计算机导论与程序设计 [CS006001-60]

段江涛

机电工程学院

西安电子科技大学 XIDIAN UNIVERSITY

2019 年 11 月

lecture-15 主要内容

善于利用指针

指针就是指向变量存储单元 (首) 地址的变量

- 在程序中定义的各个变量, 在对程序进行编译时, 系统就会给每个变量分配一个存储单元。
- 基本存储单元是一个字节 (byte)。
- 根据变量类型, 系统分配相应长度的存储单元, 其长度是字节的整数倍。
- 不同的编译器, 给变量分配的存储空间可能会有所不同。sizeof(类型), 可获得该编译器给相应类型的变量分配的存储空间大小。
- 指针就是指向目标变量存储单元 (首) 地址的变量。

```
char c;//分配1个存储单元, 即1个字节
int  a;//分配2个存储单元, 即2个字节(大多数是4个字节)
float f;//分配4个存储单元, 即4个字节
double d; //分配8个存储单元, 即8个字节
printf("%d,%d,%d,%d\n",sizeof(char),sizeof(int),
    sizeof(float),sizeof(double));
// 1,4,4,8
```

变量名	地址	内容
c	2000	'A'
	...	
a	3000	20
	3001	
	...	
f	4000	10.2
	4001	
	4002	
	4003	
	...	
d	5000	20.8
	5001	
	5002	
	5003	
	...	

定义指针变量: int *p;

指针就是指向变量存储单元(首)地址的变量。

```
数据类型 *指针变量名; //数据类型表示指针指向目标变量类型,*表示其后的变量是指针变量

int *pa; // 定义指向整数类型变量的指针变量pa, *表示其后的变量是指针变量
char *pc; // 定义指向字符类型变量的指针变量pc
float *pf; // 定义指向单精度浮点类型变量的指针变量pf
double *pd; // 定义指向双精度浮点类型变量的指针变量pd
```

特别注意

以上定义了各个指针变量,但是还没有设置指针指向的目标变量。未设置目标变量前的指针变量,是一个“无用”的变量,此时定义的指针变量仅说明了将来的目标变量的数据类型。

设置指针变量指向的目标变量

地址运算符 &, 可获得变量在内存中的(首)地址。

```
// 定义指针变量时，初始化它指向的目标变量

// 定义指针变量pa，并初始化pa指向变量a
int a=20,*pa=&a;
// 定义指针变量pc，并初始化pc指向变量c
char c='A',*pc=&c;
// 定义指针变量pf，并初始化pf指向变量f
float f=10.2,*pf=&f;
// 定义指针变量pd，并初始化pd指向变量d
double d=20.8,*pd=&d;

int b=30,*p;

// 根据需要，可以随时设置指针指向的目标变量

// p前无'*'，此时p已经是指针变量了，仅是设置它的指向
p=&b;
// 改变pa的指向
pa=p; // 或
pa=&b;
```

变量名	指针	地址	内容
c	pc →	2000	'A'
		...	
a	pa →	3000	20
		3001	
		...	
f	pf →	4000	10.2
		4001	
		4002	
		4003	
		...	
d	pd →	5000	20.8
		5001	
		5002	
		5003	
		...	

变量的直接访问

- **直接访问:** 通过变量名访问 (读写) 存储单元的内容 (变量的值)
- 此前的程序, 均采用直接访问的形式, 读写变量的值。

```
char c; // 分配1个存储单元, 即1个字节
int  a; // 分配2个存储单元, 即2个字节(大多数是4个字节)
float f; // 分配4个存储单元, 即4个字节
double d; // 分配8个存储单元, 即8个字节
// 直接访问
// "写" 变量的值
c='A';
a=20;
f=10.2;
d=20.8;
// "读"变量的值
printf("%c,%d,%f,%lf\n",c,a,f,d);
```

变量名	地址	内容
c	2000	'A'
	...	
a	3000	20
	3001	
	...	
f	4000	10.2
	4001	
	4002	
	4003	
	...	
d	5000	20.8
	5001	
	5002	
	5003	
	...	

变量的间接访问

- **间接访问:** 通过变量的 (首) 地址访问 (读写) 存储单元的内容 (变量的值)
- * 指针变量, **间接访问运算符 ***, 表示读写指针的目标变量值

```
// 定义指针变量, 并初始化, 指向特定的变量地址
char c, *pc=&c; //pc是指向char类型变量的指针变量
int  a, *pa=&a; //pa是指向char类型变量的指针变量
float f, *pf=&f; //pf是指向char类型变量的指针变量
double d, *pd=&d; //pd是指向char类型变量的指针变量
// 间接访问
// "写" 变量的值
*pc='A'; // 不要忘记前面的*, 否则成为改变指针指向的语句。
*pa=20;
*pf=10.2;
*pd=20.8;
// "读"变量的值
printf("%c,%d,%f,%lf\n",*pc,*pa,*pf,*pd);
```

变量名	指针	地址	内容
c	pc →	2000	'A'
		...	
a	pa →	3000	20
		3001	
		...	
f	pf →	4000	10.2
		4001	
		4002	
		4003	
		...	
d	pd →	5000	20.8
		5001	
		5002	
		5003	
		...	

指针变量使用(间接访问)前,必须有指向。

```
// 注意: 指针变量使用前, 必须有指向
float *p;
*p=10.2; // 错!!! p没有设置指向的目标变量, 即p未赋值, 间接访问是错误的。

float f;
p=&f;
*p=10.2; // 合法

int a, *pa;
scanf("%d",pa); // 错!!! pa没有设置指向的目标变量
pa=&a; // 设置pa指向a
scanf("%d",pa); // pa有指向了, 正确
scanf("%d",&a); // 等效
printf("%d,%d\n",a,*pa); // 输出相同的值
```

例: 交换两个指针变量的值 (地址, 指向)

```
int a=10, *pa=&a, b=20, *pb=&b;
int *tmp;
if(a<b) // 不交换a,b的值, 交换pa,pb的指向
{
  tmp=pa;
  pa=pb;
  pb=tmp;
}
printf("%d,%d,%d,%d\n",a,b,*pa,*pb); // 10,20,20,10

// 或者, 不使用中间变量
if(a<b) // 不交换a,b的值, 交换pa,pb的指向
{
  pa=&b;
  pb=&a;
}
printf("%d,%d,%d,%d\n",a,b,*pa,*pb); // 10,20,20,10
```

指针变量作为函数参数 (地址传递, 形参和实参指向同一存储单元)

```
void swap(int *p1, int *p2);
int main()
{
    int a=10, *pa=&a, b=20, *pb=&b;
    if(a<b) swap(pa,pb); //地址传递
    //if(a<b) swap(&a,&b); //等效, 地址传递
    printf("%d,%d,%d,%d\n",a,b,*pa,*pb);
    // 20,10,20,10
    return 0;
}
// 交换p1,p2指向的内容, 注意不是交换p1,p2本身
void swap(int *p1, int *p2)
{
   int tmp;
   tmp=*p1;
   *p1=*p2;
   *p2=tmp;
}
```

main 函数: 调用 swap 前

实参指针	地址	内容
pa=&a →	3000	10
	3001	
pb=&b →	4000	20
	4001	

swap 函数: p1,p2 指向内容交换前

形参指针	地址	内容
p1 →	3000	10
	3001	
p2 →	4000	20
	4001	

形参实参地址相同。

指针变量作为函数参数 (地址传递, 形参和实参指向同一存储单元)

```
void swap(int *p1, int *p2);
int main()
{
  int a=10, *pa=&a, b=20, *pb=&b;
  if(a<b) swap(pa,pb); //地址传递
  //if(a<b) swap(&a,&b); //等效, 地址传递
  printf("%d,%d,%d,%d\n",a,b,*pa,*pb);
  // 20,10,20,10
  return 0;
}
// 交换p1,p2指向的内容, 注意不是交换p1,p2本身
void swap(int *p1, int *p2)
{
  int tmp;
  tmp=*p1;
  *p1=*p2;
  *p2=tmp;
}
```

main 函数: 调用 swap 后

实参指针	地址	内容
pa=&a →	3000	10 → 20
	3001	
pb=&b →	4000	20 → 10
	4001	

swap 函数: p1,p2 指向内容交换后

形参指针	地址	内容
p1 →	3000	10 → 20
	3001	
p2 →	4000	20 → 10
	4001	

形参实参地址相同。函数内对指针目标变量的改变, 就是改变实参的内容。

普通变量作为函数参数 (值传递, 形参和实参处于不同存储单元)

```
void swap(int p1, int p2);
int main()
{
  int a=10, *pa=&a, b=20, *pb=&b;
  if(a<b) swap(a,b);//值传递, 实参的值不会改变
  printf("%d,%d,%d,%d\n",a,b,*pa,*pb);
  // 10,20,10,20
  return 0;
}
// 对形参的改变不会引起实参的改变
void swap(int p1, int p2)
{
  int tmp;
  tmp=p1;
  p1=p2;
  p2=tmp;
}
```

main 函数: 调用 swap 前后

实参	地址	内容
a	3000	10
	3001	
b	4000	20
	4001	

swap 函数: 形参是局部变量, 不会影响实参的值

形参	地址	内容
p1	5000	10 → 20
	5001	
p2	6000	20 → 10
	6001	

形参实参地址 “不同”。

地址传递与值传递小结

- **地址传递:** 指针变量作为函数参数, 形参和实参指向同一存储单元。
  函数内对形参指向内容的改变**会**引起实参指向内容的改变。
- **值传递:** 普通变量作为函数参数, 形参和实参处于不同存储单元。对形参的改变**不会**引起实参的改变。

```c
// 地址传递
void swap(int *p1, int *p2)
{
  int tmp;
  tmp=*p1;
  *p1=*p2;
  *p2=tmp;
}
```

```c
// 值传递
void swap(int p1, int p2)
{
  int tmp;
  tmp=p1;
  p1=p2;
  p2=tmp;
}
```

常见错误

```
//对p1,p2值交换而不是交换p1,p2指向的内容
//p1,p2是局部变量,不会引起实参值的改变
void swap(int *p1, int *p2)
{
  int *tmp;
  tmp=p1;
  p1=p2;
  p2=tmp;
}
```

```
// tmp使用前，要有指向
void swap(int *p1, int *p2)
{
  int *tmp;//没有指向，即未赋值
  *tmp=*p1;//错！ tmp没有指向就间接访问
  *p1=*p2;
  *p2=*tmp;
}
```

借用地址传递, 使函数“返回”多个值

```c
int fun(int *p1, int *p2, int *p3);
int main()
{
  float a=10,b=20,c=30;
  int r=fun(&a,&b,&c);
  printf("%d,%f,%f,%f\n",r,a,b,c);
  // 4,1,2,3
  return 0;
}
```

```c
//对指针指向内容的改变, "返回"给调用者
int fun(int *p1, int *p2, int *p3)
{
  *p1=1;
  *p2=2;
  *p3=3;
  return 4;
}
```

不能混用不同类型的指针变量

```
int a=10, *p1=&a;
float b=20,*p2=&b;
p1=&b; // 错误,虽然编译器不报错
p2=&a; // 错误,虽然编译器不报错
p2=p1; // 错误,虽然编译器不报错
```

不同数据类型存错空间长度不同,因此,不能混用。

不同类型的指针变量	**地址**	**内容**
p1	5000	10
	5001	
p2	6000	20
	6001	
	6002	
	6003	

数组与指针

- `int a[10],*p=a;//p指向数组元素的首地址`
- `p=a; //等效p=&a[0];`
- `p+i; //p+i指向数组的第i个元素,*(p+i)=a[i]`
- `p++; //p指向数组的下一个元素,注意越界`
- `p--; //p指向数组的上一个元素,注意越界`
- `a=a+1; //错误, a是地址常量`
- `p=p+1; //正确, p是地址变量`

p=a	a+0	→	a[0]
p+1	a+1	→	a[1]
p+2	a+2	→	a[2]
...	...	...	...
p+i	a+i	→	a[i]
...	...	...	...
p+n-1	a+n-1	→	a[n-1]

用数组名作函数参数(地址传递,数组名是数组首地址)

```
//编译器认为a是指针变量
void fun(int a[],int n)
// void fun(int *a,int n) //自动转化为此形式
{
  a[0]=10;
  a[1]=20;
  a[3]=30;
  // 因为a被当作指针变量, 因此下面的语句是合法的
  a++; // 使a指向a[1]
  *a=20; // a[1]=20
}
```

```
// 指针与数组通用
void fun(int *a,int n)
{
    *a=10; // a[0]=10
    a++;
    *a=20; // a[1]=20
    a++;
    *a=30; // a[3]=30
}
```

注

用数组名作函数参数时,编译器会把数组名自动转化为指针变量处理。函数内既可以按指针变量间接访问数组元素,也可以按数组常规访问数组元素。

例: 将数组 a 中 n 个整数按相反顺序存放.

```c
#define N 100
void inv(int *x,int n);
int main()
{
   int i,n=6,a[N]={1,2,3,4,5,0};
   inv(a,n);
   for(i=0;i<n;i++)
     printf("%d␣",a[i]);
     // 0,5,4,3,2,1
   printf("\n");
   return 0;
}
```

```c
// n是数组长度
void inv(int *x,int n)
{
   int *p1,*p2,tmp;
   p1=x; // 指向首元素
   p2=x+n-1; // 指向最末元素
   //while(p1!=p2)//不适于n为偶数
   while(p1<p2)
   { //互换两个指针的目标变量值
      tmp=*p1;
      *p1=*p2;
      *p2=tmp;
      p1++; // 右移一元素
      p2--; // 左移一元素
   }
}
```

字符串常量

```
char s1[]="abcd"; // 数组长度为4(不包含末尾的'\0'), s1[4]='\0'
char s2[10]="abcd"; // 数组长度为10, s2[4]='\0', s2[5]到s2[9]未赋值

// 指针就是字符串常量的首地址
char *s3="abcd";  // 数组长度为4(不包含末尾的'\0'), s3[4]='\0'
// s3与s1等效, 但是与s2不等效。

s1="1234"; // 错误, 数组名是地址常量
strcpy(s1,"1234"); // 合法, 但是s1要大于等于"1234"的长度+1
s3="1234"; // 正确, 指针是变量
```

注意:

数组有长度概念,指针没有长度概念。系统维护一个存储空间,存储字符串常量,因此每个字符串常量具有唯一地址。

例: 实现自己的字符串拷贝函数

```
void copy(char to[], char from[]);
int main()
{
   char a[81],b[81];
   gets(a); // 输入字符串
   copy_string(b,a);
   puts(b);
}
// 数组版本
void copy(char to[], char from[])
{
   int i=0;
   while(from[i]!='\0')
   {
      to[i]=from[i];
      i++;
   }
   to[i]='\0'; // 使字符串以'\0'结束
}
```

```
// 指针版本
void copy(char *to, char *from)
{
   // 等效while(*from!='\0')
   // 或while(*from!=0)
   while(*from)
   {
      *to=*from;
      from++;
      to++;
   }
   *to='\0';// 不要忘记此语句
}
```

字符串数组

```
#define N 100
char name[N][81]; // N个字符串
int n=10,i; // n是实际字符串个数

for(i=0;i<n;i++) // 输入n个字符串
{
   gets(name[i]);
}

for(i=0;i<n;i++) // 输出
{
   puts(name[i]);
}
```

例: 选择法排序字符串数组

```
#define N 100
void sort(char name[][81],int n);
int main()
{
  char name[N][81]; // N个字符串
  int n=10,i; // n是实际大小

  for(i=0;i<n;i++) // 输入n个字符串
  {
    gets(name[i]);
  }
  sort(name,n); // 排序
  for(i=0;i<n;i++) // 输出
  {
    puts(name[i]);
  }
}
```

```
// 用选择法排序(从小到大)
void sort(char name[][81],int n)
{
  char tmp[81];
  int i,j,k;
  for(i=0;i<n-1;i++)
  {
    k=i;// 未排序部分最小的字符串
    for(j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0)
        k=j;
    if(k!=i) // 交换字符串
    {
      strcpy(tmp,name[i]);
      strcpy(name[i],name[k]);
      strcpy(name[k],tmp);
    }
  }
}
```

lecture-15 主要内容

善于利用指针

1. 指针就是指向变量存储单元 (首) 地址的变量
2. 定义指针变量: int *p;
3. 变量的直接访问与间接访问
4. 地址传递与值传递
5. 数组与指针
6. 指针与字符串
7. 字符串数组

欢迎批评指正！