Pointers and Arrays (contd)

This lecture deals with Pointers and Arrays.

Chapter 5 K & R

Pointer Address Arithmetic

If p points to some element of an array, then

p++ increments p to point to the next element,
p+= i increments p to point to i elements beyond

C is consistent and regular in its approach to address arithmetic; its integration of pointers, arrays, and address arithmetic is one of the strengths of the language.

Pointer arithmetic is consistent: if p is a pointer to int (float) then p++ advances to next int (resp. float)

All pointer manipulations automatically take into account the size of the object pointed to.

Pointer Address Arithmetic

The following pointer operations are illegal

To add two pointers

To multiply or divide or shift or mask pointers

To add float or double to pointers

To assign a pointer of one type to a pointer of another type without a cast (except for void *)

Character Pointers and Functions

A string constant
" This is a string "
is an array of characters.

In the internal representation of a string , the array is terminated with the null character '\0'.

Storage requirement is one greater than the number of characters between the double quotes.

The double quotes are string delimiters and are not a part of the string.

Character Pointers and Functions

printf ("hello, world\n");

Here a string constant is passed as an argument to the function printf(). The access to it is through a character pointer.

So printf() receives a pointer to the beginning of a character array.

A string constant is accessed by a pointer to its first element.

Character Pointers and Functions

```
char   *pmessage;          /* pmessage is a char pointer */

pmessage  =  "now is the time";
```

The above statement assigns to pmessage a pointer to the character array.

```
char  amessage[]  =  "now is the time";   /* an array */
char  *pmessage  =  "now is the time";   /* a  pointer */
```

Notice the difference between the above definitions.

strcpy

```
/*  strcpy:  copy t to s,  array subscript version  */

void strcpy(char  *s,   char  *t)
{
      int   i;

      i  =  0;
      while  ((s[i]  =  t[i])   !=  '\0')
            i++;
}
```

strcpy

```
/*  strcpy:  copy  t to s,   pointer version 1 */

void strcpy(char  *s,   char  *t)
{
      while  ((*s  =  *t)   !=  '\0')   {
            s++;
            t++;
      }
}
```

strcpy

```
/*  strcpy:  copy  t  to s,   pointer version 2 */

void strcpy(char  *s,   char  *t)
{
      while  ((*s++  =  *t++)   !=  '\0')
            ;
}
```

Ultimate strcpy - C idiom

```
/*  strcpy:  copy  t  to s,   pointer version 3 */

void strcpy(char  *s,   char  *t)
{
      while  (*s++   =   *t++)
            ;
}
```

strcmp - array version

```
/*  strcmp:  return <0 if  s < t,  0 if s = t,  >0 if s> t */

void strcmp(char  *s,   char  *t)
{
      int   i;

      for  (i = 0;  s[i]  ==  t[i];  i++)
            if (s [i]  ==  '\0')
                  return 0;
      return s[i]  -  t[i];
}
```

strcmp - pointer version

```
/*  strcmp:  return <0 if  s < t,  0 if s = t,  >0 if s> t */

void strcmp(char  *s,   char  *t)
{
      int   i;

      for  (;   *s  ==  *t;  s++, t++)
            if (s [i]  ==  '\0')
                  return 0;
      return  *s  -  *t;
}
```

Some C Idioms

```
*p++ = val;       /* push val onto stack */
val  = *--p;      /* pop top of stack into val */


while (*s++ = *t++)    /* for copying a string */
        ;

while ((c = getchar()) != EOF)
```

Pointer Arrays

Motivation

Sorting lines of text of different lengths. (K&R page 107)

char *lineptr[MAXLINES];
declares linepointer is an array of size MAXLINES elements,
each element of which is a pointers to char.

Thus
lineptr[i] is a character pointer and

*lineptr[i] is the character it points to

swap

```
/*  swap:  interchanges  pointers v[i] and v[j]  */

void swap  (char  *v[],   int   i,  int   j)
{
      char       *temp;

      temp  =  v[i];
      v[i]  =  v[j];
      v[j]  =  temp;

}
```

Multidimensional Arrays

C allows arrays of any dimesnsion to be defined.
Multidimensional arrays are much less used than arrays of pointers.

A two dimenional array can be declared in the same way as a one dimensional array is declared.

```
int a[4][5];
```

declares array a to be a two dimensional array of 4 rows and 5 columns and each position contains an integer value.

Initialization of a declared array is similar to the one dimensional case

```
e.g.  int a[2][3] = {
                      {1, 2, 3},
                      {4, 5, 6}
                    };
```

Multidimensional Arrays

If a two dimensional array is passed to a function, the parameter declaration in the function must include the number of columns.

e.g. if an array declared as

```
char daytab[2][13];
```

is to be passed to a function f, the declaration of f would be

```
f(int daytab [2][13]) { ... }

f(int daytab [ ][13]) { ... }

f(int (*daytab [13])) { ... }
```

In general the first dimension of an array is free, all others have to be specified.

Pointers vs Multidimensional Arrays

```
int   a[10][20];

int   *b[10];
```

Declarations

```
int   *f( )

int   (*pf )( )

char      **argv

int   (*daytab) [13]

void      *comp( )

void      (*comp) ( )

char      (*(*x( ) )[ ] ) ( )

char      (*(*x[3] )( ) ) [5]
```