Example 9.1. Assume there are 50 students in the class, and their grades on the first exam are stored in an array. Write a function that calculates and returns class average. Write a second function that finds and returns the highest grade on the exam.

Solution: Assuming the grades vary from 0 to 100, the data could be stored in **char**, **short** or **long** format. To be compatible with the ARM architecture, we will create a 32-bit array. In this example, we will not worry about how grades are entered, but assume there is data in this array.

```
long Grades[50];
```

To calculate average, we sum the values and divide by the number of students. The return statement will return the 32-bit calculation of sum/50. This will never create an error, because the sum of 50 numbers ranging from 0 to 100 will always be less than 4 trillion (2^{32}) and the average of these numbers will never go below 0 or above 100.

```
long Average(void){long sum,i;
  sum = 0;
  for(i=0; i<50; i++){
    sum = sum+Grades[i];
  }
  return (sum/50);
}
```

Program 9.3. A function that calculates the average of data stored in an array.

To make this more general, we will pass in the array and the size.

```
long Average(long class[],long size){
long sum,i;
  sum = 0;
  for(i=0; i<size; i++){
    sum = sum+class[i];  // add up all values
  }
  return (sum/size);
}
```

The array parameter is actually a pointer to the array. So to use this new function we call

```
MyClassAverage = Average(Grades, 50);
```

Similarly to find the largest we will search the array. We initialized the result parameter to the smallest possible answer, and then every time we find one larger, we place the result parameter.

```
long Max(long class[],long size){
long largest,i;
  largest = 0;  // smallest possible value
  for(i=0; i<size; i++){
    if(class[i] > largest){
      largest = class[i];  // new maximum
    }
  }
  return (largest);
}
```

To use this new function we call

```
HighestsScore = Max(Grades, 50);
```

Example 9.2. Design an exponential function, $y = 10^x$, with a 32-bit output.

Help

Solution: Since the output is less than 4,294,967,295, the input must be between 0 and 9. One simple solution is to employ a constant word array, as shown in Figure 9.3. Each element is 32 bits. In assembly, we define a word constant using **DCD**, making sure in exists in ROM.

In C, the syntax for accessing all array types is independent of precision. See Program 9.4. The compiler automatically performs the correct address correction. We will assume the input is less than or equal to 9. If **x** is the index and **Base** is the base address of the array, then the address of the element at **x** is **Base+4*x**. In assembly, we can access the array using indexed addressing. We will assume the Register R0 input is less than or equal to 9.

0x00000134	1
0x00000138	10
0x0000013C	100
0x00000140	1,000
0x00000144	10,000
0x00000148	100,000
0x0000014C	1,000,000
0x00000150	10,000,000
0x00000154	100,000,000

0x00000158	1,000,000,000

Figure 9.2. A word array with 10 elements. Addresses illustrate the array is stored in ROM. Each 32-bit value requires 4 bytes of storage. The actual values for the addresses depends on where the compiler chooses to place the array, but in all cases the values are stored at consecutive locations in memory.

```
      AREA  |.text|,CODE,READONLY,ALIGN=2
Powers DCD  1, 10, 100, 1000, 10000
       DCD  100000, 1000000, 10000000
       DCD  100000000, 1000000000
; Input: R0=x    Output: R0=10^x
power  LSL R0, R0, #2   ; x = x*4
       LDR R1, =Powers  ; R1 = &Powers
       LDR R0, [R1, R0] ; y=Powers[x]
       BX  LR
```

```
const unsigned long Powers[10]
  ={1,10,100,1000,10000,
    100000,1000000,10000000,
    100000000,1000000000};
// input x must be less than 10
 unsigned long power(unsigned long x){
  return Powers[x];
}
```

Program 9.4. Array implementation of a nonlinear function.

In general, let **n** be the precision of a zero-origin indexed array in bytes. If **I** is the index and **Base** is the beginning address of the array, then the address of the element at **I** is

```
Base+n*I
```

The origin, **o**, of an array is the index of the first element. The origin of a zero-origin indexed array is zero (**o**=0). In general, the address of the element at **I** is

```
Base+n*(I-o)
```

Help