Pointers and Arrays

This lecture deals with Pointers and Arrays.

Chapter 5 K & R

Pointers and Arrays

A pointer is a variable that contains the address of a variable.
i.e. a pointer is a reference variable.

Pointers are much used in C, often their use is indispensible to express a computation also their use can lead to compact and efficient code.

Careless use of pointers can lead to chaos.

With discipline pointers can be used to achieve clarity and simplicity.

Arrays and pointers are closely related in C

Pointers

The declaration

```
    int   *p;
```

makes p a " pointer to int ".

The value of p is the address of a location in memory, the contents stored at that address are of type int.

int   *p; can be read in two ways
1. *p is an int. Here * is the dereferencing operator.
2. p is of type int *.

Note *p is an expression wheras p is a variable.
Both interpretations are equivalent.

& : Address-of operator

The statement

```
        p = &x;
```

makes p to point to x.

So the expressions x and *p are equivalent.

```
main()
{
        int    x;
        int    *p;
        p  = &x;
        *p = 2;
        printf ("%d\n", x);
}
```

This program prints 2.

Pointers Declarations and Use

```
int   x  =  1,  y  =  2,  z[10];
int   *ip;          /*  ip is a pointer to int */

ip  = &x;           /*  ip now points to x  */
y  =  *ip;          /*  y is now 1  */
*ip = 0;            /*  x is now  0  */
ip  =  &z[0];       /*  ip now points to z[0]  */
```

Declaration of a variable mimics the syntax of expressions in which the variable might appear.

e.g.

```
double    *dp, atof(char  *);
```

Pointers and Referential Transparency

General Rule
If ip points to integer x, then *ip can occur in any context where x could. So *ip just stands for x.

```
*ip  =  *ip  + 10;     /* increments *ip by 10 */
```

It is the same as x = x + 10;

```
y  =  *ip  +  1        /* same as   y  =  x + 1  */

*ip  + =  1            /*  same as   x  + =  1  */

++*ip                  /* same as  ++x  */

(*ip)++                /* same as  x++  */

*ip++                  /* same as  *(ip++)   */
```

Assigning Pointers

```
int   *ip,  *iq;    /* declares ip and iq as integer pointers */

      iq   =  ip;    /*  copies contents of ip into iq so that iq
                         points to whatever ip pointed to */
```

The declaration

```
      int *  ip, iq;
```

does not declare both ip and iq as integer pointers, it declares ip as integer pointer and iq as an integer variable.

Pointers and Function Arguments

```
void swap( int x,  int y)        /* wrong */
{
        int  temp;

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

The call
swap (a, b);
will NOT affect the arguments a and b as the function only swaps only copies of a and b.

Pointers and Function Arguments

```
void swap( int *px,  int *py)  /* interchange *px and *py */
{
      int  temp;

     temp  =  *px;
      *px =  *py;
      *py  =  temp;
}
```

The call
swap (&a, &b);
will interchange the values of a and b.

Pointer arguments enable a function to access and change objects in the function that called it.

Pointers and Arrays

Arrays and Pointers are closely related.

Array subscripting can be achieved using pointers, often the pointer version is more efficient.

Declaration

```
        int   a[10];
```

defines an array a of size 10, a block of 10 consecutive objects named a[0], a[1],..., a[9]

a[i] refers to the i-th element of the array.

```
int   *pa;                  /* pa is an int pointer  */
pa  =   &a[0];          /*  pa points to the element a[0]  */
x  =  *pa;               /* same effect as   x  =  a[0];     */
```

Pointer Arithmetic

```
int   *pa;    /* pa is an int pointer  */
pa  =   &a[0];      /*  pa points to the element a[0]  */
x  =  *pa;           /* same effect as   x  =  a[0];     */
```

If   pa points to an element of an array,
pa+1 points to the next element,
pa+i points to i elements after pa
pa-i points i elements before pa

if pa points to a[0]
*(pa+1) refers to the contents of a[1]
*(pa+i) refers to the contents of a[i]

The above pointer arithmetic true regardless of the type or size of the variables in the array.

Pointer Arithmetic and Array Indexing

```
pa  =  &a[0];                /* pa and a have identical values  */
```

The name of an array is a synonym for the location of the initial element.

pa = &a[0]; can also be written as
pa = a;

a[i] can also be written as *(a+i).

In evaluating a[i], C converts it to *(a+i) immediately;
so the two forms a[i] and *(a+i) are equivalent.

Applying & operator to both parts of above equivalence,
&a[i] and (a+i) are also identical.

Pointer Arithmetic and Array Indexing

If pa is a pointer expression may use it with a subscript,
pa[i] is identical to *(pa+i).

Array and index-expression is equivalent to a pointer and offset-expression.

A basic difference between an array name and a pointer:

A pointer is a variable so pa = a and pa++ are legal.
But an array name is not a variable, so constructions such as a = pa and a++ are illegal.

Arrays as function arguments - 1

```
/*  strlen:  return length of a string s   */

int   strlen (char *s)
{

        int   n;

        for  (n  = 0;  *s  !=  '\0' ;  s++)
                n++;
        return  n;
}
strlen ("hello, world");     /* a string constant  */
strlen (array);              /* char array[100];  */
strlen (ptr);                /*  char *ptr;        */
```

The above calls to strlen() are legal

Arrays as function arguments - 2

When an array name is passed to a function what is passed is the location of the initial argument. Within the called function this argument is a local variable and so array name parameter is a pointer, i.e, a variable containing an address.

As formal parameters in a function definition

```
char   s[]   and      char *s
```

are equivalent.

char *s is preferred as its says more explicitly that the parameter is a pointer.

When an array name is passed to a function the function can at its convenience treat it either as an array or as a pointer.

Array, Pointer duality in function argument

Arrays as function arguments - 3

It is possible to pass part of an array to a function by passing a pointer to the beginning of the subarray.

If a is an array

```
f(&a[2])     and      f(a+2)
```

both pass to the function f the address of the subarray that starts at a[2].

Within f the parameter declaration can read

```
f(int  arr[]) { ...}     or    f(int    *arr) {...}
```

Arrays and Pointers

p[-1], p[-2], ... are syntatically legal and refers to elements that immediately precede p[0] if the elements exist.