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

SFWR ENG 2S03
Exercise Sheet 4
Solution Hints

SFWR ENG 2S03 — Principles of Programming

4 October 2006

Exercise 4.1 — Fibonacci Instrumentation

Modify the program fib1.c shown in the lecture so that your modified program produces the following output:

```
fib(5) start
      fib(4) start
            fib(3) start
                  fib(2) start
                        fib(1) start
                        fib(1) = 1
                        fib(0) start
                        fib(0) = 0
                  fib(2) = 1
                  fib(1) start
                  fib(1) = 1
            fib(3) = 2
            fib(2) start
                  fib(1) start
                  fib(1) = 1
                  fib(0) start
                  fib(0) = 0
            fib(2) = 1
      fib(4) = 3
      fib(3) start
            fib(2) start
                  fib(1) start
                  fib(1) = 1
                  fib(0) start
                  fib(0) = 0
            fib(2) = 1
            fib(1) start
            fib(1) = 1
      fib(3) = 2
fib(5) = 5
5  5
```

Solution Hints

```
#include <stdio.h>          /* fib1instr.c */
#include <stdlib.h>
void space(int k);

int fib(int indent, int n) {
 int result;
 space(indent); printf("fib(%d) start\n", n);
 if ( n == 0 || n == 1)
   result = n;
 else
   { int f1, f2, newindent = indent + 6;
     f1 = fib( newindent, n − 1);
     f2 = fib( newindent, n − 2 );
     result = f1 + f2;
   }
 space(indent); printf("fib(%d) = %d\n", n, result);
 return result;
```

```
}

int main(int argc, char * argv[]) {
  int s = 0, i = atoi(argv[1]);
  s = fib(0, i);
  printf("%d  %d\n", i, s);
  return 0;
}

void space(int k) {
  int i;
  for ( i=0; i<k; i++ ) printf(" ");
}
```

Exercise 4.2 — Simulation of C Program Execution (30% of Midterm 3, 2003)

Simulate execution of the following **correct ANSI C** program:

- Show all calls to the function *f* and their arguments and local variables
- Document intermediate states of the array *q* and indicate where changes are produced
- Show which output is produced, and when

```
#include <stdio.h>
#define SIZE 2
char q[SIZE+2] = "ae";

void f(int m);   // forward declaration

int main() {
  f(0);
  return 0;
}
void f(int m) {
  char h;
  printf("f(%d) <-- %s\n", m, q);
  if (m >= SIZE) return;
  h = q[m];
  q[m] = q[m+1];
  f(m+1);
  q[m+1] = h+1;
  printf("f(%d) --> %s\n", m, q);
}
```

Solution Hints

Output:

```
f(0) <--  ae
f(1) <--  ee
f(2) <--  e
f(1) -->  e
f(0) -->  ebf
```

Printing also the numerical values of the four array elements:

```
f(0) <--  97 101   0   0 --- ae
f(1) <-- 101 101   0   0 --- ee
f(2) <-- 101   0   0   0 --- e
f(1) --> 101   0 102   0 --- e
f(0) --> 101  98 102   0 --- ebf
```

Exercise 4.3 — Histograms (75% of Midterm 1, 2005)

Assume a sensor that produces int-valued readings in the range from 0 to *MAX_READING*.

Throughout this question, we will deal with arrays

long int *readings*[*MAX_READING* + 1]

that contain information about the sensor readings in a certain time interval in the following way:

> For $k \in \{0, \ldots, MAX\_READING\}$, the array element *readings*[*k*] contains the **number of times** the sensor reading produced value *k*.

Note: The solutions of the items are **independent of each other.**

Solution Hints

```
#include <stdio.h>
#include <unistd.h>

#define MAX_READING 7
#define SIZE (MAX_READING + 1)

int getSensorReading();

// Just for testing:
// a pseudo-random number generator without consideration to quality
//
int getSensorReading() {
  static int seed = 1234567;
  seed = 456789 * seed + 1001;
  int m = seed % SIZE;
  return m < 0 ? -m : m;
}
```

(a) Assume that the function

int *getSensorReading*();

(which you do not have to implement) obtains an individual reading from the sensor in question.

Design and implement the function

void *collect*(long int *readings*[], long int *number_of_samples*);

which collects *number_of_samples* sensor readings into the array *readings* such that after the call, *readings*[*k*] contains the **number of times** the sensor reading produced value *k* during this call to *collect*.

Implement *collect* in such a way that it waits 0.2 milliseconds between readings; for these delays, use the following library function:

> ```
> #include <unistd.h>
> void usleep(unsigned long usec);
> ```
> The *usleep*() function suspends execution of the calling process for (at least) *usec* microseconds.

Solution Hints

Design:

- "during this call" $\Rightarrow$ initialisation neccessary.
- After that: repeat *number_of_samples* time:
  - obtain sensor reading *reading*
  - increment *readings*[*reading*]
  - wait 0.2 milliseconds
- (Last wait was not demanded, but also not forbidden…)

```
void collect(long int readings[], long int number_of_samples) {
  long int i;
  int reading;
  for ( i = 0; i ≤ MAX_READING; i++) { // initialisation necessary!
    readings[i] = 0;
  }
  for ( i = 0; i < number_of_samples; i++) {
    reading = getSensorReading();
    readings[reading]++;
    usleep(200);
  }
}
```

(b) Assume that the sensor vendor provided the function *getSensorReading*() as a library function without providing source code for it.

What do you have to do to make programs that use *getSensorReading*() compile and execute properly? Explain!

Solution Hints

- Instruct the preprocessor to find the header file containing the prototype for *getSensorReading*() (or, less recommended, include the prototype in your file).

  This is necessary to make the function **known** to the compiler, so that the compiler can properly set up argument and result passing in calls to to the function.

- Instruct the linker which library to link in (and where to find it).

  Only this makes the **actual implementation** of the function a part of your program — otherwise no relation between the name *getSensorReading* and the machine code the vendor shipped as its implementation is established.

(c) **Design and implement** the function

double *mean*(long int *readings*[])

to calculate ***with minimal loss of precision*** the mean of all sensor readings collected in the array *readings*.

Solution Hints

Design:

- "with minimal loss of precision" $\Rightarrow$ add into long long int, be careful with division
- Adding: For each array element:

  - the index is the reading value
  - the contents is the number of readings it represents: accumulate in *count*
  - index multiplied with contents is the contribution of these readings: accumulate in *sum*
- Integer division is safe for integral part of average
- Before dividing remainder by *count*, need to convert to double

```
double mean(long int readings[]) {
  double sum = 0, count = 0;            // non-portable alternativs (GNU C): long long int
  int i;
  for ( i = 0; i < SIZE; i++) {
    count += readings[i];
    sum   += (double)(readings[i]) * i;       // cast avoids overflow
  }
  return sum / count;                  // division at type double!
}
```

(d) **Design and implement** the function

void *display*(long int *readings*[], long int *step*, int *height*)

to print a histogram representing the contents of *readings* to the screen. The histogram is truncated (or padded) to height *height*.

```



  ^
^**
***  ^
***  *
***^ *
****^*
****** ^
```

In this histogram, each element of *readings* is turned into one column; each '*' character represents *step* sensor readings, and on the top of a column, a '^' character represents less than *step* sensor readings (but at least one).

The **example** histogram to the left should be produced e.g. by calling *display*(*readings*, 10, 10) with *MAX_READING* = 7 and *readings* containing the values 55, 60, 69, 23, 17, 45, 0, 5.

Solution Hints

Design:

- Idea: same as for zig-zag: at each screen position, find out what to print: space, '*', or '^'
- For each row, the bounds for *readings*[*j*] to produce one of these three characters are the same: pre-calculate.

```
void display(long int readings[], long int step, int height) {
  long int i;
  int j;
  for ( i = height−1; i ≥ 0; i−−) {        // row index
    long int hat = step * i, stars = hat + step;
    for ( j = 0; j < SIZE; j ++) {
      printf("%c",
        readings[j] ≥ stars ? '*' :
        readings[j] >  hat  ? '^' : ' ');
    }
    printf("\n");
  }
}
```

Solution Hints

Main function for testing:

```
int main() {
  long int readings[SIZE] = {55, 60, 69, 23, 17, 45, 0, 5};

//  collect(readings, 100);
  display(readings, 10, 10);
  printf("\n");
  printf("%f\n", mean(readings));
  printf("\n");
  int j;
  for ( j = 0; j < SIZE; j ++) printf("%2d %5ld\n", j, readings[j]);
  return 0;
}
```