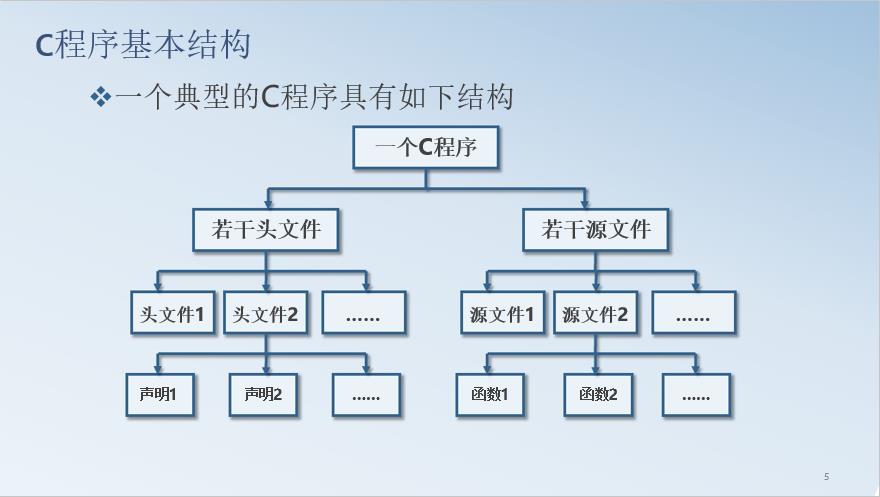
# C语言的基本结构

**一个简单的C程序具有如下结构**

#include<stdio.h>

int main(void)

{

int a=2,b=3;

printf(“%d”,a+b);

return 0;

}

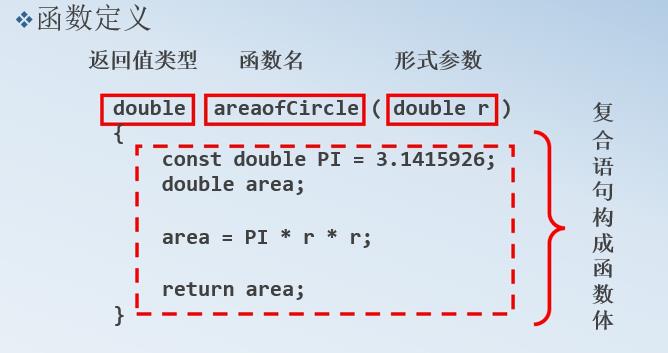
**函数是能够完成特定功能的代码集合。**

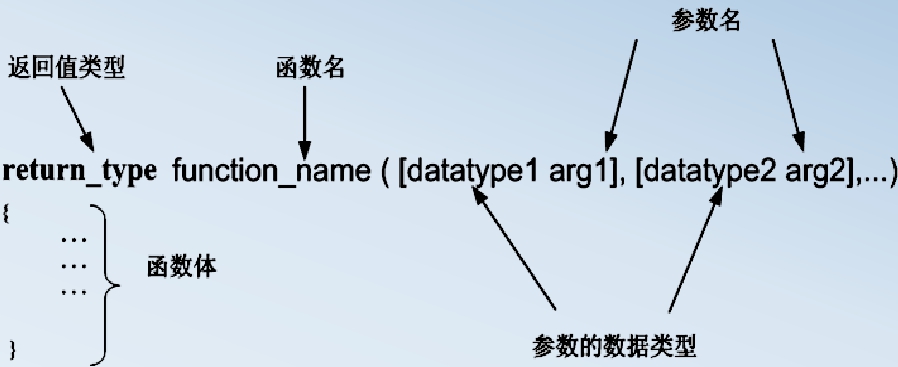
**函数的作用**

(1)减少程序的代码量 (2)代码复用 (3)程序具有良好的结构

# 函数的定义和声明

## 函数的定义





**注意：**

**1.在函数定义中若有参数，写明形参列表参数类型及参数名，称作有参函数.若无参数称作无参函数，需写void.**

**2.若函数有返回值，写明类型，否则写void.**

**3.函数定义不可嵌套**

**4.函数的命名规范，从第二个单词起首字母大写.**

## 函数的声明

**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.**

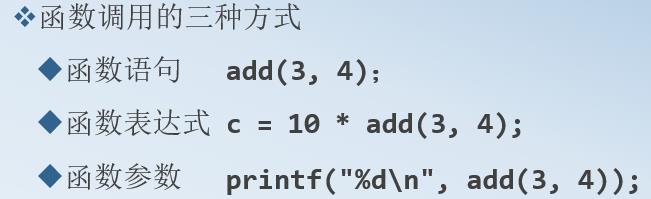
**5.函数返回值不能是局部变量的地址！**

# 函数的调用

定义一个函数后，就可以在程序中调用这个函数。

**（1）自定义函数必须先定义后调用**

**（2）库函数调用必须包含相应的头文件（包含定义）**

****

## 函数调用的过程

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;

}

返回两个值得和

#include<stdio.h>

#include<stdlib.h>

int swap(int x,int y)

{

int z;

z=x+y;

return z;

}

int main(void)

{

int a=3,b=5;

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

system("pause");

return 0;

}

## 函数原型

**函数原型声明**

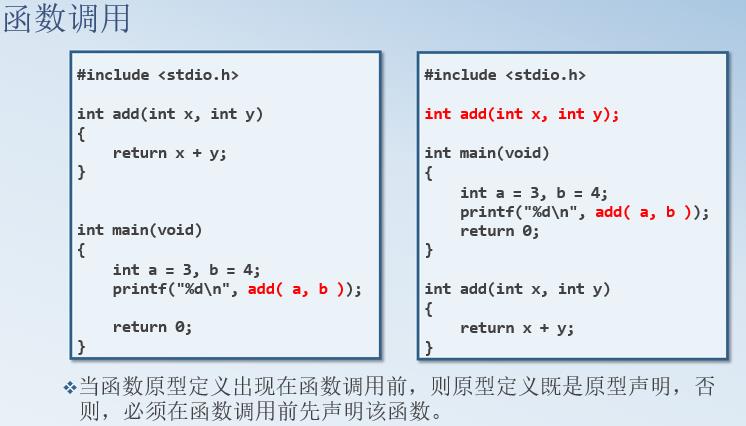
**返回值类型 函数名(参数类型 参数1,…,参数类型 参数n);**

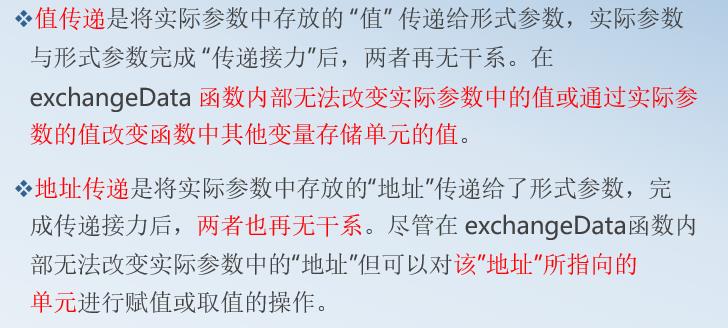
**说明函数的类型和参数的情况，以保证程序编译时能判断对该函数的调用是否正确。**

**函数原型声明标示了函数的返回值类型、函数名、参数个数和类型，是函数的“名片”。**

**函数原型声明和函数原型定义在返回类型、函数名、参数个数和类型必须完全一致。**

**函数原型声明不必包含参数的名字，而只要包含参数的类型。**

****

****

#include<stdio.h>

#include<stdlib.h>

void printArr(int \*p, const int len)//将\*p换成p[]

{

int i;

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

{

printf("%d",\*(p+i));

}

printf("\n");

}

int main(void)

{

int arr[10]={1,2,3,4,5,6,7,8,9,0};

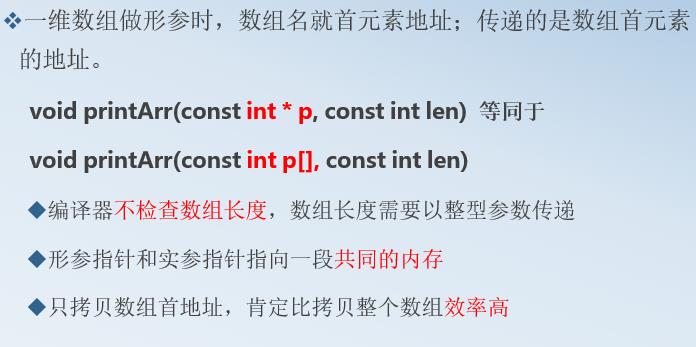
printArr(arr,10);

system("pause");

return 0;

}

## 一维数组做形参

****

**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;

}

# 函数的嵌套调用和递归调用

**函数直接或间接调用自己为递归调用**

## 编程求n的阶乘。

#include<stdio.h>

#include<stdlib.h>

int f(int n)

{

if (n==1)

return 1;

else

{

int t=n\*f(n-1);

return t;

}

}

int main(void)

{

int t=f(3);

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

system("pause");

return 0;

}

## 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

****

**递归原理：**

问题的求解可通过降低问题规模实现，而小规模的问题求解方式与原问题的一样，小规模问题的解决导致问题的最终解决

**递归调用应该能够在有限次数内终止递归：**

（1）递归调用如果不加以限制，将无数次的循环调用

（2）必须在函数内部加控制语句,只有当满足一定条件时，递归终止

## 函数一：add函数求解一个数组的各个元素的和（用两种方法）

### 递归法

#include<stdio.h>

#include<stdlib.h>

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;

}

}

int main(void)

{

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

int sum;

sum=add(a,5);

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

system("pause");

return 0;

}

### 非递归法

#include<stdio.h>

#include<stdlib.h>

int add(int p[],int len)

{

int t=0,i;

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

{

t=t+\*(p+i);

}

return t;

}

int main(void)

{

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

int sum;

sum=add(a,5);

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

system("pause");

return 0;

}

## 练习：读程序写结果

#include<stdio.h>

#include<stdlib.h>

void ff(int n)

{

if (n==0)

return ;

else

{

ff(n/10);

printf("%d",n%10);

}

}

int main(void)

{

ff(123);

printf("\n");

system("pause");

return 0;

}

结果：

123

请按任意键继续. . .

#include<stdio.h>

#include<stdlib.h>

void ff(int n)

{

if (n==0)

return ;

else

{

printf("%d",n%10);

ff(n/10);

}

}

int main(void)

{

ff(123);

printf("\n");

system("pause");

return 0;

}

结果：

321

请按任意键继续. . .

## 函数二：斐波那契数列

#include<stdio.h>

#include<stdlib.h>

int f(int x)

{

int ff=0;

if (x==1)

return 1;

else

{

if (x==2)

return 1;

else

{

ff=f(x-1)+f(x-2);

return ff;

}

}

}

int main(void)

{

printf("%d",f(6));

system("pause");

return 0;

}

## 函数三：折半查找

#include<stdio.h>

#include<stdlib.h>

int binofSearch(int \*p,int low,int high,int x)

{

int mid=(low+high)/2;

if (low>high)

return -1;

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

return mid;

if (x>\*(p+mid))

return binofSearch(p,mid+1,high,x);

if (x<\*(p+mid))

return binofSearch(p,low,mid-1,x);

}

int main(void)

{

int a[10]={1,4,8,23,45,678,4567,34567,345678,3456789};

int y;int x;

scanf("%d",&y);

x=binofSearch(a,0,9,y);

if (x==-1)

printf("没有找到");

else

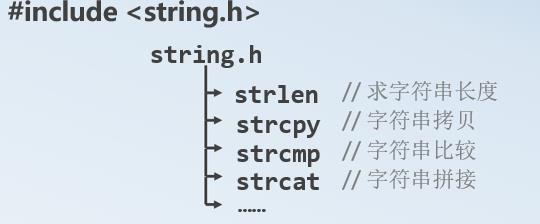
printf("找到了，是第%d个数",x+1);

system("pause");

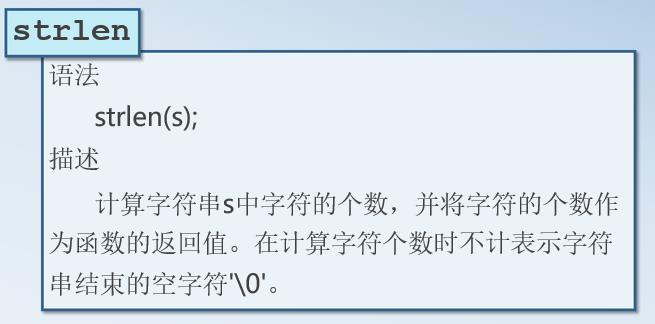
return 0;

}

# 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;

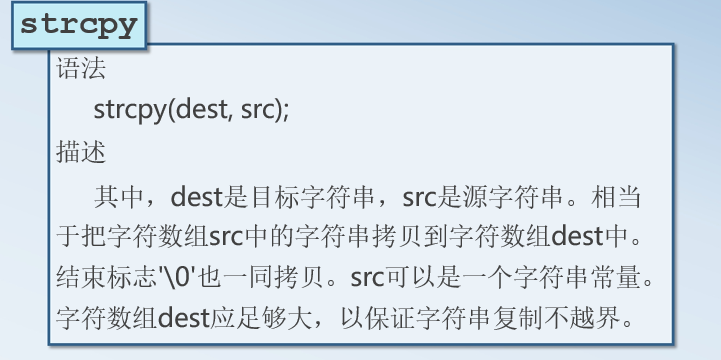
while(\*(p+s)!='\0')

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);

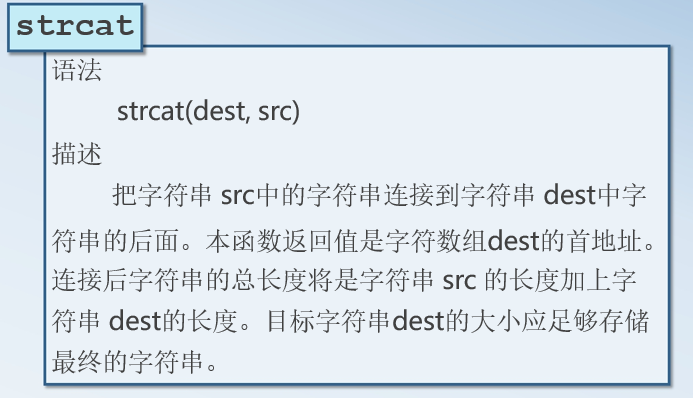
printf("%s",x);

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);

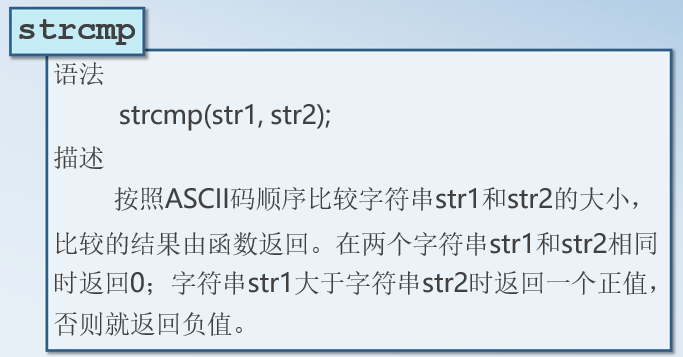
printf("%s",x);

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;

}

# 生存周期与声明作用域

## 生存周期

**生存周期的概念：**

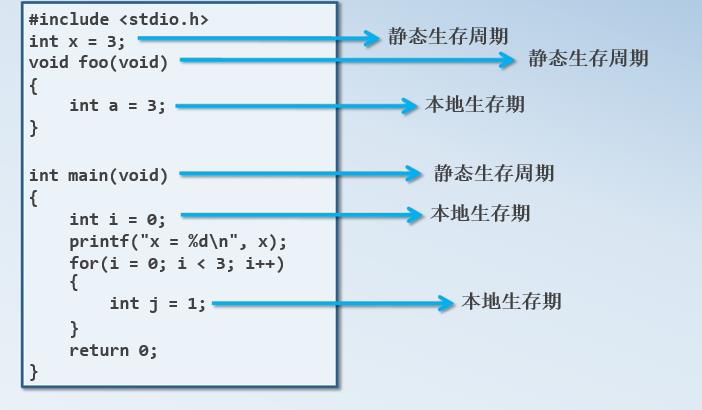
**变量保持所分配存储空间的时间，称为变量的存储期间或生存期。**

**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）

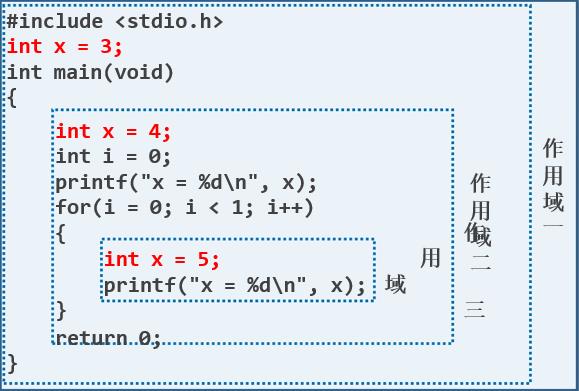
函数定义中的形式参数（从声明点到函数体结束之间的文本区域）

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。**

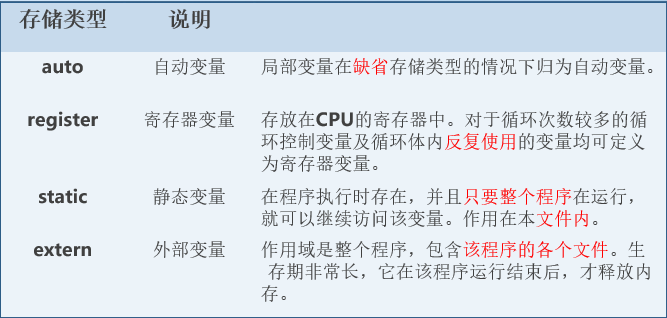
**总结：**

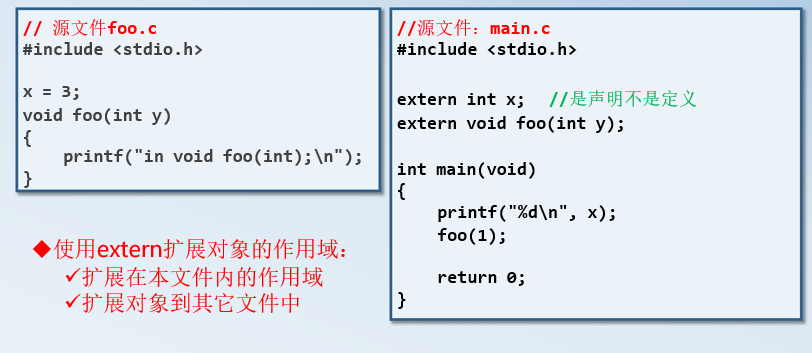
**1.局部变量的作用域为定义它的块儿**

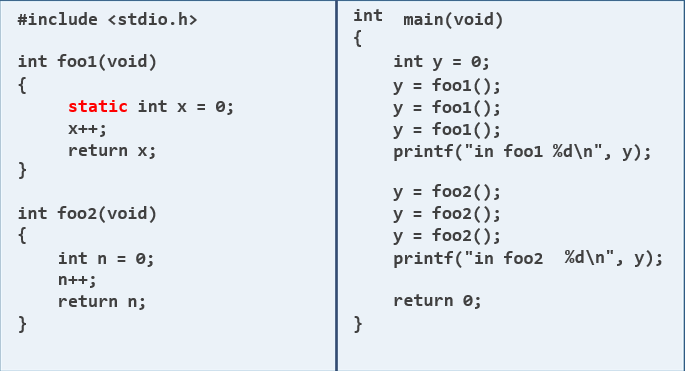
**2.全局变量的作用域为从定义它开始到程序结束**

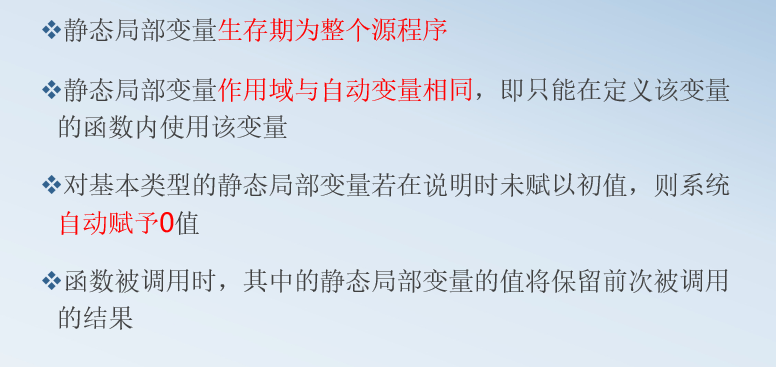
# 存储类型说明符

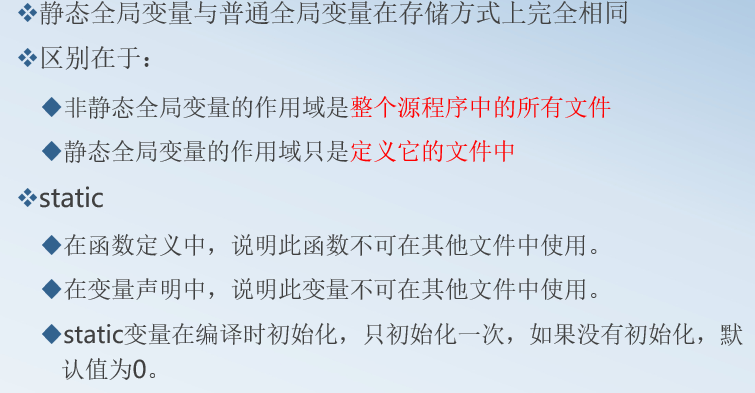
**存储类说明符用于确定声明对象的生存期**（typedef 除外），**存储类说明符出现在声明说明符中，一个声明说明符中最多允许一个存储类说明符。存储类说明符包括：**











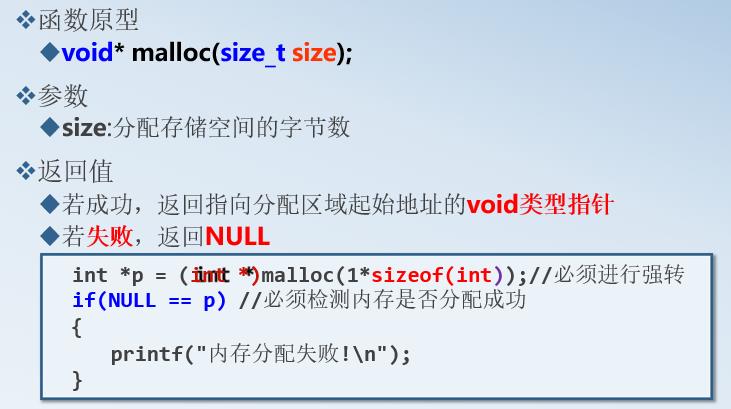
# 动态内存分配标准库函数

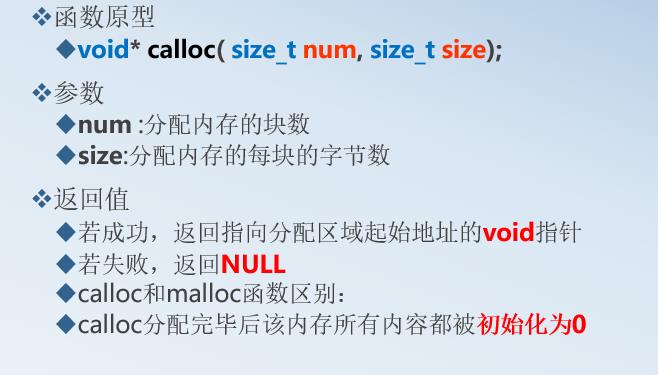
## 程序在内存中的分布区域

**库函数中**有一类函数**可以让我们自己分配内存**

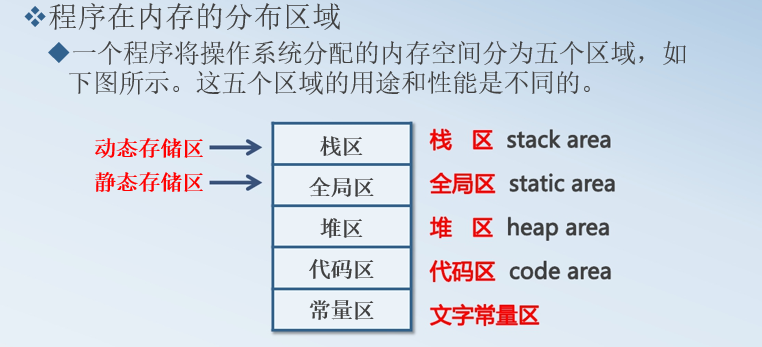
**Malloc** **calloc** **free**

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。

　int (\*f) (int x); /\* 声明一个函数指针 \*/

　f=func; /\* 将func函数的首地址赋给指针f \*/

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)  
    其实这里不能称为函数名，应该叫做指针的变量名。这个特殊的指针指向一个返回整型值的函数。指针的声明笔削和它指向函数的声明保持一致。  
        指针名和指针运算符外面的括号改变了默认的运算符优先级。如果没有圆括号，就变成了一个返回整型指针的函数的原型声明。  
    例如：  
        void (\*fptr)();  
    把函数的地址赋值给函数指针，可以采用下面两种形式：  
        fptr=&Function;  
        fptr=Function;  
    取地址运算符&不是必需的，因为单单一个函数标识符就标号表示了它的地址，如果是函数调用，还必须包含一个圆括号括起来的参数表。  
    可以采用如下两种方式来通过指针调用函数：  
        x=(\*fptr)();  
        x=fptr();  
    第二种格式看上去和函数调用无异。但是有些程序员倾向于使用第一种格式，因为它明确指出是通过指针而非函数名来调用函数的。下面举一个例子：

# 大练习：

**实现输入一个数，是这个数作为字符串的个数，并且让用户输入不多于30个字符，实现这几个字符串的排序。**

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

int strlen(char \*x)

{

int s=0;

while (x[s]!='\0')

s++;

return s;

}

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 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;

}

void paixu(char \*\*x,int len)

{

int i,j;

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

{

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

{

if (strcmp(x[i],x[j])==1)

break;

else

{

char \*t=x[j];

x[j]=x[j+1];

x[j+1]=t;

}

}

}

printf("排序之后的字符串为:");

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

printf("%s",\*(x+i));

}

int main(void)

{

int len;

int i=0;

char \*\*p;

char a[30];

int t\_len=0;

scanf("%d",&len);

p = (char \*\*)malloc(len\*sizeof(char\*));

while (i<len)

{

printf("请输入您要的字符串:\n");

scanf("%s",a);

t\_len=strlen(a);

a[t\_len]='\0';

p[i]=(char \*)malloc((t\_len+1)\*sizeof(char));

strcpy(p[i],a);

i++;

}

printf("您输入的字符串为:");

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

printf("%s",\*(p+i));

paixu(p,len);

system("pause");

return 0;

}