5,e=6,f;
10. f=a<b||b<c&&c<d||d<e;
11. printf("%d",f);
12. system("pause");
13. **return** 0;
14. }

### 关系判等表达式

1. /\*
2. 关系运算符都是二元运算符，其结合性均从左至右。关系运算符的优先级低于算术运算符，
3. 高于赋值运算符。在六个关系运算符中，<、<=、>、>=的优先级相同，高于==和!=，==和!=的优先级相同。
4. \*/
5. #include<stdio.h>
6. #include<stdlib.h>
8. **int** main(**void**)
9. {
10. **int** x=3,y;
11. y=2<x>=8;//从左至右运算
12. printf("%d",y);
13. system("pause");
14. **return** 0;
15. }

### 算术表达式

1. #include<stdio.h>
2. #include<stdlib.h>
4. **int** main(**void**)
5. {
6. **int** x=4,y=3,z;
7. printf("%d,%d,%d\n",20/7,-20/7,-20/-7);//当两个操作数均为整型或者是char型时，除法运算实现了整型的运算，结果与被除数的符号相等
8. printf("%f,%f\n",20.0/7,-20.0/7);//只要有一个操作数是实型，则结果的类型为实型
9. printf("%d\n",100%3);//求余运算要求两个操作数均为整型
10. z=x+++y;//++i：i自增1后再参与其它运算。--i：i自减1后再参与其它运算。i++：i参与运算后，i的值再自增1。i--：i参与运算后，i的值再自减1。
11. printf("%d,%d",x,z);
12. z=x+++--y;
13. printf("%d,%d,%d",x,y,z);
14. system("pause");
15. **return** 0;
16. }

### 条件表达式

1. /\*
2. 条件运算符为?和：，它是一个三元运算符，即有三个参与运算的操作数。由条件运算符组成条件表达式的一般形式为：
3. 表达式1 ?  表达式2 : 表达式3
4. 其求值规则为：如果表达式1的值为真，则以表达式2 的值作为条件表达式的值，否则以表达式3的值作为整个条件表达式的值。
5. 在这个过程中，如果表达式1为真，那么表达式3的结果不在进行运算，即发生忽略，反之，表达2的结果进行忽略。
6. \*/
7. #include<stdio.h>
8. #include<stdlib.h>
9. **int** main(**void**)
10. {
11. **int** a=3,b=4,c=5,d=6,max;
12. max=a>b?a:b;//结果就是为了得到a和b中较大的那个数
13. printf("%d",max);//条件运算符的运算优先级低于关系运算符和算术运算符，但高于赋值符。
14. max=a>d?a:c>d?c:d;//条件运算符的结合方向是自右至左。
15. printf("%d",max);
16. system("pause");
17. **return** 0;
18. }

### 位运算表达式

1. #include<stdio.h>
2. #include<stdlib.h>
4. **int** main(**void**)
5. {
6. **int** a=9,b=5,c;
7. c=a&b;
8. printf("%d\n",c);//按位与运算符"&"是二元运算符。其功能是参与运算的两数各对应的二进位相与。只有对应的两个二进位均为1时，结果位才为1 ，否则为0。参与运算的数以补码方式出现。
9. c=a|b;
10. printf("%d\n",c);//按位或运算符“|”是双目运算符。其功能是参与运算的两数各对应的二进位相或。只要对应的二个二进位有一个为1时，结果位就为1。参与运算的两个数均以补码出现。
11. c=a^b;
12. printf("%d\n",c);//按位异或运算符“^”是双目运算符。其功能是参与运算的两数各对应的二进位相异或，当两对应的二进位相异时，结果为1.相同为0.
13. c=~a;
14. printf("%d\n",c);//求反运算符～为单目运算符，具有**右结合**性。 其功能是对参与运算的数的各二进位按位求反。
15. c=a<<4;
16. printf("%d\n",c);//左移运算符“<<”是双目运算符。其功能把“<< ”左边的运算数的各二进位全部左移若干位，由“<<”右边的数指定移动的位数
17. c=b>>4;
18. printf("%d\n",c);//右移运算符“>>”是双目运算符。其功能是把“>> ”左边的运算数的各二进位全部右移若干位，“>>”右边的数指定移动的位数。
19. // 应该说明的是，对于无符号数，在移动的时候前面补0，对于有符号数，在右移时，符号位将随同移动。当为正数时， 最高位补0，而为负数时，符号位为1
20. system("pause");
21. **return** 0;
22. }

### 练习：编写一个程序，让其以二进制的形式输出

#include <stdio.h>

void binary(int num)

{

int i;

for(i=0; i<32; i++)

{

printf("%d", ((num<<i)<0) ? 1 : 0);

if(i%4 == 3)

printf(" ");

}

printf("\n");

}

int main(void)

{

int n;

scanf("%d", &n);

binary(n);

return 0;

}

## 第三部分：条件、循环

### 条件语句

**if语句的一般形式：**

**if(表达式)**

**{**

**语句;**

**}**

**其语义是：如果表达式的值为真，则执行其后的语句，否则不执行该语句。其过程可表示为下图。**

**if – else 语句的一般形式：**

**if(表达式)**

**{**

**语句1;**

**}**

**else**

**{**

**语句2;**

**}**

**注意：如果if语句后面的东西是一个赋值语句，那么这个值一定是真的**

switch语句被称为**多路分支语句**

**一般形式：**

**switch(表达式)**

**{**

**case 整型字面值1:**

**语句1(集合);**

**break;**

**case 整型字面值2:**

**语句2(集合);**

**break;**

**case 整型字面值n:**

**语句n(集合);**

**default:**

**语句n+1(集合);**

**}**

**注意：**

1. **case的标号不能够超过1023个**
2. **表达式的值为整型或者是字符型。**
3. **switch语句的执行方法如下：**

**若表达式的值等于switch 语句中某个case 标号中的整型字面值(每个case 标号中的字面值不能相同)则程序控制转移到该case 标号表示的点，从此点开始执行,即执行其后的语句，然后不再进行判断，继续执行后面所有case后的语句。直到遇见 break跳出switch。**

**若表达式的值不等于任何 case 标号中的字面值，则程序控制转移到default 标号（如果有的话）表示的点，从此点开始执行。**

**若表达式的值不等于任何 case 标号中的字面值，又没有 default 标号，则不执行switch 语句体中的语句。**

1. **在case后，允许有多个语句，可以不用{}括起来。**
2. **各case和default子句的先后顺序可以变动，而不会影响程序执行结果。**
3. **default子句可以省略不用。**

### 循环语句

**for循环语句的基本形式为：**

**for(初始表达式1;条件表达式2;循环表达式3)**

**{**

**语句;** **//** **被称为for** **语句体**

**}**

**执行过程：**

1)执行表达式1

2)判断是否满足表达式2，满足进入，否则退出循环

3)顺序执行循环体内容

4)执行表达式3，然后转入2

注意事项：

for循环中三个表达式都不是必须，不管是否有内容，执行过程不变

循环编写的时候，要特别注意边界

逻辑错误很难排查

while语句是另一种循环语句,其一般形式如下

**while(表达式)**

**{**

**语句；**

**}**

**执行过程：**

**1)判断表达式是否为真，满足进入2)，否则退出循环**

**2)顺序执行循环体内容，完毕后进入1)**

do…while语句是第三种循环语句，基本形式：

**do**

**{**

**语句;** **//** **被称为do** **语句体**

**}** **while(表达式);**

**注意：do...while循环语句至少执行一次(与前面两种语句根本区别:结构不同)**

**执行过程：**

**1)先执行一次循环体内容（由于还要执行一次循环才能退出）**

**2)判断表达式是否为真，满足进入3)，否则退出循环**

**3)顺序执行循环体内容，完毕后进入2)**

**循环语句的执行方式可以被改变：**

break：停止整个循环的执行，执行循环后的内容

continue：停止当次循环执行，执行下一次循环

goto：跳到指定标号执行

**goto语句的一般形式为：goto 命名标号;**

标号及紧随其后的语句形成了一个标号语句，标号语句的一般形式是：

标号：

-语句

### 小测：

**请输出下列的结果，一定要注意switch语句的使用情况和a—的使用情况。**

int main(void)

{

int a;

scanf("%d",&a);

switch (a--)

{

case 1:a+=3;

case 2:a+=3;

default:printf("\*");

}

printf("%d",a);

system("pause");

return 0;

}

**实现计算1-2+3-4+……+99-100**

int main(void)

{

int i=1;

int sum=0;

int t=0;

for (;i<=100;i++)

{

if (i%2==0)

{

t=-i;

sum+=t;

}

else

sum+=i;

}

printf("%d",sum);

system("pause");

return 0;

}

**实现九九乘法表**

int main(void)

{

int i,j;

for (i=1;i<=9;i\*=2)

{

for (j=1;j<=i;j++)

{

printf("%dx%d=%d\t",j,i,i\*j);

}

printf("\n");

}

system("pause");

return 0;

}

## 第四部分：指针

### 指针变量的声明和含义

**int x = 3; //普通变量的定义**

**int \* p1, //指针变量的定义**

**const int y = 3 ；//普通常量**

**int \* const q=&y; //指针常量**

* + **指针变量也是变量;&+变量名表示该变量的地址。**
  + **如果指针变量p的值为变量a的地址，则称p指向a**
  + **指针变量p也是有类型的（它的类型含义：该变量只能存放它所对应的类型去掉一个\*后的那个类型的变量或常量的地址，例如:int \*p,则p只能存放int的变量或常量的地址。）**
  + **P是指针变量，&p为p的地址，\*p表示p指向单元的值**

### 多重指针的声明及其等价形式

**double f = 3.24;**

**double \*p1 = &f;**

**double \*\*p2 = &p1;**

**double \*\*\*p3 = &p2;**

* **p2的等价表达式有：**

**\*p3,&p1,&(\*p2),&(\*\*p3)**

* **\*p2的等价表达式有：**

**&f,p1,\*(&p1),&(\*p1)\*\*p3,\*\*(&p2)**

### 几种特殊的指针

**const int x;**

**int const x;**

**//两者完全等价；我们推荐第一种；**

**int \* const p;**

**const int \* p;**

**//两者天壤之别，第一种const修饰变量p;第二种const修饰变量\*p;**

**多重const指针注意：**

**1. const出现的位置不同意义不同**

**2. const后面紧跟的变量不可改变**

**3.大家应该尽量多的使用const**

* **一个值为空指针字面值的指针变量被称为空指针**
* **例如：**
  + **int \*p = 0;**
  + **double \*p = NULL;**
* **空指针注意：**
* **若声明一个指针变量时，如果不指定它所指向的变量，则指针的指向是不确定的，我们要避免**

### 指针的运算

**P+1指的是p这个指针所在的数组的位置加上1个p所指向的一个类型为int的一个值。**

**例如，一个数组p指向的是第一个地址，而p+1，就是指向了数组的第二个元素。**

## 第五部分：数组

### 数组的定义、初识化和元素的引用

**一维数组的初始化：**

初始化方法一：**全部初始化**

**int** **arr[5]** **=** **{11,** **12,** **13,** **14,** **15};**

**注意：**

1. 初值的个数不能大于数组的长度

int a[3]={1,2,3,4,5};（这种初始化是不正确的）

1. 每个初值的类型最好与数组的类型一致

初始化方法二：**部分元素赋初值**

**int** **b[5]={1,2};//** b[0]=1 b[1]=2b [2]～b[4]都为0 未赋值的均为0

初始化方法三：省略长度赋初值

**一维数组元素的引用**

**初始化数组**

**for(i=0;i<N;i++)**

**a[i]=0;**

**读入数据到数组**

**for(i=0;i<N;i++)**

**scanf(“%d”,&a[i]);**

**数组元素求和**

**for(i=0;i<N;i++)**

**sum += a[i];**

### 二维数组的表示

* **&a 表示数组地址。**
* **a 表示首行（首行元素）地址。**
* **&a[0] 表示首行地址。**
* **a[0] 表示首行首元素地址。**
* **a[0][0] 表示首元素。**
* **&a[0][0] 表示首元素地址。**

### 二维数组得顺序存储

### 指针操作数组

**指针操作一维数组的时候，可以运用等价关系\*(p+i)和p[i];**

**指针**

**p[i][j]的等价形式为：**

* + - **\*(&p[i][j])、\*(p[i] +j)、\*(&p[0][0]+i\*M +j)**

**&p[i][j]的等价形式为：**

* + - **p[i] + j、&p[0][0]+i\*M +j**

**数组指针和指针数组**

**数组指针的表示方法 类型 （\* 变量名）[hang]；**

**指针数组的表示方法 （表示这是一个数组，只不过里面表示的都是指针）**

**类型 \*变量名[];**

### 练习

**表示一下二维数组对角线元素的和**

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

int x[][3]={1,3,5,7,2,4,6,8,9},i,j,s1=0,s2=0;

for (i=0;i<3;i++)

{

for (j=0;j<3;j++)

{

if (i==j) s1+=x[i][j];

if (i+j==2) s2+=x[i][j];

}

}

printf("%d %d",s1,s2);

system("PAUSE");

return 0;

}

**数组进行合并**

#include<stdio.h>

#include<stdlib.h>

int main(void)

{

int a[4]={3,6,9,11};

int b[5]={1,3,8,45,89};

int c[9];

int i,j,t;

i=0;

for (j=0;j<4;j++)

{

c[i]=a[j];

i++;

}

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

{

c[i]=b[j];

i++;

}

for(j=0;j<9;j++)

{

for(i=0;i<9-1-j;i++)

{

if (c[i]>c[i+1])

{

t=c[i];

c[i]=c[i+1];

c[i+1]=t;

}

}

}

for (i=0;i<9;i++)

{

if(c[i]==c[i+1])

printf("");

else

printf("%d\t",c[i]);

}

system("pause");

return 0;

}

**实现数组的逆置**

int main (void)

{

int a[5]={1,2,3,4,5};

int i,j=4,t;

for (i=0;i<5;i++)

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

i=0;

while (i<j)

{

t=a[i];

a[i]=a[j];

a[j]=t;

i++;

j--;

}

for (i=0;i<5;i++)

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

system("pause");

return 0;

}

**实现数组的冒泡排序**

int main (void)

{

int a[5]={5,4,12,7,345};

int i,j,t;

for (i=0;i<5;i++)

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

for (i=0;i<5;i++)

{

for (j=0;j<5-i-1;j++)

{

if (a[j]>a[j+1])

{

t=a[j];

a[j]=a[j+1];

a[j+1]=t;

}

}

}

for (i=0;i<5;i++)

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

system("pause");

return 0;

}

**实现数组的折半查找和顺序查找**

int main (void)

{

int a[5]={5,4,12,7,345};

int i=0;

int x=6;

for (;i<5;i++)

{

if (a[i]==x)

break;

i++;

}

if (i<5)

printf("找到了");

else

printf("sorry");

system("pause");

return 0;

}

int main (void)

{

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

int x,max,min,mid;

min=0;

max=5;

printf("请输入要查找的数");

scanf("%d",&x);

for(;min<=max;)

{

mid=(max+min)/2;

if(x>a[mid])

min=mid+1;//若大于中点，则说明在数组的下半部分，将数组的最小值提前到中点加一

else

if(x<a[mid])//再次验证是否比中点小，不要直接判断是否相等

max=mid-1;//同上一步，将最大值退到中点减一

else

break;//说明相等，退出循环

}

if(min<=max)//判断是从min<max中退出还是从break中退出

printf("查找的数为%d",a[mid]);

else

printf("没有找到该数");

system("pause");

return 0;

}

## 第六部分：函数

### 函数的定义和声明

**函数的定义包括：返回值的类型，函数名，形参列表。**

**形参：定义函数时函数名后面括号中的变量列表**

**实参：调用函数时函数名后面括号中的表达式列表**

**（1）形参是局部变量，调用时分配内存，调用结束后系统自动回收形参所占内存。**

**（2）实参可以是任意合法的常量、变量、表达式。**

**（3）实参与形参个数一致，类型一致（可能会发生类型转换），顺序一致。**

**返回值**

**1.如果函数没有返回值一定要注意写void类型.不需return.**

**2.如果函数有返回值的话一定要注意返回值的类型与接收函数返回值变量的类型.**

**3.接收函数返回的变量的类型需同返回值类型相同**

**4.一个函数可能有零个或者多个参数，最多只能由一个返回值.**

**5.函数返回值不能是局部变量的地址！**

### 函数的调用（在写函数时，尽量使用函数原型）

1. **主调程序保护现场**
2. **给格式参数（如果有）分配内存，将实参传给形参，将程序的控制权交给被调函数**
3. **执行被调函数语句**
4. **保存返回值（如果有）到某处，这个值所有是可见的，释放被调函数在栈区分配的空间**
5. **将程序控制权交给主调函数，主调函数取得返回值**

交换两个值

#include<stdio.h>

#include<stdlib.h>

void swap(int \*x,int \*y)

{

int z;

z=\*x;

\*x=\*y;

\*y=z;

}

int main(void)

{

int a=3,b=5;

printf("交换之前a=%d\tb=%d\n",a,b);

swap(&a,&b);

printf("交换之后a=%d\tb=%d\n",a,b);

system("pause");

return 0;

}

### 一维数组作形参的过程

**一维数组做形参时，实际参数为&x[0],或者是x，此时的形式参数是\*a，a[ ]。**

**二维数组做形参时，此时的形式参数是a[ ][4],(\*a)[4]。**

### 在数组中实现元素的查找。

#include<stdio.h>

#include<stdlib.h>

int searchinArray(int p[],const int len,const int x)

{

int i=0;

if (len<=0||len>8)

return -1;

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

{

if (x==\*(p+i))

return i;

}

return -1;

}

int main(void)

{

int a[8]={1,34,345,3,934,59,35,56};

int len=8;

int x=3;

int index=searchinArray(&a[0],len,x);

if (-1==index)

printf("在数组中未找到%d\n",x);

else

printf("%d在数组中的下标量是%d\n",x,index);

system("pause");

return 0;

}

通过这个例子我们可以看到：

1. 一个函数里面可以有一个return，两个return，也可以没有return。
2. 一个函数里面最多且只能执行一条return语句。

### 在数组中实现排序

#include<stdio.h>

#include<stdlib.h>

void sort(int p[],int len)

{

int i,j;

for (i=0;i<len-1;i++)

{

for (j=0;j<len-1-i;j++)

{

if (p[j]>p[j+1])

{

int t=p[j];

p[j]=p[j+1];

p[j+1]=t;

}

}

}

}

void print(int p[],int len)

{

int t=0;

while (t<len)

{

printf("%d\t",p[t++]);

}

}

int main(void)

{

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

print(a,8);

printf("\n");

sort(a,8);

print(a,8);

system("pause");

return 0;

}

### 动手练习：请定义一个函数实现数组的元素的逆置

#include<stdio.h>

#include<stdlib.h>

void inverse(int p[],int len)

{

int i=0;

while (i<len)

{

int t=p[i];

p[i]=p[len-1];

p[len-1]=t;

i++;

len--;

}

}

void print(int p[],int len)

{

int t=0;

while (t<len)

printf("%d\t",p[t++]);

}

int main(void)

{

int a[5]={3,4,6,1,2};

print(a,5);

printf("\n");

inverse(a,5);

print(a,5);

system("pause");

return 0;

}

### 编程求n的阶乘

int f(int n)

{

int s=n;

if (n==1)

return 1;

else

{

s=s\*f(n-1);

return s;

}

}

**递归函数**

任何一个递归调用程序必须包括两部分

（1）递归调用继续的过程 （2）递归调用结束的过程

int add(int p[],int len)

{

if (p[0]==p[len-1])

return p[len-1];

else

{

int t=p[0]+add(&p[1],len-1);

return t;

}

}

### 生存周期和作用域

**生存周期的概念：**

**变量保持所分配存储空间的时间，称为变量的存储期间或生存期。**

**静态生存周期**

若某对象在程序开始执行之前即分配到存储空间，而且保持到程序终止，则称该对象具有**静态生存期**。（所有函数、在顶层声明的变量、静态变量。）

**本地生存期**

如果对象在进入块或函数时分配到存储空间，且在退出块或函数时删除，则称该对象具有**本地生存期**。C 语言把具有本地生存期的变量称为自动变量。

### 声明作用域

**声明作用域**是声明变量有效的区域(空间)

顶层标示符（从声明点到源程序结束之间的文本区域）

函数定义中的形式参数（从声明点到函数体结束之间的文本区域）

块内标示符（从块中的声明点到块结束之间的文本区域）

标号标示符（整个函数体）

**全局变量和局部变量**

称在顶层声明的变量为全局变量，与全局变量相对的是局部变量（如：函数体内的变量、形式参数、块内的变量）。

C语言程序中，**在同一作用域内不允许有同名的变量、函数等**，**但在不同的作用域内，可能会出现同名的变量。**

若使用的变量是在不同作用域内的同名变量，则**作用域小的变量优先。**

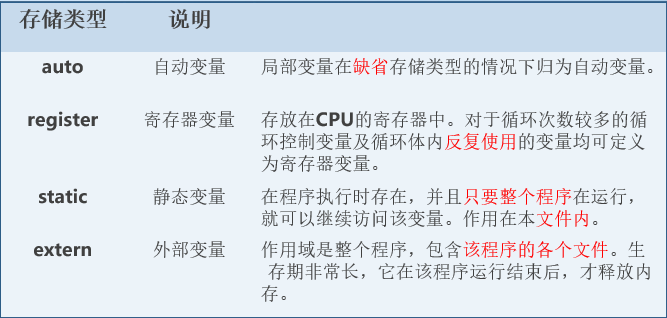
**总结：**

**1.局部变量的作用域为定义它的块儿**

**2.全局变量的作用域为从定义它开始到程序结束**

### 存储类型说明符

**存储类说明符用于确定声明对象的生存期**（typedef 除外），**存储类说明符出现在声明说明符中，一个声明说明符中最多允许一个存储类说明符。存储类说明符包括：**



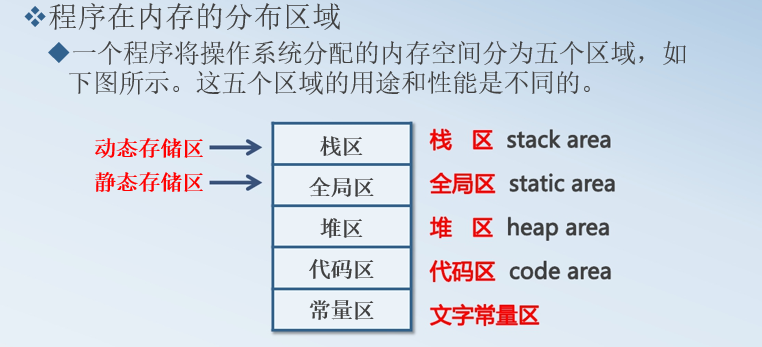
注意static只是初始化一次，并且如果没有初始化，默认是0；

### 程序在内存中的分布区域

**库函数中**有一类函数**可以让我们自己分配内存**

**Malloc** **calloc** **free**

应包含**malloc.h或stdlib.h**



**使用malloc、calloc分配的内存一定要自己用free释放。**

**用malloc、calloc进行内存分配，一定要检测成功与否。**

**malloc、calloc函数的返回值是void类型，一定要根据需要进行强制类型转换。**

### 指针函数和函数指针

 当一个函数声明其返回值为一个指针时，实际上就是返回一个地址给调用函数，以用于需要指针或地址的表达式中。  
    格式：  
         类型说明符 \* 函数名(参数)

函数指针是指向函数的指针变量，即本质是一个指针变量。

指向函数的指针包含了函数的地址，可以通过它来调用函数。声明格式如下：  
        类型说明符 (\*函数名)(参数)

### 字符串的比较以及排序

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int main (void)

{

char \*a[5]={"abcd","1889","efga","666a","bcde"};

char \*temp;

int i,j;

for (i=0;i<5;i++)

{

for (j=0;j<5-i-1;j++)

{

if (strcmp(a[j],a[j+1])>0)

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

for(i=0;i<5;i++)

printf("%s\n",a[i]);

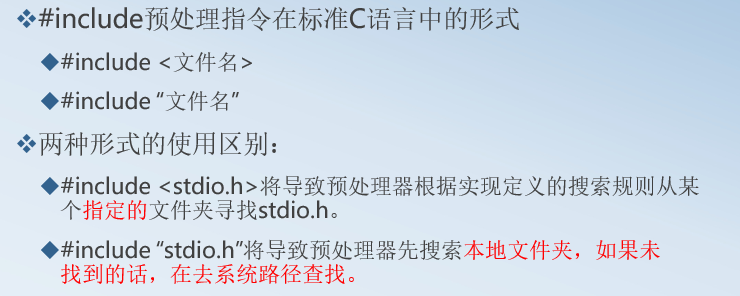
system("pause");

return 0;

}

## 第七部分：预处理和宏

### 文件包含

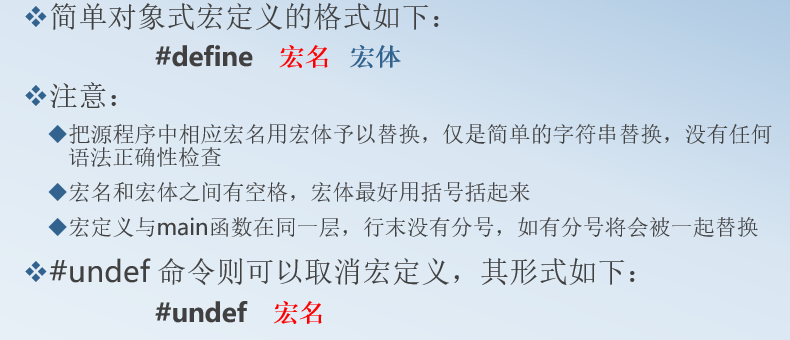


**注意：在<>或“”中，有时会出现 .. 或 .**

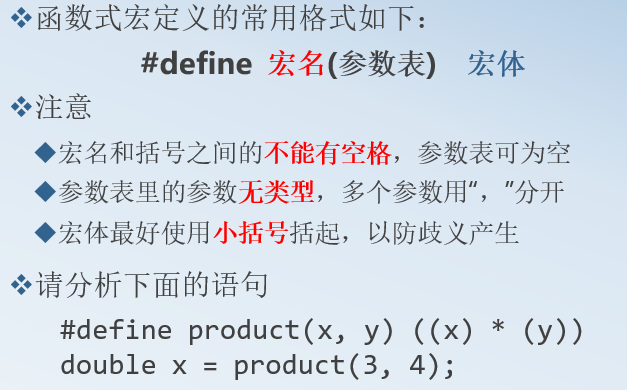
**.. 表示当前文件所在目录的上一级目录；**

**. 则表示当前文件所在的目录.**

### 对象宏（无参宏）

****

### 函数式宏（有参宏）



#include<stdio.h>

#include<stdlib.h>

#define F(x) x-2

#define D(x) x\*F(x)

int main(void)

{

printf("%d,%d",D(3),D(D(3)));

system("pause");

return 0;

}

**要求功能简单，代码短小，则可以使用函数宏；如果要求效率较高，可以采用函数宏，函数宏没有函数调用的代价。**

### 条件编译

预处理器条件指令（#if、#else、#elif、#endif 等）允许预处理器根据计算条件处理和替换宏。

**#ifdef、#ifndef用于测试一个名称是否被定义为预处理宏。**

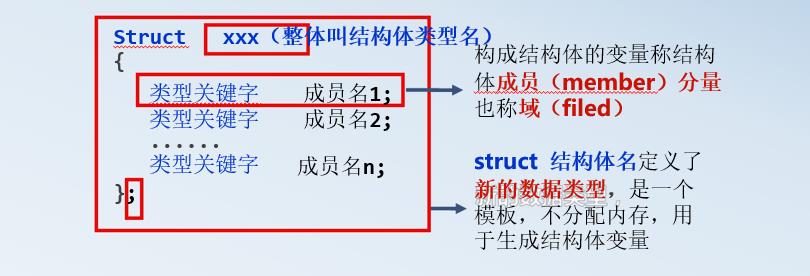
**主要用来解决头文件重复包含的问题。**

## 第八部分：结构体

### 结构体

**结构体是一种自定义数据类型**，结构体**变量代表的结构体类型的数据对象作为一个整体存储在内存中。**

**结构体类型定义的基本语法形式为：**



#include <stdio.h>

#include <stdlib.h>

typedef struct student

{

int no;

char name[20];

float score;

}STU;

/\*输出成绩最高的人的学号\*/

int f(STU \*x,const int len)

{

int i=0;

int no;

float s=x[0].score;

while (i<len)

{

if (x[i].score>s)

{

no=i;

s=x[i].score;

}

i++;

}

return x[no].no;

}

int main(void)

{

STU stus[5]={{11,"zhangsan",80.0},{12,"lisi",59.5},{13,"wangwu",94.5},{14,"zhaoliu",98.5},{15,"dingqi",86.0}};

int s;

s=f(stus,5);

printf("%d",s);

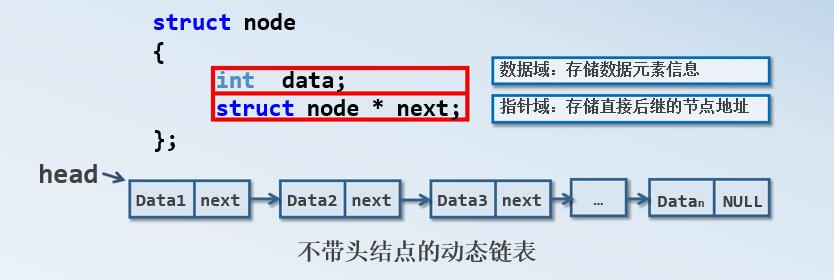
system("pause");

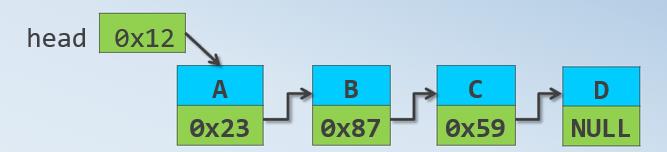
return 0;

}

### 动态链表的构成：

动态链表有一个或者多个结点构成，每个节点都是一个结构体对象。每个结点有数据域和指针域(关系)





**注意：**① 链表有一个**头指针**，存放第一个结点的地址

② 每一个**结点**由数据域和指针域构成

③ 最后一个结点指针域为NULL，称为“**表尾**”

④ 结点通常就是一个结构体变量

**链表特点：**① 链表中各元素可以不是连续存放的.

② 要找某结点必须先找到上个结点因此必须提供头指针.

③ 创建结点就是创建结构体和利用指针做成员.

### 动态链表

动态链表的结点是临时生成的。

可以根据需要添加或者删除元素。

程序员手工分配内存与释放内存（堆区）。

* **Malloc**
* **Calloc**
* **Realloc**
* **free**

### 链表程序

#### 创建一个链表并对它进行遍历

方法一：头查法

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

typedef struct Node

{

int x;

struct Node \*y;

}Node;

/\*创建一个链表\*/

Node \* f(int n)

{

Node \*p;

int i=0;

Node \*h=NULL;

while (i<n)

{

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

printf("请输入第%d个元素",n-i);

scanf("%d",&(p->x));

p->y=h;

h=p;

i++;

}

return h;

}

/\*实现对链表的遍历\*/

void visit(Node \* l)

{

while (l!=NULL)

{

printf("%d\t",l->x);

l=l->y;

}

}

/\*检验\*/

int main(void)

{

Node \*h;

h=f(5);

visit(h);

system("pause");

return 0;

}

**方法二：尾查法**

Node \*f(int n)

{

Node \*p;

Node \*h=NULL;

Node \*l;

int i=0;

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

printf("请输入第%d个数",i+1);

scanf("%d",&(p->data));

h=p;

l=h;

i++;

while (i<n)

{

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

printf("请输入第%d个数",i+1);

scanf("%d",&(p->data));

h->next=p;

h=p;

i++;

}

h->next=NULL;

return l;

}

#### 求链表结点的个数

int number(Node \*l)

{

int i=0;

while (l!=NULL)

{

i++;**//若是求和则改为i=i+l->x;如果是正数之和,则需要进行判断。**

l=l->y;

}

return i;

}

#### 请返回倒数第k个元素

int ff(Node \*l,int k,int n)

{

int i=0;

while (l!=NULL)

{

i++;

if (i==n-k+1)

return l->x;

l=l->y;

}

}

#### 链表的查找和更新

问题描述：在链表中找到a元素将a元素找到后变换为b元素，并将更新后链表输出。

int ff(Node \*l,int a,int b)

{

while ((l!=NULL)&&(l->x!=a))

{

l=l->y;

}

if (l!=NULL)

{

l->x=b;

return 1;

}

return 0;

}

#### 链表的逆置

typedef struct Node

{

int x;

struct Node \*y;

}Node;

Node \* inverse(Node \*l)

{

Node \*ph=NULL;

while (l!=NULL)

{

Node \*q=l;

l=l->y;

q->y=ph;

ph=q;

}

return ph;

}

#### 链表的插入

**问题一：在第k个结点的前面插入一个结点，数据域是x**

Node \*ff(Node \*l,int k,int x)

{

Node \*h=l;

Node \*p;

int i=1;

if (k==1)

{

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

p->x=x;

p->y=l;

l=p;

return l;

}

else

{

while (l!=NULL&&i<k-1)

{

l=l->y;

i++;

}

if (l!=NULL)

{

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

p->x=x;

p->y=l->y;

l->y=p;

return h;

}

else

printf("不存在第k个结点！");

}

}

**问题二：在链表中在元素x之后插入一个结点元素值为y。**

Node \*ff(Node \*l,int x,int y)

{

Node \*h=l;

Node \*p;

while (l!=NULL&&l->x!=x)

{

l=l->y;

}

if (l!=NULL)

{

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

p->x=y;

p->y=l->y;

l->y=p;

return h;

}

}

#### 链表的删除

**问题一：删除第k个结点。**

Node \*ff(Node \*l,int k)

{

Node \*h=l;

Node \*p;

int i=1;

if (k==1)

{

p=l->y;

free(l);

return p;

}

else

{

while (l!=NULL&&i<k-1)

{

l=l->y;

i++;

}

if (l!=NULL)

{

p=l->y;

l->y=p->y;

free(p);

return h;

}

else

printf("不存在第k个结点！");

}

}

**问题二：在链表中删除元素x。**

Node \* f(int n)

{

Node \*p;

int i=0;

Node \*h=NULL;

while (i<n)

{

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

printf("请输入第%d个元素",n-i);

scanf("%d",&(p->x));

p->y=h;

h=p;

i++;

}

return h;

}

Node \*ff(Node \*l,int x)

{

Node \*h=l;

Node \*p;

int k;

int i=0;

while (l!=NULL)

{

i++;

if (l->x==x)

k=i;

l=l->y;

}

if(k==1)

{

p=l->y;

free(l);

return p;

}

else

{

int j=1;

while (l!=NULL&&j<k-1)

{

l=l->y;

j++;

}

if (l!=NULL)

{

p=l->y;

l->y=p->y;

free(p);

return h;

}

else

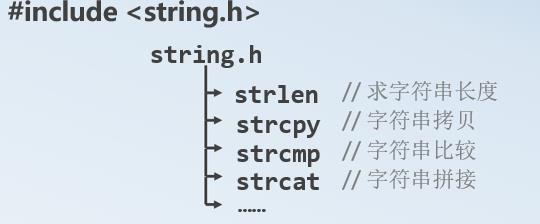
printf("不存在第k个结点！");

}

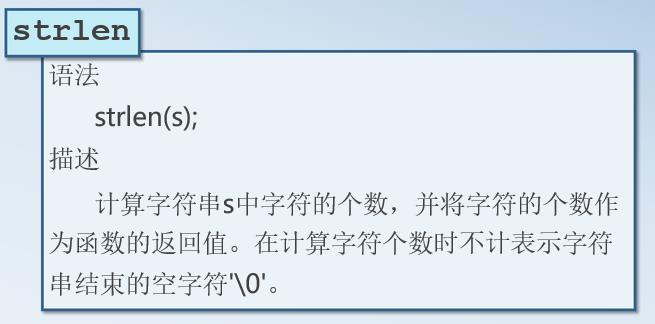
}

## 第九部分：字符串函数

### 字符串函数



### 1、strlen



#include<stdio.h>

#include<stdlib.h>

int strlen(char \*p);

int main(void)

{

char a[]="abcdefg";

printf("%d",strlen(a));

system("pause");

return 0;

}

int strlen(char \*p)

{

int s=0;

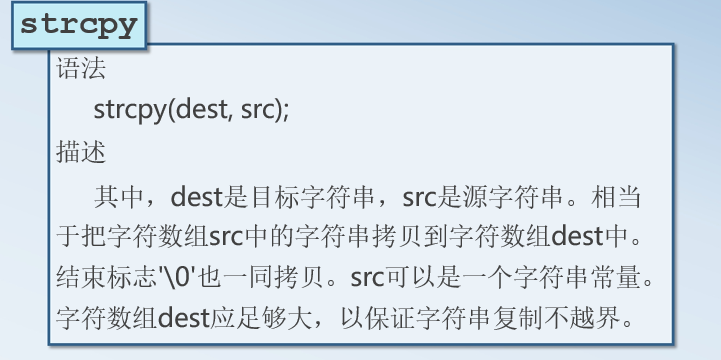
while(\*(p+s)!='\0')

s++;

return s;

}

### 2、strcpy



#include<stdio.h>

#include<stdlib.h>

char \*strcpy(char \*x,char \*y)

{

int j=strlen(y);

int i=0;

while (i<j)

{

\*(x+i)=y[i];

i++;

}

x[i]='\0';

return x;

}

int main(void)

{

char y[]="abc";

char x[100];

\*strcpy(x,y);

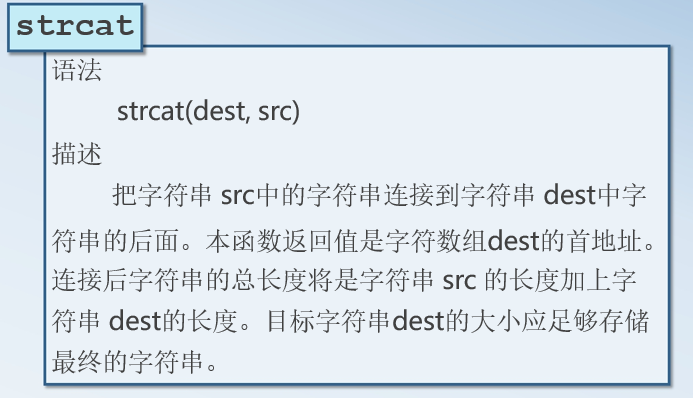
printf("%s",x);

system("pause");

return 0;

}

### 3、strcat



#include<stdio.h>

#include<stdlib.h>

char \*strcat(char \*x,char \*y)

{

int i=strlen(x);

while (\*y!='\0')

{

x[i]=\*y;

y++;

i++;

}

x[i+1]='\0';

return x;

}

int main(void)

{

char y[]="abc";

char x[]="welcome to hebei normal university";

\*strcat(x,y);

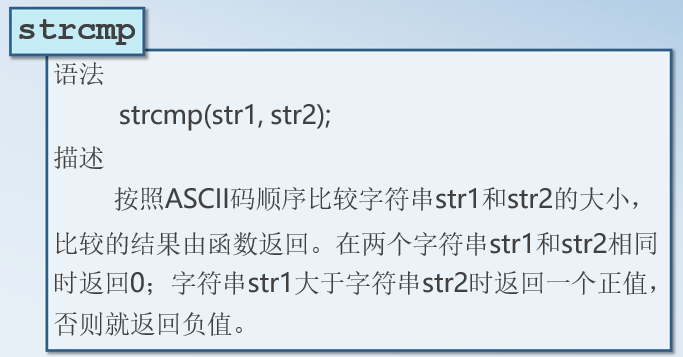
printf("%s",x);

system("pause");

return 0;

}

### 4、strcmp



#include<stdio.h>

#include<stdlib.h>

int strcmp(char \*x,char \*y)

{

int i=0,j=0;

int lena=strlen(x);

int lenb=strlen(y);

while ((i<lena)&&(j<lenb))

{

if (x[i]>y[i])

return 1;

else if (x[i]<y[i])

return -1;

i++;j++;

}

if ((i==lena)&&(j==lenb))

return 0;

else if (i<lena)

return 1;

else

return -1;

}

int main(void)

{

char y[]="abc";

char x[]="abacome to hebei normal university";

printf("%d",strcmp(x,y));

system("pause");

return 0;

}