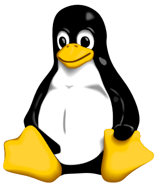

C Programming

Exercises With Arrays and Strings

1 Background

Arrays are collections of *elements*. The elements go into memory, one after the other. If an array is declared as **int** *array*[5] then there are five elements; the first is *array*[0], the last is *array*[4].

1.1 Initialialising an Array

You can *initialise* an array when you *define* the array:

```
int array[ 5 ] = { 10, 20, 30, 40, 50 };
```

but you cannot *assign* multiple values to an array *after* you have defined it:

```
int array[ 5 ];
array = { 10, 20, 30, 40, 50 }; // BIG ERROR!
```

Notice the difference between the terms *assign* and *initialise*.

1.2 Assigning to elements of an array

After the array is defined, we can assign values to individual elements:

```
int array[ 5 ];
array[ 0 ] = 10;
array[ 1 ] = 20;
array[ 2 ] = 30;
array[ 3 ] = 40;
array[ 4 ] = 50;
```

and we can use these elements just as we would an ordinary variable:

```
printf( "The third element is %d\n", array[ 2 ] );
```

However, there the only real advantage of using arrays is so that we can use loops to process them. You could imagine how silly it would be to write a program to fill all elements of this array with tens:

```
int tens[ 10000 ];
tens[ 0 ] = 10;
tens[ 1 ] = 20;
// ... 9997 more assignments ...
tens[ 9999 ] = 100000;
```

It would be much smarter to use a loop. With arrays, we usually use **for** loops. We could fill our *tens*[] array with this **for** loop:

```
int i, tens[ 10000 ];
for ( i = 0; i < 10000; ++i )
        tens[ i ] = ( i + 1 ) * 10;
```

Notice that we could use a **while** loop to do the same thing:

```
int tens[ 10000 ];
int i = 0;
while ( i < 10000 ) {
        tens[ i ] = ( i + 1 ) * 10;
        ++i;
}
```

1.3 Comparing for and while loops

for loop:

```
for ( ⟨init⟩; ⟨test⟩; ⟨update⟩ ) {
        ⟨body of loop⟩
}
```

example:

```
for ( int i = 0; i < 5; ++i )
        printf( "%d\n", array[ i ] );
```

while loop:

```
⟨init⟩;
while ( ⟨test⟩ ) {
        ⟨body of loop⟩
        ⟨update⟩;
}
```

example:

```
int i = 0;
while ( i < 5 ) {
        printf( "%d\n", array[ i ] );
        ++i;
}
```

2 Strings

In the C programming language, a *string* is just an array of characters:

```
char string[ 8000 ];
```

2.1 The null character marks the end of a string

The string library routines (such as *strlen*()) assume that there is a null character `'\0'` at the end of each string. The null character is used as a marker to see where the end of the string is.

You always need to leave room for the null character. The declaration of *string*[] above can hold a string with a maximum of 7999 characters, since the last character in the array should be the null character.

It is okay to have some of the string unused:

```
char string[ 8000 ] = "Hello";
```

2.2 Printing strings

printf() can print a string using the `"%s"` format string:

```
printf( "The string contains %s\n", string );
```

The output if *string* still contains `"Hello"` is:

```
The string contains Hello
```

2.3 Finding the length of a string

To find out how many characters there are in a string, you can use the string library function *strlen*(). You need to **#include** <*string.h*> to use *strlen*().

If the string *string* defined above is initialised as shown, then

```
printf( "String length of %s is %d\n", string, strlen( string ) );
```

The output would be:

```
String length of Hello is 5
```

3 Exercises

1. Write a program that defines the array

   ```
   int array[ 5 ];
   ```

   and which *initialises* it so that each element holds a value equal to its own index.

2. Write a program that defines the array

   ```
   int array[ 5 ];
   ```

   and which *assigns* values to its elements so that each element holds a value equal to its own index, *without* using a loop.

3. Write a program that defines the array

   ```
   int array[ 5 ];
   ```

   and which *assigns* values to its elements so that each element holds a value equal to its own index, using a **for** loop.

4. Write a program that defines the array

   ```
   int array[ 5 ];
   ```

   and which *assigns* values to its elements so that each element holds a value equal to its own index, using a **while** loop.

5. Write a program to that defines the string

   ```
   char name[ 8000 ];
   ```

   and reads a line of text from standard input using the Standard I/O library function *gets*(), then prints it out to standard output.

6. Modify your program to loop through each character of the string and print out each character individually using *putchar*(). Again, don't forget to **#include** <*stdio.h*>.