

面向对象程序设计

授课：姜娓娓

邮箱：jwwzjut@zjut.edu.cn

QQ：281943101

课程概述

◆课程教材:

- 《C++程序设计》（第3版）. 谭浩强. 清华大学出版社, 2015,8. （清华大学出版社 清华大学出版社电子图书数据库“文泉学堂”免费开放 <HTTPS://LIB-NUANXIN.WQXUETANG.COM>）

◆学时与学分:

- 64学时=48学时讲授+16学时实验

◆考核方式:

- 笔试,120分钟

◆成绩构成:

- 平时50%(作业20%, 上课/随堂测验10%, 上机实验20%);
- 考试50%

课程要求

1. 课前做好预习
2. 保持课堂安静，头脑清醒，思维活跃
3. 认真、独立、按时完成并提交作业，三次未交作业，作不及格处理
4. 重视上机实践，有效利用宝贵的上机时间，三次上机缺席，作不及格处理

C语言复习

类的声明和对象的定义

类的成员函数和数据成员

构造函数、析构函数

对象数组、对象指针
共用数据的保护

堆与拷贝构造函数
浅拷贝与深拷贝

静态成员

友元的概念
友元的使用

单链表

单链表访问和基本操作

运算符重载

继承与派生

派生类的构造

多态

虚析构函数

抽象类

抽象类下的派生

纯虚函数

输入输出流

函数模板

类模板

异常的概念

异常机制与规则

序号	实验项目名称
1	类的声明、定义和使用
2	类的构造构造函数、类的使用
3	链表
4	类的操作符重载
5	类的继承及派生类的使用
6	虚函数的使用
7	类的多态性
8	模板类和类模板

C语言知识点复习

- 控制结构
- 函数
- 指针

控制结构-选择结构

```
if (condition) {  
    statements  
}  
else {  
    statements  
}
```

if 结构的嵌套（变形）

if - else if 结构

判断学生成绩等级

```
if (score<60) printf("你的成绩为不及格\n");
```

```
else if (score<70) printf("你的成绩为及格\n");
```

```
else if (score<80) printf("你的成绩为中等\n");
```

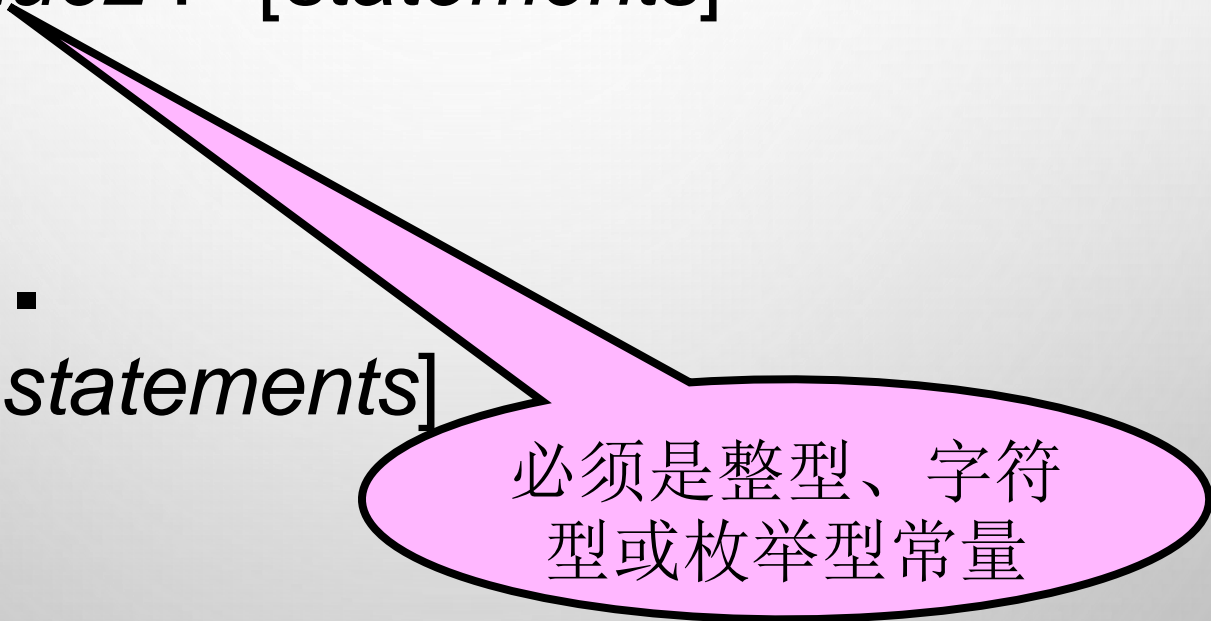
```
else if (score<90) printf("你的成绩为良好\n");
```

```
else printf("你的成绩为优秀\n");
```

五分支

控制结构-选择结构

```
switch(expression) {  
    Case value1 : [statements]  
    Case value2 : [statements]  
        ▪  
        ▪  
        ▪  
    default : [statements]  
}
```



必须是整型、字符
型或枚举型常量

判断学生成绩等级

```
grade = score/10;
switch (grade)
{ case 10:
  case 9: printf("你的成绩为优秀\n"); break;
  case 8: printf("你的成绩为良好\n "); break;
  case 7: printf("你的成绩为中等\n "); break;
  case 6: printf("你的成绩为及格\n "); break;
  case 5:
  case 4:
  case 3:
  case 2:
  case 1:
  case 0: printf("你的成绩为不及格\n"); break;
  default: printf("The score is out of range!\n");
}
```

控制结构-循环结构

在C语言中，可用以下语句实现循环：

- (1) 用for语句。
- (2) 用do-while语句。
- (3) 用while语句。

例：输入 10个数，求它们的和并输出。

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

```
void main()
```

```
{ float a,s; int i;
```

```
    for(s=0,i=1;i<=10;i++)
```

```
    {
```

```
        scanf("%f",&a);
```

```
        s+=a;
```

```
    }
```

```
    printf("%f\n",s);
```

```
}
```

while 结构

一般格式:

```
while(循环继续条件)
{ 循环体语句组; }
```

```
void main()
{ int i=1,sum=0;
  while( i<=100 )
  { sum += i; /*实现累加*/
    i++; /*循环控制变量i增1*/
  }
  printf("sum=%d\n",sum);
}
```

1. 测试 *i* 是否 > 100 。若是的话, 则 退出循环。
2. 执行语句 *sum=sum+i*。
3. *i* 增加1。
5. 重复步骤 1 到步骤 3。

do while结构

一般格式:

```
do  
{ 循环体语句组; }  
while(循环继续条件);
```

```
void main()  
{ int i=1,sum=0;  
  do  
    { sum += i; /*实现累加*/  
      i++; /*循环控制变量i增1*/  
    }while( i<=100 );  
  printf("sum=%d\n",sum);  
}
```

1. 执行语句 *sum=sum+i*。
2. *i* 增加1。
3. 测试 *i* 是否 *>100*。若是的话，则 退出循环。
4. 重复步骤 1 到步骤 3。

```
while( i<=100 )  
{ sum += i; /*实现累加*/  
  i++; /*循环控制变量i增1*/  
}
```



有什么区别?

```
do  
{ sum += i; /*实现累加*/  
  i++; /*循环控制变量i增1*/  
}while( i<=100 );
```

当一开始循环条件就不满足:

- while 结构的循环体一次也不执行
- do while 结构的循环体则执行一次

循环结构举例

输入x,n, 求下列的级数和:

$$1+x+x^2/2!+x^3/3!+\dots+x^n/n!$$

(分析: 多项式求和一般利用循环来实现)

完整的程序：

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

```
void main( )
```

```
{ float x; double y,item; int n,i;
```

```
    printf(" please input x:"); scanf("%f",&x);
```

```
    printf(" please input n:"); scanf("%d",&n);
```

```
    y=1; item=x;
```

```
    for(i=1;i<=n;i++) {
```

```
        y=y+item;
```

```
        item=item*x/ (i+1) ;
```

```
    }
```

```
    printf("结果为:y=%f\n", y);
```

```
}
```

几种不同的写法:

```
y=0; item=1;  
for(i=1;i<=n+1;i++) {  
    y=y+item;  
    item=item*x/i;  
}
```

习惯上把循环次数能确定的循环用for结构实现

```
y=1; item=1;  
for(i=1;i<=n;i++) {  
    item=item*x/i;  
    y=y+item;  
}
```

```
y=1; item=1; i=1;  
while(i<=n) {  
    item=item*x/i;  
    y=y+item;  
    i++;  
}
```

GOTO 语句，BREAK语句和CONTINUE 语句

强制跳转语句

一般认为，goto ,break和continue不符合结构化程序设计的思想，应限制使用；有时，为了合理表达算法的思路，以及程序的效率，适当使用。

- **break:** 强行结束循环，转向执行循环之后的语句。

例

.....

while(1)

if (scanf("%c",&ch)==EOF)

break;

else

k++;

printf(".....");

Ctrl+Z

stdio.h中定义的符号常量，相当于-1



.....

while(scanf("%c",&ch) != EOF)


k++;

•**continue**: 对于for循环, 跳过循环体其余语句, 转向循环变量增量表达式的计算; 对于while和do-while循环, 跳过循环体其余语句, 但转向循环继续条件的判定。

例8 输入10个数, 将这10个数中非0数相乘, 计算其乘积, 并统计非0数据个数。

```
.....  
for(i=1; i<=10; i++){  
    scanf("%f",&x);  
    if(x==0) continue;  
    y*=x;  
    n++;  
}
```

```
.....  
for(i=1; i<=10; i++){  
    scanf("%f",&x);  
    if(x!=0) {  
        s+=x;  
        n++; }  
}
```



说明：

（1）`break`能用于循环语句和`switch`语句中，`continue`只能用于循环语句中。

（2）循环嵌套时，`break`和`continue`只影响包含它们的最内层循环，与外层循环无关。

- **goto 语句:** 把程序的执行强制性地改变到标号指明的语句上。

```
While(.....)
```

```
{ .....
  for(i=1; i<n; i++){
```

```
    .....
    if( score<0)
```

```
    .....
    if( score<0)
```

```
    .....
    goto error1
```

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

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

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

```
error1: printf("The data is damaged\n");
```

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

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



多重循环

循环体内，可以又包括一个循环结构。

- ❖ 三种循环可互相嵌套,层数不限
- ❖ 外层循环可包含两个以上内循环,但不能相互交叉
- ❖ 嵌套循环的跳转

禁止:

- 从外层跳入内层
- 跳入同层的另一循环
- 向上跳转

```
(1) do
    { .....
      do
        { .....
          }while( );
      .....
    }while( );
```

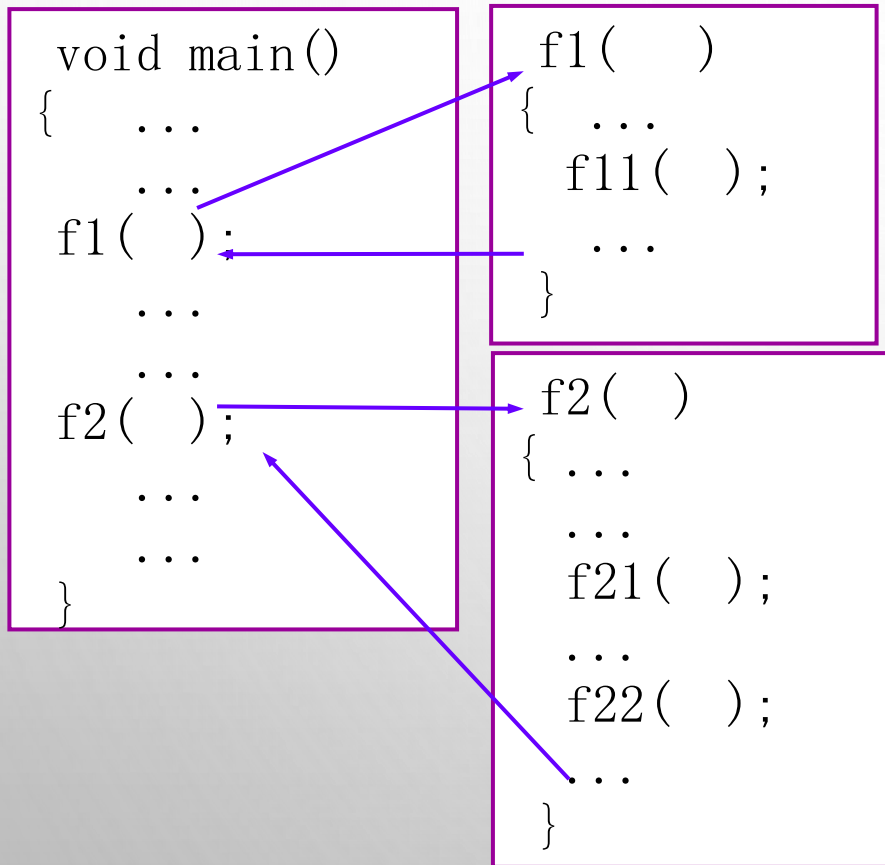
```
(2) while()
    { .....
      do
        { .....
          }while( );
      .....
    }
```

```
(3) for( ; ; )
    { .....
      do
        { .....
          }while();
      .....
      while()
        { .....
        }
      .....
    }
```

外层循环

内层循环

函数



(1) 每个C程序至少有一个函数：**main()**函数。

(2) 可以在程序里调用标准库函数。

系统已为一些常用的任务（比如基本输入、输出、求sin）编写了函数，并放在标准库中。

(3) 用户自定义函数

例 已知五边形的各边及对角连线长度，求面积。

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

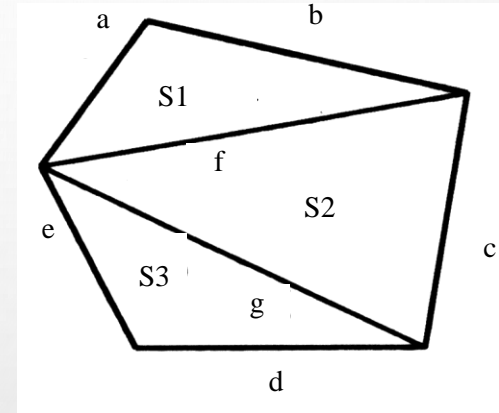
```
#include<math.h>
```

```
float area(float x,float y,float z) // 自定义求三角形面积函数
```

```
{ float s, a;  
  s=(x+y+z)/2;  
  a=sqrt(s*(s-x)*(s-y)*(s-z));  
  return a; }
```

```
void main()
```

```
{ float a,b,c,d,e,f,g;  
  float s1,s2,s3,s;  
  scanf("%f%f%f%f%f",&a,&b,&c,&d,&e); //输入五条边长  
  scanf("%f%f",&f,&g); //输入两条对角连线  
  s1=area(a,b,f); //调用计算三角形面积函数  
  s2=area(c,g,f); //调用计算三角形面积函数  
  s3=area(d,e,g); //调用计算三角形面积函数  
  s=s1+s2+s3;  
  printf("五边形面积%f",s);} printf("s1%f", area(a,b,f));
```

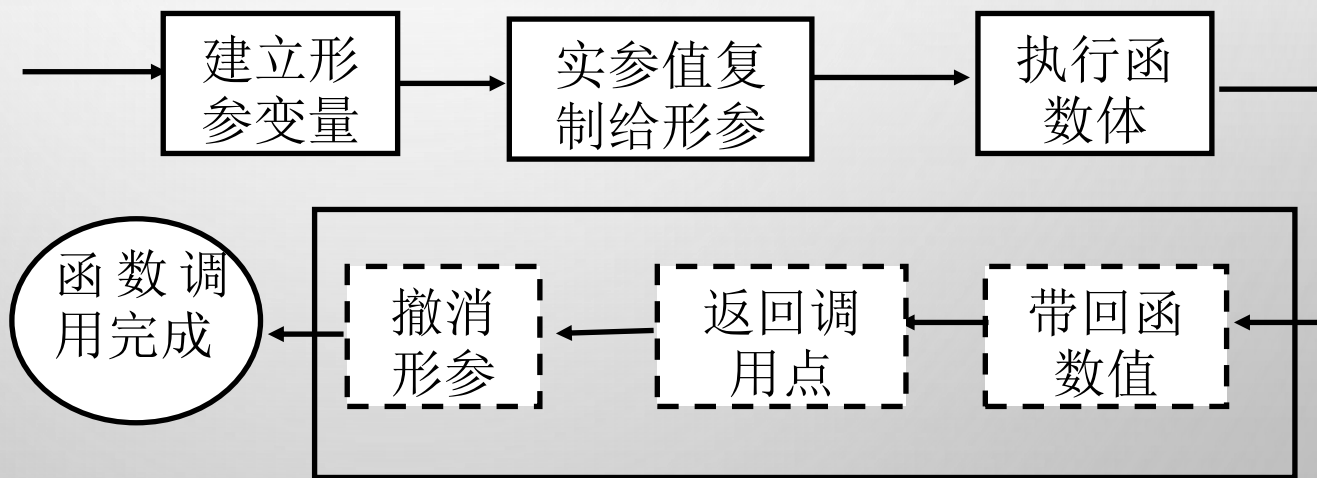


函数-参数传递

```
float area(float x,float y,float z) // 自定义求三角形面积函数
```

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

```
void main()  
{  
    .....  
    s1=area(a,b,f);  
    s2=area(c,g,f);  
    .....  
}
```



函数调用的整个执行过程

① 传值调用

结论：在传值调用中，实参的值不随调用函数中形参的变化而变化

```
void main()
```

```
{ void s(int n);
```

```
  int n=100; s(n);
```

```
  printf("(in main) n=%d\n",n); //输出调用后实参的值
```

```
}
```

```
void s(int n)
```

```
{ int i;
```

```
  printf("(in s) n=%d\n",n);
```

```
  for(i=n-1; i>=1; i--) n=n+i;
```

```
  printf("(in s) n=%d\n",n);
```

```
}
```

//说明函数

/*调用函数*/

100

100

100

//输出改变前形参的值

//改变形参的值

//输出改变后形参的值

5050

② 引用调用(传址)

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

```
void main()
```

```
{ float a=10,b=5;
```

```
void swap(float &,float &);
```

```
swap(a,b);
```

```
printf("a=%f b=%f\n",a,b);
```

```
}
```

```
void swap(float &x,float &y)
```

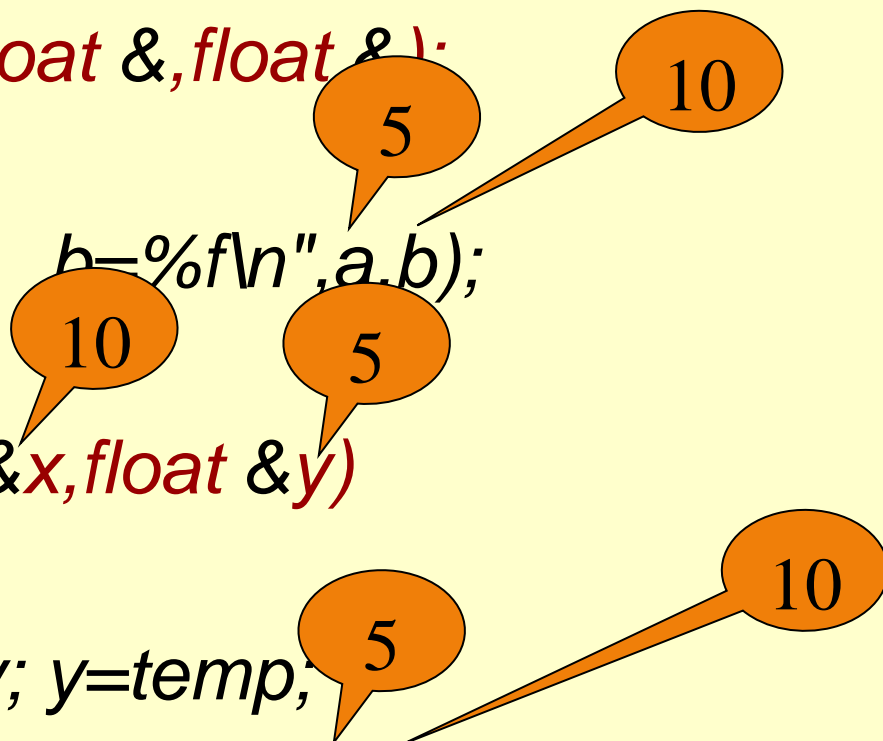
```
{ float temp;
```

```
temp=x; x=y; y=temp;
```

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

```
}
```

结论：在引用调用中，函数中对形参的操作，实际上就是对实参的操作。



指针

指针的声明

❖ **[存储类型] 数据类型 *指针名 ;**

例 `int *p1,*p2;
 float *q ;
 static char *name;`

• 运算符 **&** 与 *

含义: 取变量的地址
单目运算符
优先级: 2
结合性: 自右向左

含义: 取指针所指向变量的内容
单目运算符
优先级: 2
结合性: 自右向左

指针的赋值

```
例    int i;  
      int *p=&i;
```

变量必须已说明过
类型应一致

```
例    int i;  
      int *p;  
      p=&i;
```

指针变量必须先赋值, 再使用

```
例    main( )  
    {   short i=10;  
        short *p;  
        *p=i;  
        printf( "%d" ,*p);  
    }
```

```
例    main( )  
    {   short i=10,k;  
        short *p;  
        p=&k;  
        *p=i;  
        printf( "%d" ,*p);  
    }
```

悬挂指针


```
int x,*p=&x;
```



```
int x,*p; p=&x;
```



```
int x,*p;
```

```
*p=x;
```

```
*p=&x;
```



p还是悬挂指针

```
int x,*p;  
p=2000;
```

✗ 指针变量中只能
存储地址

★ 零指针

● 定义: 指针变量值为零

● 表示: `int * p=0;`

p指向地址为0的单元，
系统保证该单元不作它用
表示指针变量值没有意义

指针与数组

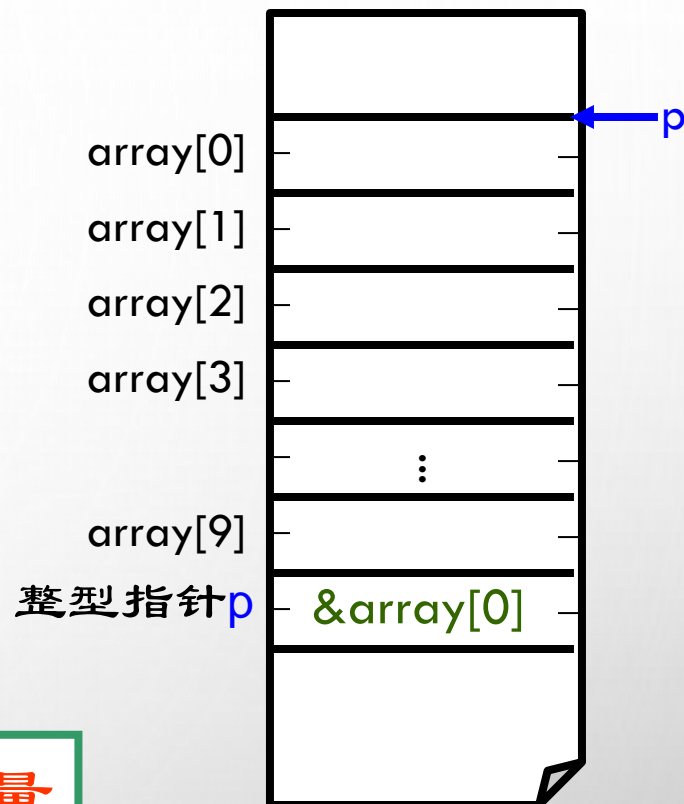
例 `int array[10];`

`int *p;`

`p=&array[0];` `//⇔ p=array;`

或 `int *p=&array[0];`

或 `int *p=array;`



数组名是表示数组首地址的地址常量

指针算数

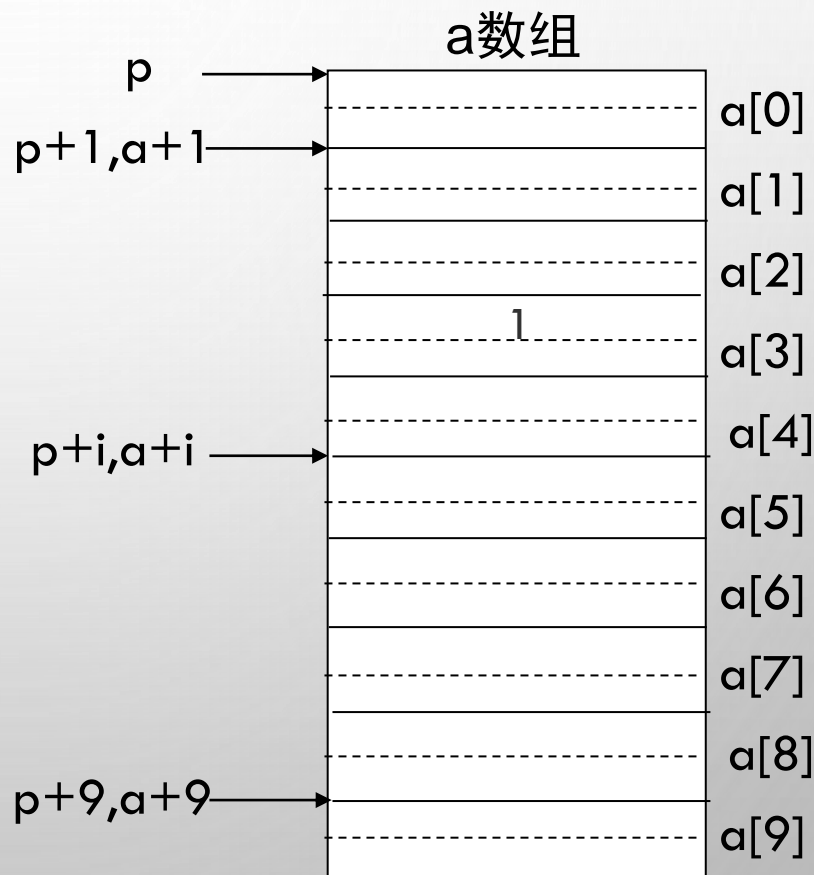
- $p \pm i \Leftrightarrow p \pm i \times d$ (i 为整型数, d 为 p 指向的变量所占字节数)
- $p++$, $p--$, $p+i$, $p-i$, $p+=i$, $p-=i$ 等
- 若 $p1$ 与 $p2$ 指向同一数组, $p1-p2$ =两指针间元素个数 $\Leftrightarrow (p1-p2)/d$
- $p1+p2$ 无意义

例 p 指向float数, 则 $p+1 \Leftrightarrow p+1 \times 4$

例 p 指向short型数组, 且 $p=\&a[0]$;
则 $p+1$ 指向 $a[1]$

例 `short a[10];`
`short *p=&a[2];`
`p++;`
`*p=1;`

例 `short a[10];`
`short *p1=&a[2];`
`short *p2=&a[5];`
则: $p2-p1=3$;



例 int a[]={1,2,3,4,5,6,7,8,9,10},*p=a,i;

数组元素地址的正确表示:

(A) &(a+1)

(B) a++

(C) &p

(D) &p[i]

✓

数组名是地址常量

p++,p-- (✓)

a++,a-- (✗)

a+1, *(a+2) (✓)

数组表示法和指针表示法

[] 变址运算符

$a[0] \Leftrightarrow *a$, 表示在地址 a 处的元素值

$a[i] \Leftrightarrow *(a+i)$, 表示在地址 $a+i$ 处的元素值

数组名和指针变量都是如此, 因此对于指针和数组名, 既可以使用指针表示法, 也可以使用数组表示法

$$a[i] \Leftrightarrow p[i] \Leftrightarrow *(p+i) \Leftrightarrow *(a+i)$$

```
int a[5];  
int *pt = a;  
*pt = 5;  
pt[0] = 6;  
pt[4] = 44;  
int coats[10];  
*(coats+4) = 12;
```

指针与字符串

字符指针**初始化**:把字符串**首地址**赋给 pstring

```
char *pstring;
```

```
pstring= "I love China!";
```

或者

```
char arr[20] = "I love China!";
```

```
pstring =arr;
```

```
char str[]={ "Hello!" };
```

(✓)

```
char str[]= "Hello!";
```

(✓)

```
char str[]={ 'H' , 'e' , 'l' , 'l' , 'o' , '!' }; (✓)
```

```
char *cp= "Hello";
```

(✓)

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

(✓)

```
int *p={1,2,3,4,5};
```

(✗)

```
char str[10],*cp;
```

```
int a[10],*p;
```

```
str= "Hello"; (✗)
```

```
cp= "Hello!"; (✓)
```

```
a={1,2,3,4,5}; (✗)
```

```
p={1,2,3,4,5}; (✗)
```

函数指针

★ 函数指针：函数在编译时被分配的入口地址，用函数名表示

★ 定义形式： **数据类型 (*指针变量名)(行参列表);**

如 int (*p)(int, int);

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

```
float mult(float x, float y) // 函数mult, 求x,y的乘积。
```

```
{ return x*y; }
```

```
float div(float x, float y) // 函数div, 求x,y的商。
```

```
{ if (y!=0) return x/y; // 返回x,y的商
```

```
else
```

```
{ printf("error\n");
```

```
return 0; } // 返回0, 表示除数为0
```

```
}
```

```
void main()
```

```
{ float (*p)(float, float); // 定义p为指向函数的指针变量
```

```
float y;
```

```
p=mult;
```

```
y=p(5,3); // 调用函数mult(5,3), 也可写成(*p)(5,3)
```

```
printf("%.1f\n", y);
```

```
p=div;
```

```
y=p(5,3); // 调用函数div(5,3), 也可写成(*p)(5,3)
```

```
printf("%.1f\n", y);
```

```
}
```

指针函数

- ★ 指针函数：返回指针值的函数
- ★ 定义形式：函数类型 *函数名([形参表])
如 `int *p(int, int);`
- ★ 数组名作函数参数，是地址传递；

<例>将数组a中的n个整数按相反顺序存放

```
void convert(int *p, int n)
```

```
{  int t,*q;
    for(q=p+n-1; p<q; p++, q--)
    {  t=*p;
        *p=*q;
        *q=t;
    }
}
```

```
void main()
```

```
{  int i,a[10]={3,7,9,11,0,6,7,5,4,2},*pa=a;
```

```
    convert(pa,10);
```

```
    printf("reverted:\n");
```

```
    for(i=0;i<10;i++)
```

```
        printf("%d,", *(pa+i));
```

```
    printf("\n");
```

```
}
```

```
for( ; pa<a+10; pa++)
    printf( "%d" , *pa);
```

```
for(i=0; i<10; i++,pa++)
    printf( "%d" , *pa);
```

```
for(i=0; i<10; i++)
    printf( "%d" , *pa++);
```