McMaster University
Department of Computing and Software
Dr. W. Kahl

SFWR ENG 2S03
Exercise Sheet 7
Solution Hints

SFWR ENG 2S03 — Principles of Programming

26 October 2006

Exercise 6.2 — Please finish!

Exercise 7.1 — Array Mapping (45% of Midterm 4, 2003 — new (d))

Let the following functions be given:

```
double square(double x) { return x * x; }
double sqrt(double x);                    /* from math.h */
```

You are to produce a function *arrayMap* that has the following features:

- It takes as arguments a one-dimensional array a, the size n of the array a, and a function f that could be for example *square* or *sqrt* from above.
- For each array element, *arrayMap* applies its argument function f to the contents of this array element, and then stores the result returned by f back in that array element.
- Each time the call to f sets *errno* to a non-zero value, the library function *perror* is used to print an informative message **that includes the current array index**.

  > void *perror*(const char *s);
  >
  > The routine *perror*() produces a message on the standard error output, describing the last error encountered during a call to a system or library function. First [...] the argument string *s* is printed, followed by a colon and a blank. Then the message and a new-line.

- *arrayMap* returns the number of calls to f that set *errno* to a non-zero value.

(a) [≈5%] Produce a prototype for *arrayMap*.

(b) [≈20%] Implement *arrayMap*.

(c) [≈20%] (**independent of (a) and (b)!**)

Assume *arrayMap* to be given, and also the following function for some kind of initialisation of an array of doubles:

void *arrayInit*(double *ar*[], int *size*);

Write a *main* function that contains a 3×10 array of doubles, initialises the first row through a call to the function *arrayInit*, copies the first row to the second and third rows using the library function *memmove*, and finally uses *arrayMap* and the functions from above to square all elements of the second row, and to change all elements of the third row to their respective square roots. After each call to *arrayMap*, the number of errors should be displayed.

> void * *memmove*(void * *dest*, const void * *src*, *size_t n*);
>
> The *memmove*() function copies *n* bytes from memory area *src* to memory area *dest*.

(d) [*New*] Produce a variant *arrayMapNew* that does not chage its argument in-place, but produces a new array to store the results of the argument function applications.

Solution Hints

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <math.h>
#define SIZE 10

int arrayMap(double ar[], int size, double (*f)(double x))
{
  int i, errCount = 0;
  char message[100];  /* need to allocate space for error message */

  for (i = 0; i < size; i++ ) {
    errno = 0;       /* functions like sqrt() don't clear it, only set it — we have to clear it. */
    ar[i] = (*f)(ar[i]);
    if (errno) { /* set by this call to (*f) */
      errCount++;
      sprintf(message, "arrayMap: error at index %d", i); /* prepare message */
      perror(message);
    }
  }
  return errCount;
}

void arrayInit(double ar[], int size) {
  int i;
  for (i = 0; i < size; i++)
     ar[i] = 10.0 * sin((double)i);  /* produces some numbers, including negative ones … */
}

void arrayPrint(double ar[], int size) {
  int i;
  for (i = 0; i < size; i++) printf("%3d: %20f\n", i, ar[i]);
  printf("\n");
}

double square(double x) { return x * x; }
double cube(double x) { return x * x * x; }

int main () {
  int i;
  double ar[3][SIZE];
  arrayInit(ar[0],SIZE);
  memmove(ar[1], ar[0], SIZE * sizeof(double));
  memmove(ar[2], ar[0], SIZE * sizeof(double));
  for (i=0; i<3; i++) arrayPrint(ar[i],SIZE);
  printf("First mapping: %d errors\n", arrayMap(ar[1], SIZE, square));
  printf("Second mapping: %d errors\n", arrayMap(ar[2], SIZE, sqrt));
  for (i=0; i<3; i++) arrayPrint(ar[i],SIZE);
  return 0;
```

}

"10" must not occur more than once; a constant or macro like *SIZE* must be used. sizeof needs to be used.

For (d), we first separate argument and result arrays in *arrayMap*, and then produce *arrayMapNew* as a wrapper around this new *arrayMap*.

One problem for *arrayMapNew* is that it now has two results to return: the error count, and the new array. We chose to return the new array via a reference parameter *result*. In case of unsuccessul memory allocation, the variable referenced by that parameter will be set to *Null*, and −1 will be returned as sentinel value (instead of error count).

We include a definition for *arrayMapInPlace* which implements the behaviour specified for (a) and (b) using the new function *arrayMap*.

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#define SIZE 10

int arrayMap(double arg[], double result[], int size, double (*f)(double x))
{
  int i, errCount = 0;
  char message[100];  /* need to allocate space for error message */

  for (i = 0; i < size; i++ ) {
    errno = 0;       /* functions like sqrt() don't clear it, only set it — we have to clear it. */
    result[i] = f(arg[i]);
    if (errno) { /* set by this call to (*f) */
      errCount++;
      sprintf(message, "arrayMap: error at index %d", i); /* prepare message */
      perror(message);
    }
  }
  return errCount;
}

int arrayMapInPlace(double ar[], int size, double (*f)(double x))
{
  return arrayMap(ar, ar, size, (*f));
}

int arrayMapNew(double arg[], double ** result, int size, double (*f)(double x))
{
  *result = malloc(size * sizeof(double));
  if ( *result ≠ NULL )
    return  arrayMap(arg, *result, size, (*f));
  else return −1; /* sentinel for unsuccessful result allocation */
}
```

```
void arrayInit(double ar[], int size) {
  int i;
  for (i = 0; i < size; i++)
     ar[i] = 10.0 * sin((double)i);  /* produces some numbers, including negative ones … */
}

void arrayPrint(double ar[], int size) {
  int i;
  for (i = 0; i < size; i++) printf("%3d: %20f\n", i, ar[i]);
  printf("\n");
}

double square(double x) { return x * x; }
double cube(double x) { return x * x * x; }

int main () {
  int i;
  double ar[3][SIZE];
  arrayInit(ar[0],SIZE);
  for (i=0; i<3; i++) arrayPrint(ar[i],SIZE);
  printf("First mapping: %d errors\n", arrayMap(ar[0], ar[1], SIZE, square));
  printf("Second mapping: %d errors\n", arrayMap(ar[0], ar[2], SIZE, sqrt));
  for (i=0; i<3; i++) arrayPrint(ar[i],SIZE);

  double * result;
  printf("First mapping: %d errors\n", arrayMapNew(ar[0], &result, SIZE, square));
  arrayPrint(result, SIZE);
  printf("Second mapping: %d errors\n", arrayMapNew(ar[0], &result, SIZE, sqrt));
  arrayPrint(result, SIZE);
  return 0;
}
```