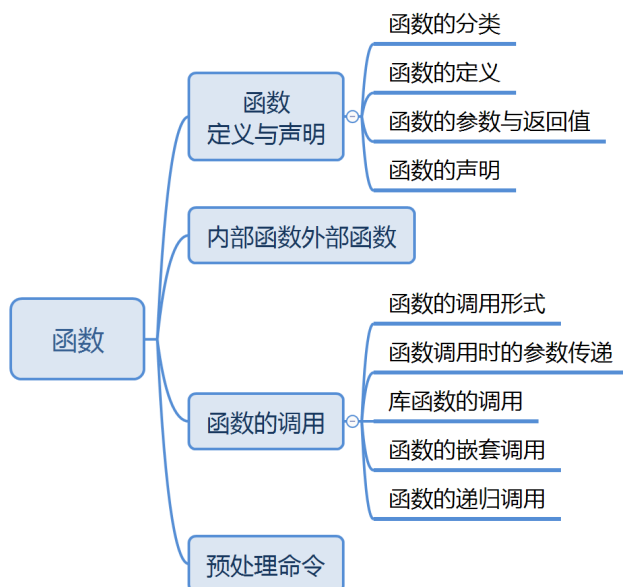




专题八 函数

【内容预览】



【知识清单】

8.1、函数的定义与声明

8.1.1、函数的分类

1. 从用户使用的角度看

- (1) **标准函数，即库函数**：由 C 语言标准库提供，用户无须定义，可以直接使用它们。只需在程序前包含该函数原型的头文件即可。如 `printf()`、`scanf()` 等函数均属此类。
- (2) **用户自定义函数**：由用户自己定义，用以解决用户的专门需要。

2. 从函数形式的角度看

- (1) **无参函数**：函数定义及函数调用中均不带参数。主调函数和被调函数之间不进行参数传递。此类函数通常用来完成一组指定的功能，可以返回或不返回函数值。
- (2) **有参函数**：也称带参函数。在调用函数时，主调函数和被调函数之间有数据传递。即主调函数可以将数据传给被调函数使用，被调函数中的数据也可以带回来供主调函数使用。

8.1.2、函数的定义

1. 无参函数的定义

定义形式：类型标识符 函数名 () //函数首部
{
.....

```
} //函数体
```

(1) 类型标识符指明本函数的类型，也就是函数返回值的数据类型。`int` 型函数定义时可以省略类型标识符 `int`, **int 是有返回值函数的缺省类型：即若函数无类型标识符，则返回值进行强制转换成 `int` 型**。另外，函数也可以无返回值，其类型标识符为 `void`，又称“空类型函数”，此函数不向主调函数返回值，主调函数也禁止使用此函数的返回值。

(2) 函数名是一个标识符，它是唯一的，**两个函数不能重名**。函数名后有一对括号，即使是无参函数，其中无参数，但**括号不能省，这是函数的标志**。

如：#include<stdio.h>

```
void printstar()
{   printf("*****\n");
}
int main()
{   printstar();    //调用 printstar 函数
    return 0;
}
```

2. 有参函数的定义

定义形式：函数类型 函数名（形参类型说明表）

```
{ ..... }
```

在形参类型说明表中，给出的参数称为形参，它们可以是各种类型的变量，**各参数之间用逗号间隔**。

如：double power(int x,int n)

3. 函数的公共特点：

- (1) 每个函数都有唯一的名字，用这个名字，程序可以转去执行该函数所包含的语句，这种操作称为**调用函数**。
- (2) C 语言中，所有函数的定义都是独立的、封闭的、包括主函数在内。也就是说，在一个函数体内，不能再定义另一个函数，即**不能嵌套定义，但允许函数之间互相调用**。
- (3) **C 语言源程序中有且只有一个 `main()` 函数。`main()` 函数可以调用其它函数，但不允许被其它函数调用。因此，C 语言程序的执行总数从 `main()` 函数开始，完成对其它函数的调用后再返回到 `main()` 函数，最后在 `main()` 函数中结束整个程序的运行。**
- (4) 要注意的是，**C 语言源程序可以分别放在不同的文件里，因而一个 C 语言程序可以有一个或多个源程序文件，每个源程序文件可以由一个或多个函数组成。**

8.1.3、函数的参数和返回值

1. 函数的形参和实参

形参是在定义函数时函数名后面括弧中的变量名；实参是主调函数调用一个函数时，函数名后括号中的参数。

```
int max(int x,int y);    //x,y 为形参
int main()
{   int a,b,c;
    scanf("%d%d",&a,&b);
    c=max(a,b);          //a,b 为实参
    printf("%d",c);
    return 0;
}
```

实参和形参的特点

- (1) **形参只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。**因此，形参只

例如: `#include<stdio.h>`

```
void fun(int x,int y)
{
    int a,b;
    a=1,b=2;
    printf("%d\t%d\n",a,b);
}

void main()
{
    int a,b;
    a=3;b=4;
    fun(a,b);
    printf("%d\t%d\n",a,b);
}
```

运行结果为：

1	2
3	4

- ### (3) 函数参数表的一致性原则

- a) 函数调用中实参的**数目**必须与函数定义形参表中形参的**数目一致**，**按照自己所需功能来安排顺序**。

例如: `int fun(int x)`

$$\{\dots\}$$

```
main()
```

{int x,y;fun(x,y);}是错误的。

- b) 函数调用中实参的数据类型必须与函数定义形参表中对应形参的数据类型相同或赋值兼容。

- (4) 函数调用中发生的**数据传递是单向的**。即**只能把实参的值传递给形参，而不能把形参的值反向传递给实参**。因此在函数调用过程中，形参的值发生改变，而实参中的值不会变化。（此处在后面数据的调用方式处会做详细说明）

2. 函数的返回值

定义：函数被调用时，执行函数体中的程序段，需要返回给主调函数的结果值。

形式 1: return 表达式;

如: `int max(int x,int y)`

```
{  int z;  
  z=(x>y)?x:y;  
  return z; }
```

形式 2: `return(表达式);`

如: `int max(int x,int y)`

```
{ return ((x>y)?x:y); }
```

- (1) 函数的值只能通过 `return` 语句返回给主调函数。在函数中允许有多个 `return` 语句，但每次调用时只能有一个 `return` 语句被执行。这种情况通常出现在 `if-else` 语句中，表示在不同的条件下返回不同的值。

例如：

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

```
int fun(int a,int b)
```

 $\{$

```
if(b==0)
```

```

    return a;
    else return(fun(--a,--b));
}
void main()
{
    printf("%d\n",fun(4,2));
}

```

8.1.4、函数的声明

1. 函数声明的形式:

类型说明符 被调函数名 (类型 形参 1, 类型 形参 2, ……);

如 `int max(int a,int b);`

或

类型说明符 被调函数名 (类型, 类型, 类型……); (分号不可少)

如 `int max(int,int);`

2. 进行函数声明的原因：在 C 语言源程序中函数的排列位置是任意的，即允许主调函数排列在被调函数前面，也允许被调函数排列在主调函数前面。C 语言规定，函数先定义后调用，若对某函数先调用后定义，则在调用该函数之前先对该被调用函数进行声明。

```
如:  int fun(...){...}      int fun(...);
      main(){...}           main(){...}      //这两种方法都是可以的
      int fun(...){...}
```

3. 以下几种情况可以省略对被调函数的函数声明

(1) 当被调函数的函数定义出现在主调函数之前,或在所有函数定义之前,在函数外预先对各个函数进行了声明,则在后面的各主调函数中,可不再对被调函数声明。

(2) 对库函数的调用不需要再作说明, 但必须把该函数的头文件用 `include` 命令包含在源文件前部。如 `#include<math.h>` (指的是调用数学库函数)

【解题技巧】

例 8.1.1 若有以下函数首部 `int fun (double x[10],int *n)`,则下列针对此函数的函数声明语句中正确的是: ()

A)int fun(double x,int *n);

```
B)int fun(double,int);
```

```
C)int fun(double *x,int n);
```

```
D)int fun(double *,int *);
```

正解: D

分析：实际上，C 编译系统都是将形参数组作为指针变量来处理的。`int fun(double x[10], int *n);`相当于 `int fun(double *x, int *n);`

例 8.1.2 如果有函数 void func(void);那么下面哪个语句是合法的 ()

A)x=func();

B)while(1) func();

C)if(func()){puts("yes");}

D)while(func());

正解: B

分析: 因为函数 func 是无返回值的函数, 因此, 无法在主函数中调用函数 func 的值, 而 A、C、D 都需要用到 func 的值, 故选 B。

8.2、函数的调用

8.2.1、函数的调用形式

1. 调用的一般形式: 函数名 (实际参数表); //参数可为常量、变量或表达式, 各实参间用逗号隔开。
无参函数的形式: 函数名 (); 如: getchar();
2. 调用函数方式
 - (1) 函数语句: 函数调用的一般形式加上分号即构成函数语句。 如: printf("%d",a);
 - (2) 函数表达式: 函数调用作为表达式中的一项出现在表达式中, 用函数返回值参与表达式的计算。如:
z=2*max(x,y);
 - (3) 函数实参: 函数调用作为另 1 个函数调用的实参出现。 如: z=max(max(a,b),c);
3. 在主调函数执行过程中, 一旦遇到函数调用, 系统首先计算实参的值并为每个形参分配存储单元, 然后把实参值复制到对应形参的存储单元中。

8.2.2、函数调用时的参数传递

1. 按值调用: 当形参和实参是基本类型变量时, 参数传递的方式是按值调用。
过程: a)函数被调用时, 形参变量被创建。
b)形参变量的值从实参中一一对应复制而得。
c)当从被调用的函数返回主调函数时, 形参所占内存单元被释放。

由于形参和实参在内存中各自占用不同的内存单元, 所以形参的数值变化不会导致实参的相应变化。按值调用采用的就是一种“单向”的值传递方式, 即由实参向形参进行值传递。

(1) 变量作函数参数

如: #include<stdio.h>

```
void swap(int a,int b)
```

```
{    int t;
```

```
    t=a,a=b,b=t;
```

```
}
```

```
int main()
```

```
{    int x=7,y=11;
```

```
    swap(x,y);
```

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

```
    return 0; }    输出结果为: 7 11    //x,y 的值并未发生交换
```

(2) 数组元素做函数实参与普通变量没有区别

2. 按地址调用

当形参和实参是指针 (地址) 类型的数据时, 实参到形参的参数传递是地址值的传递, 称为按地址调用。

特点: 主调函数中的实参和被调函数中的形参仍占用不同的存储单元, 但是形参和实参所指向的是同一地址, 对形参所指地址上的内容的操作会影响实参所指向的地址上的内容。此时, 函数也无须通过

return 语句返回函数值了。

(1) 指针作函数参数

如: #include<stdio.h>

```
void swap(int *a,int *b)
{
    int t;
    t=*a,*a=*b,*b=t;
}
```

```
int main()
```

```
{
    int x=7,y=11;
    swap(&x,&y);
    printf("%d %d\n",x,y);
    return 0; }    输出结果为: 11 7    //x,y 的值被交换
```

(2) 数组名作参数

数组名是数组的地址, 属于是地址传递。 如: **float sum(float a[]);**

但是在编译时是将数组 a 按指针变量处理的, 所以上面定义相当于

float sum(float *a);

用数组名作为函数参数时还应注意以下问题:

- a) 形参数组和实参数组的类型必须一致, 否则将引起错误, 因为不同类型数据存储时所占字节数不同。
- b) 形参数组可以不指定大小, 实参数组和形参数组的大小可以不一致, 编译时不做检查。
- c) 由于是地址传递, 所以**实参和形参指向内存单元的同一个地址, 形式上是两个数组, 而实质上是同一个数组**, 所以形参中所指各元素值的变化, 实际上就是实参所指的各元素的变化。

8.2.3、库函数的调用

C 语言提供了丰富的库函数, 包括常用的数学函数, 如求绝对值的 fabs() 函数, 求正弦值的 sin() 函数, 也包括对字符和字符串进行处理的函数, 还包括进行输入和输出的各种函数等。

1. 调用 C 语言标准库函数时的格式: #include<头文件> 或 #include“头文件”

include 命令必须以 # 开头, 系统提供的头文件以 .h 作为文件后缀, include 命令不是 C 的语句, 所以不能在最后加分号。

2. 常用头文件

- (1) **stdio.h** 输入/输出库函数
- (2) **math.h、stdlib.h、float.h** 数学库函数
- (3) **string.h** 内存缓冲区和字符串处理库函数
- (4) **time.h** 时间库函数
- (5) **malloc.h、stdlib.h** 内存动态分配库函数

8.2.4、函数的嵌套调用

当一个函数作为被调函数时, 它也可以作为另一个函数的主调函数, 而它的被调函数又可以调用其它函数, 这就是函数的**嵌套调用**。

如: 用函数调用的方法计算 $sum=1!+2!+3!+\dots+10!$ 。

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

```
float sum(int);
```

```
float fac(int);
```

```
int main()
```

```
{    float sum(int);           //主函数调用 sum() 函数
```

```
    float x;
```

```
    x=sum(10);
```

```

        printf("a=%10.1f",x);
        return 0; }
float sum(int n)
{ float fac(int);    //sum()函数调用 fac()函数
  .....
}
float fac(int a)
{ ..... }

```

8.2.5、函数的递归调用

当一个函数直接或间接地调用它自身时，称为函数的递归调用。

1. 用递归函数解决问题，应满足两个条件：
 - (1) 要直接或间接地调用它本身。
 - (2) 应有使递归结束的条件。
2. 说明：C 编译系统对递归函数的自调用次数没有限制。
3. 每调用函数一次，在内存栈区分配空间，用于存放函数变量、返回值等信息，所以递归次数过多，可能引起堆栈溢出。

【解题技巧】

例 8.2.1 函数 `int reverse(int *arr,int size)` 可以将一个整型数组逆序，如果在 `main` 函数中定义了整型数组 `int arr[10]`，那么可以正确调用 `reverse` 函数的方式是 ()

- | | |
|--|--|
| A) <code>reverse(&arr[0],10);</code> | B) <code>reverse(int arr[10],int 10);</code> |
| C) <code>reverse(arr[10],10);</code> | D) <code>reverse(arr[0],10);</code> |

正解：A

分析：函数 `reverse` 的第一个形参是 `int` 型的指针，第二个是 `int` 型变量，故此题选 A。

例 8.2.2 有以下程序：

```

#include<stdio.h>
void fun1(char *p)
{
    char *q;
    q=p;
    while(*q!='\0')
    {
        (*q)++;
        q++;
    }
}
void main()
{
    char a[]={"Program"},*p;
    p=&a[3];
    fun1(p);
    printf("%s\n",a);
}

```

```
}
```

程序执行后的输出结果是

A)Prphsbn

B)Prohsbn

C)Progsbn

D)Promgram

正解：B

分析：`p=&a[3]`, 表示 `p` 指向字符 'g' ; `(*q)++` 是 `q` 所指的字符加 1, `q++` 就是指针移到下一个字符。因此 B 选项正确。

8.3、内部函数与外部函数

8.3.1、内部函数

1 个函数只能在所定义的源文件中其它函数调用, 而不能被程序中其它源文件中的函数所调用, 这样的函数称为内部函数。

形式: **static** 类型定义符 函数名 (形参表); 如: `static float function(int x,int y);`

意义：不同的人编写不同的函数时, 不用担心自己定义的函数是否会与其他文件中的函数同名, 因为在不同的文件中同名的内部函数, 互不干扰。

8.3.2、外部函数

1 个函数可以被所定义的源文件中被其它函数调用, 还可以被程序中其它源文件中的函数所调用, 这样的函数称为外部函数。

形式: **extern** 类型定义符 函数名 (参数表);

如: `//file1.c`

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

```
int main()
```

```
{    extern int max(int,int);
```

```
    ..... }
```

`//file2.c`

```
int max(int x,int y)
```

```
{    ..... }
```

通过此例看出, 使用 **extern** 声明就能够在文件中调用其它文件中定义的函数, 或者说把该函数的作用域扩展到本项目程序。C 语言允许在声明函数时省略 **extern**。

8.4、预处理命令

8.4.1、宏定义

在程序中以 **#include**、**#define** 开头的行, 这些特殊的行称为预处理控制指令。预处理命令可以出现在程序中的任何位置, 作用域是从出现点到所在源程序的末尾。编译预处理的主要功能有: **宏定义**, **文件包含**和**条件编译**。

1. 宏定义格式

(1) 不带参数的宏定义

格式: **#define** 宏名 字符串 如: `#define SIZE 20`

a) 在程序中用“”括起来的字符串中, 即使有的字符串与宏名相同, 也不进行替换。

b) 宏定义只是一种简单的字符替换, 不进行语法检查, 只有在编译已被宏展开后的源程序时才会

发现语法错误并报错。

- c) `#define` 命令出现在函数的外部，宏名的有效范围为定义命令之后到本文件结束。可以用 `#undef` 命令终止宏定义的作用域。
- d) 宏定义与变量定义不同，它只作字符替换，不分配内存空间。
- e) 宏定义可以嵌套使用。

(2) 带参数的宏定义

格式：`#define` 宏名(形参表) 字符串 如：`#define S(r) r*r`

- a) 宏定义时宏名和括号之间没有空格，若有空格则会把空格后的所有字符都看成宏体。

- b) 建议带运算符的宏体和形参用 `()` 括起来。 如：`#define S(r) r*r`

当参数为 `a+2` 时，`S(a+2)=a+2*a+2`；（与 `(a+2*a+2)`

不同）

(3) 带参数的宏和有参函数的区别

- a) 函数调用是在程序运行时处理，分配临时的内存单元；宏展开是在编译时进行的，在展开时不分配内存单元。
- b) 对函数的形参和实参都要定义类型，且要求一致；宏不存在类型问题，宏名无类型，其参数也无类型。
- c) 函数只可得到一个返回值；宏可以设法得到几个结果。
- d) 函数调用不会使源程序变长；宏展开会使源程序变长。函数调用占用运行时间；宏展开不占运行时间，只占编译时间。

2. 文件包含

文件包含是指一个源文件可以将另外一个源文件的全部内容包含进来。

(1) 格式一：`#include`“文件标识”

文件标识可以包含文件路径（即文件在哪个目录下），使用时系统先在引用被包含文件的源文件所在的目录下寻找被包含文件。如果找不到，系统再按指定的标准方式检索其他目录，直到找到为止。此格式通常用于用户自己编写的文件。

(2) 格式二：`#include`<文件名>

系统只按规定的标准方式检索文件目录，通常使用系统提供的标准头文件时用 < >。

说明：

- a) 文件包含的作用就是在编译预处理时将包含的文件的全部内容复制并插入 `#include` 命令处。
- b) 一个 `include` 命令只能指定一个被包含文件。如果有 `n` 个被包含文件，则需要用 `n` 个 `include` 命令，且一个命令占一行。
- c) 使用文件包含时，在被包含文件中决不能有 `main()` 函数。
- d) 文件包含可以嵌套使用。
- e) 被包含文件中的全局变量在其所在的文件中有效。

3. 条件编译

所谓条件编译，是对部分内容指定编译的条件，使其只在满足一定的条件时才进行编译。条件编译也是一种编译预处理命令，它主要是为了解决在不同系统条件下 C 语言程序的可移植性问题，使程序能在不同系统环境下，选择合适的不同程序段进行编译，生成不同的目标函数代码文件。

条件编译命令有如下三种形式：

(1) `#ifdef` 标识符

程序段 1

#else

程序段 2

#endif

其中，标识符一般是用#define 定义的宏名，也可缺省#else 部分。

功能：若标识符定义过，则对程序段 1 进行编译，否则对程序段 2 进行编译。

(2) #ifndef 标识符

程序段 1

#else

程序段 2

#endif

功能：若标识符未定义过，则对程序段 1 进行编译，否则对程序段 2 进行编译。

(3) #if 表达式

程序段 1

#else

程序段 2

#endif

功能：当表达式的值非 0 时，则对程序段 1 进行编译，否则对程序段 2 进行编译。

条件编译指令都以#开头，其意义与选择结构语句 if-else 完全不同。if-else 语句分支中的程序段都会被生成到目标代码中，有程序在执行过程中根据条件来决定执行哪一个分支的代码，但条件编译#if-#else-#endif 则是在编译预处理时起作用，满足条件的程序段会被生成到目标代码中，而另一部分则会被舍弃。

【解题技巧】

例 8.4.1 下列程序段的输出结果是

```
#define T 10
```

```
#define MD 4*T
```

```
printf("%d",40/MD); ( )。
```

A) 0

B) 1

C) 10

D) 100

正解：D

分析：宏定义就是进行简单的替换，不改变顺序， $40/MD=40/4*10=10*10=100$ ；

【精选习题】

答案 P122

基础篇

1. 以下关于 C 语言中函数的描述错误的是 ()
 - A) 不同的函数中可以使用相同名字的变量, 互不干扰。
 - B) 形式参数都是局部变量
 - C) 函数不但可以嵌套调用还可以递归调用
 - D) C 语言中的函数参数传递都是单向值传递
2. 以下叙述中正确的是 ()
 - A) 函数名允许用数字开头
 - B) 函数调用时, 不必区分函数名称的大小写
 - C) 调用函数时, 函数名必须与被调用的函数名完全一致
 - D) 在函数体中只能出现一次 return 语句
3. 以下叙述中正确的是 ()
 - A) 任何情况下都不能调用函数名作为实参
 - B) 函数既可以直接调用自己, 也可以间接调用自己
 - C) 函数体中只能出现一次 return 语句
 - D) 简单递归不需要明确的结束递归的条件
4. 以下叙述中错误的是 ()
 - A) 用户定义的函数中可以没有 return 语句
 - B) 用户定义的函数中可以有多条 return 语句, 以便调用一次返回多个返回值
 - C) 用户定义的函数中若没有 return 语句, 则应当定义函数为 void 类型
 - D) 函数的 return 语句中可以没有表达式
5. 若有以下函数首部

```
int fun(double x[10],int *n)
```

则下面针对此函数的函数声明语句中正确的是 ()

- A) int fun(double ,int);
 - B) int fun(double *,int *);
 - C) int fun(double *,int n);
 - D) int fun(double x,int *n);
6. 有以下程序

```
#include<stdio.h>
fun(int a,int b)
{
    int t;
    t=a;
    a=b;
    b=t;
}
main()
{
    int c[10]={1,2,3,4,5,6,7,8,9,0},i;
    for(i=0;i<10;i++)
        fun(c[i],c[i+1]);
    for(i=0;i<10;i++)
        printf("%d",c[i]);
    printf("\n");
}
```

```
}
```

程序运行的结果是 ()

- A) 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
- B) 2, 1, 4, 3, 6, 5, 8, 7, 0, 9
- C) 0, 9, 8, 7, 6, 5, 4, 3, 2, 1
- D) 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

7.有以下程序

```
#include<stdio.h>
double f(double x);
int main()
{
    double a=0;
    int i;
    for(i=0;i<30;i+=10)
        a+=f((double)i);
    printf("%.3f\n",a);
}
double f(double x)
{
    return x*x+1; }
```

程序运行后地输出结果为 ()

- A) 500
- B) 401
- C) 503
- D) 1404

8.有以下程序

```
#include<stdio.h>
void fun(char *c,int d)
{
    *c=*c+1;
    d=d+1;
    printf("%c,%c,",*c,d);
}
int main()
{
    char b='a',a='A';
    fun(&b,a);
    printf("%c,%c\n",b,a);
}
```

程序运行后的结果为 ()

- A) b,B,b,A
- B) b,B,B,A
- C) a,B,B,a
- D) a,B,a,B

9.有以下程序

```
#include<stdio.h>
void f(int *p,int *q);
void main()
{
    int m=1,n=2,*r=&m;
    f(r,&n);
```

```
printf("%d,%d",m,n);
}
void f(int *p,int *q)
{
    p=p+1;
    *q=*q+1;
}
```

程序运行的输出结果是 ()

- A) 2, 3 B) 1, 3 C) 1, 4 D) 1, 2

10. 有以下程序

```
#include<stdio.h>
int fun(int x)
{
    int p;
    if(x==0||x==1)
        return(3);
    p=x-fun(x-2);
    return p;
}
int main()
{
    printf("%d\n",fun(7));
}
```

程序运行的结果是 ()

- A) 2 B) 3 C) 7 D) 0

提高篇

1. 有以下程序

```
#include <stdio.h>
#include <string.h>
void main()
{
    char *name[]={"Java","Basical","windows","TurboC++"};
    int a,b,n=4;
    char *temp;
    for(a=0;a<n-1;a++)
        for(b=a+1;b<n;b++)
        {
            if(strcmp(name[a],name[b])<0)
            {
                temp=name[a];
                name[a]=name[b];
                name[b]=temp;
            }
        }
}
```

```
        }  
    }  
    for(a=1;a<n;a++)  
        printf("%s\n",name[a]);  
}
```

程序最后输出的结果为_____

2. 有以下程序

```
#include<stdio.h>  
int runc(int a,int b)  
{  
    return(a+b);  
}  
main()  
{  
    int x=2,y=5,z=8,r;  
    r=runc(runc(x,y),z);  
    printf("%d\n",r);  
}  程序运行后的结果为_____.
```

3. 编写函数 `int search(int a[],int n)` 在 n 个整数中，找出第一个能被 7 整除的数。若找到，返回该数在这组数中的位置（注：找到的元素的下标值加 1）；若未找到，返回 0。在 `main` 函数中输入并调用 `search` 函数，并输出查找结果。
4. 使用循环结构编写函数 `void draw_triangle(int line)`；该函数可以在屏幕上打印如图的三角形，其中三角形函数由 `line` 控制，如在 `main` 函数中调用 `draw_triangle(5)` 时，可以输出如下图形

```
 *  
**  
***  
****  
*****
```

5. 编写一个自定义函数 `ftoc`，完成将华氏温度转换为摄氏温度，并编写 `main` 函数调用 `ftoc` 函数求输入华氏温度对应的摄氏温度。