CS113: Lecture 5

Topics:

- Arrays
- Strings

Arrays

- Often, programs use homogeneous data. For example, if we want to manipulate some grades, we might declare

  `int grade0, grade1, grade2, grade3;`

- If we have a large number of grades, it becomes cumbersome to represent/manipulate the grades, when each grade has a unique identifier. (How to find average? maximum? etc.)

- *Arrays* (feature of most every programming language) allow us to refer to a number of instances of the same data type, using a single name.

- For example,

  `int grade[4];`

  makes available the use of `int` variables `grade[0]`, `grade[1]`, `grade[2]`, `grade[3]`, in a program.

- Note that arrays in C are *zero-indexed* – numbering begins at zero. If the size of an array `a` is `SIZE`, then the first accessible element of `a` is `a[0]`, and the last is `a[SIZE - 1]`.

- Now, to access elements of this array, we can write `grade[expr]`, where `expr` is any expression (evaluating to an integer within the appropriate range).

Array example: grades

```
#include <stdio.h>

void main()
{
   int grades[11], num_grades = 0;
   int i;
   float sum, average;

   printf( "Please enter up to 10 grades, "
           "terminated by 0.\n" );
   while( 1 )
   {
      scanf( "%d", &(grades[num_grades]));
      if( grades[num_grades] == 0 ) break;
      num_grades++;
   }

   /* Compute average */
   sum = 0;
   for( i = 0; i < num_grades; i++ )
   {
      sum += grades[i];
   }
   /* Assume more than one grade entered */
   average = sum / num_grades;
   printf( "The average of the grades is: %f",
           average );
}
```

Arrays in C

- No bounds checking.

  You (the programmer) are responsible for making sure that you only access array elements `0` through `N - 1` for an array of size `N`.

  A program that writes to "out-of-bounds" locations will compile and often run – beware! Writing to such invalid locations corrupts memory, sometimes the values of other variables.

- The size of an array must be a constant. (For now...)

  Here, "constant" means that the value can be determined at compile-time.

```
void func( int size )
{
    int b[size];          /* bad */
    int g[(8 * 5) + 2];   /* good */
}
```

- C has no internal mechanism for compying or comparing arrays.

  If `a`, `b` are arrays of the same type:

  – expression `a = b;` is illegal

  – expression `a == b` is legal, but it doesn't check to see if the elements of `a` match those of `b`

Example: Change-and-sum

```
#include <stdio.h>

int change_and_sum( int *a, int size )
{
   int i, sum = 0;
   a[0] = 100;
   for( i = 0; i < size; i++ )
      sum += a[i];
   return sum;
}

void main()
{
   int a[5] = { 0, 1, 2, 3, 4 };
   printf( "Sum of elements of a: %d\n",
           change_and_sum( a, 5 ));
   printf( "Value of a[0]: %d\n", a[0] );
}
```

Notice:

- Initialization of array
- Array passed as parameter - along with the size
- Changes made to array persist

Example: sorting numbers

```
void sort_ints( int *a, int size )
{
   int i, j, k, temp;
   for( i = 0; i < size; i++ )
   {
      /* find largest elt. of
         a[i], ..., a[size-1] */
      k = i;
      for( j = i + 1; j < size; j++ )
      {
         if( a[j] > a[k] )
            k = j;
      }

      /* swap a[i], a[k] */
      temp = a[k];
      a[k] = a[i];
      a[i] = temp;
   }
}

void main()
{
   int a[6] = { 3, 2, 8, 1, 5, 9 }, i;

   sort_ints( a, 6 );
   for( i = 0; i < 6; i++ )
      printf( "%d\n", a[i] );
}
```

Strings

- Strings are one-dimensional arrays of `char`s.

- By convention, a string in C is terminated by the null character, `'\0'`, or `0`.

- String constants (such as those passed to the function `printf`) are enclosed in double quotes.

- When allocating `char` arrays that will hold strings, make sure you allocate enough space!

Example: “Double” printing

```
#include <stdio.h>

void dprint( char *s )
{
   int i;
   for( i = 0; s[i] != 0; i++ )
      printf( "%c%c", s[i], s[i] );
}

void main()
{
   dprint( "Hi there" );
}
```

Example: "squeeze" function

(Based on an example from K&R)

```
#include <stdio.h>

/* squeeze deletes all instances of the
   character c from the string s. */
void squeeze( char *s, int c )
{
   int i, j;

   for( i = j = 0; s[i] != 0; i++ )
   {
      if( s[i] != c )
      {
         s[j] = s[i];
         j++;
      }
   }
   s[j] = 0;
}

void main()
{
   char s[100];
   strcpy( s, "Clzzeazn mez zup!" );
   printf( "Before squeeze: %s\n", s );
   squeeze( s, 'z' );
   printf( "After squeeze: %s\n", s );
}
```

String handling functions

These are from `string.h`. See Appendix B3 of K&R for an exhaustive list.

- `char *strcat( char *s1, char *s2 );`

  Takes two strings as arguments, concatenates them, and puts the result in `s1`. The programmer must ensure that `s1` points to enough space to hold the result. The string `s1` is returned.

- `char *strcpy( char *s1, char *s2 );`

  The string `s2` is copied into `s1`. Whatever exists in `s1` is overwritten. It is assumed that `s1` has enough space to hold the result. The value of `s1` is returned.

- `int strcmp( char *s1, char *s2 );`

  Integer is returned that is less than, equal to, or greater than zero, depending on whether `s1` is lexicographically less than, equal to, or greater than `s2` (respectively).

A good exercise is to implement these functions yourself.

The strcmp ordering: think dictionary

From "lowest" to "highest":

```
"1"
"128"
"16"
"2"
"32"
"4"
"64"
"8"
"Avocado"
"Can"
"Can not"
"Can't"
"Cannot"
"Cantor"
"Lime"
"apple"
"banana"
"c"
"c language"
"c programmer"
"cantaloupe"
```

Example: Reversing a string

```
#include <string.h>

void reverse( char *s )
{
   int halflen, len, i;
   char temp;

   len = strlen( s );
   halflen = len / 2;

   for( i = 0; i < halflen; i++ )
   {
      /* swap s[i] and s[len - 1 - i] */
      temp = s[i];
      s[i] = s[len - 1 - i];
      s[len - 1 - i] = temp;
   }
}

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

Multidimensional arrays

- Arrays can have more than one dimension.

- Example of declaring a two-dimensional array of `int`s:

  ```
  int b[3][7];
  ```

  Makes available 21 ints for use: `b[i][j]` where `i` ranges from 0 to 2, and `j` ranges from 0 to 6.

- Can also declare three-dimensional, etc. arrays.

  ```
  int c[2][4][10];
  ```

Arrays of Strings

```
void get_string( char s[] )
{
   scanf( "%s", s );
   printf( "Length of your string: " );
   printf( "%d\n", strlen( s ));
}


void main()
{
   char arr[8][81];
   get_string( arr[1] );
   printf( "You typed the string: %s\n", arr[1] );
   printf( "The first character you typed was: " );
   printf( "%c\n", arr[1][0] );
}
```

Notice:

- Two-dimensional array of `chars` acts as array of strings (of size 8): `arr[0]`, ..., `arr[7]`
- `scanf( "%s", ... );` used to read string; reads all characters up to first whitespace character
- To refer to a specific character of one of the strings `arr[i]`, tack on another index: `arr[1][0]` for instance refers to the first (zero-indexed) character of the string `arr[1]`

A comment curiosity

```
#include <stdio.h>

void main()
{
   char s[10];
   strcpy( s,  /* /* */ " */ " /* " /* */  );
   if( strcmp( s, " */ " ) == 0 )
   {
      printf( "This program was created by a compiler "
              "which does NOT nest comments.\n" );
   }
   else
   {
      printf( "This program was created by a compiler "
              "which does nest comments.\n" );
   }
}
```