CS113: Lecture 8

Topics:

- Pointers and Arrays
- Pointer Arithmetic

String reversal revisited

```
#include <string.h>

void reverse( char *s, char *t )
/* reverses s, placing result in t */
/* s is not changed */
{
   int len, i;

   len = strlen( s );

   for( i = 0; i < len; i++ )
   {
      t[i] = s[len - 1 - i];
   }
   t[len] = 0;
}

void main()
{
   char s[20], t[20];
   strcpy( s, ".desrever ma I" );
   printf( "s before reversal: %s\n", s );
   reverse( s, t );
   printf( "t after reversal: %s\n", t );
}
```

Pointers and Arrays

- This declaration defines an `int` array `a` of size 10.

  `int a[10];`

- If `pa` is a pointer to an integer, i.e.,

  `int *pa;`

  then the assignment

  `pa = &a[0];`

  sets `pa` to point to element zero of `a`.

- When does `x = *pa;` make sense – what does the type of `x` have to be? What does it do?

- If `pa` points to an element of an array, then (by definition) `pa + 1` points to the next element.

  In general, `pa + i` points to the `i`th element after the element pointed to by `pa`.

- Example.

```
int a[4] = { 0, 1, 2, 3 };
int *p;
p = &a[0];
printf( "%d\n", *(p + 2));
scanf( "%d", p + 3 );
printf( "You typed: %d\n", a[3] );
```

More on Pointers and Arrays

- In fact, the name of an array is a synonym for the address of the initial element. As an example, when we have the declarations

  ```
  int a[10];
  int *pa;
  ```

  `&a[0]` is the same as `a`, and thus `pa = &a[0];` is the same as `pa = a;`.

- This is why the changes to an array made by a function persist: we were simply passing in a pointer to the first (zero indexed) element of the array.

- Accordingly, for any expression `b` of type `int *`, `b[i]` can always be written as `*(b + i)`, and vice-versa.

  For example, given the above declarations:

  `a[i]` and `*(a + i)` are equivalent

  `pa[i]` and `*(pa + i)` are equivalent

- Note that an array name (like `a` assuming the above declarations) is *not* a variable, so statements like `a = pa;` and `a++;` are illegal. (You also don't want to form the expression `&a`.)

Practice: Pointers and Arrays

```
void main()
{
  int a[4] = { 0, 1, 2, 3 };
  int *pa;

  pa = a + 1;
  printf( "%d\n", *pa );
  printf( "%d\n", pa[2] );
  pa++;
  printf( "%d\n", pa[0] );
  scanf( "%d", pa + 1 );
  printf( "You typed: %d\n", a[3] );
}
```

Pointer Arithmetic

- Pointer addition: pointer plus `int`

  Saw that if a pointer `p` points to an element of an array, then `p + i` is a pointer (of the same type) pointing to the `i`th element after the element pointed to by `p`.

- Pointer subtraction: pointer minus pointer

  If `p` and `q` point to elements of the same array, then `q - p` gives the number of elements between `p` and `q`.

- Pointer comparison: pointer relation pointer

  Permissible relations: `==, !=, <, <=, >, >=`

  If `p` and `q` point to elements of the same array, then

  `p < q`

  is true if `p` points to an earlier member of the array than `q` does.

- Note: CAN'T add two pointers, or perform any sort of multiplication, etc.

Example: Elements before zero

(Example from PCP)

```
void main()
{
   int array[] = { 4, 5, 8, 9, 0, 1, 3, 2 };
   int index;

   index = 0;
   while( array[index] != 0 )
      index++;

   printf( "Number of elements before 0: %d\n", index );
}
```

```
void main()
{
   int array[] = { 4, 5, 8, 9, 0, 1, 3, 2 };
   int *array_ptr;

   array_ptr = array;
   while(( *array_ptr ) != 0 )
      array_ptr++;

   printf( "Number of elements before 0: %d\n",
           array_ptr - array );
}
```

strlen implementations

(from K&R)

```
int strlen( char *s )
{
   int n;
   for( n = 0; *s != '\0'; s++ )
      n++;
   return n;
}


int strlen( char *s )
{
   char *p = s;

   while( *p != '\0' )
      p++;
   return( p - s );
}
```

strcpy implementations

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

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


/* pointer version 1 */
void strcpy( char *s, char *t )
{
   while(( *s = *t ) != '\0' )
   {
       s++;
       t++;
   }
}
```

strcpy implementations, takes 3 and 4

```
/* pointer version 2 */
void strcpy( char *s, char *t )
{
   while(( *s++ = *t++ ) != '\0' )
      ;
}


/* pointer version 3 */
void strcpy( char *s, char *t )
{
   while( *s++ = *t++ )
      ;
}
```

Two strcmp implementations

```
/* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
int 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] );
}


int strcmp( char *s, char *t )
{
   for( ; *s == *t; s++, t++ )
   {
      if( *s == '\0' )
         return 0;
   }
   return( *s - *t );
}
```

What does == do here?

```
void main()
{
   char s[20];
   strcpy( s, "Hello." );
   if( s == "Hello." )
   {
      printf( "Equal.\n" );
   }
   else
   {
      printf( "Not equal.\n" );
   }
}
```