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

SFWR ENG 2S03
Exercise Sheet 3
Solution Hints

SFWR ENG 2S03 — Principles of Programming

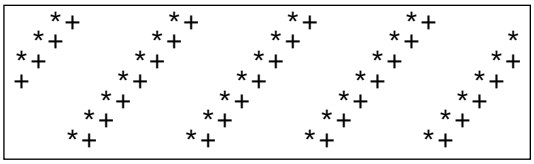
Exercise 3.1 — ASCII Art — Ribbons (60% of Midterm 2, 2003)

Throughout this question, the second dimension of the two-dimensional arrays will be some fixed WIDTH; in the examples this is 30.

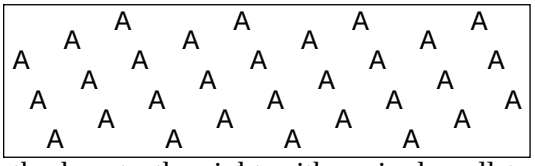
(a) Implement a C function *printCharArray* that prints the contents of a two-dimensional character array to the screen, each row on a separate line.

(b) Implement a C function *putRibbon* that, given a two-dimensional character array, a start height *h* and a character *c*, will place a "ribbon" of *c* values into the array that starts at height *h* and then wind **upwards** diagonally around the array.

Below, the first box contains the result of putting a ribbon of asterisks from start height 2 into a 7×30 array filled with space characters; the second box contains the result of additionally inserting a ribbon of plus characters.

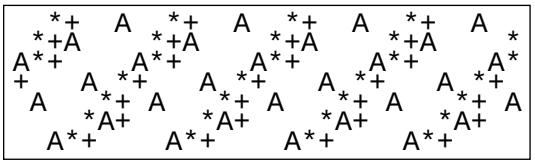


(c) Implement a C function *putSlantedRibbon* that in addition to the arguments of *putRibbon* also accepts an integral *slant* value that indicates the steepness of the ribbon's slant as it winds around the array. This allows one to produce the contents of the box to the right with a single call to *putSlantedRibbon* with a 7×30 array filled with space characters.



As an additional feature, this function **must not override non-space characters** in the array. Use an auxiliary function *squeeze* to squeeze a character into its target position, pushing right all non-space content encountered at the target position and at consecutive positions — the first space character encountered will be consumed.

The same call as for the previous box, when applied after the second box of (b), produces the box to the right — observe how the "A" characters sometimes push only a "+" to the right, sometimes the combination "*+"; at the end of the third line, a "+" has been "pushed off the board".



(d) Write a *main* program that uses the above (**and other**) functions to produce as screen output the contents of the four example boxes above **in the same sequence as above**, using a **single** array of size 7×30.

Do not forget design and documentation, in particular interface documentation for functions!

Solution Hints

```
#include <stdio.h>
#define HEIGHT 7
#define WIDTH 30
```

Only the first dimension can be free: *printCharArray*(*ar*, *h*) prints rows 0 to *h*–1 of character array *ar* to the screen, each row on a line:

```
void printCharArray(char ar[][WIDTH], int height)
{
  int i,j;
  for ( i = 0 ; i < height ; i++ ) {
    for ( j = 0 ; j < WIDTH ; j++ )
      printf ( "%c", ar[i][j] );
    printf( "\n" );
  }
}
```

A remainder function that always return a non-negative remainder:

```
int rem(int i, int j)
{
  int m = i % j;
  if (m < 0) return m + j;
  else       return m;
}
```

We use *rem* to take care of the wrap-around and of possibly negative *startheight*.

The arguments of *putRibbon* are the array (passed by reference), the height of the array, the start height of the ribbon, and the ribbob mark character.

```
void putRibbon(char ar[][WIDTH], int height, int startHeight, char c)
{
  int j;
  for ( j = 0 ; j < WIDTH ; j++ )
    ar[ rem (startHeight – j, height) ][ j ] = c;
}
```

The following function squeezes character *c* into position *k* of **one-dimensional** character array *row* of width *width*, pushing right all non-blank content encountered at position *k* and consecutive positions.

```
void squeeze(char row[], int width, int k, char c)
{
  char tmp;
  while ( k < width  &&  row[k] ≠ ' ' ) {
    tmp = row[k];                    /* swap c and row[k] */
    row[k] = c;
    c = tmp;
```

```
    k++;                        /* increment k */
  }
  if ( k < width )              /* found a blank --- insert c */
    row[k] = c;
}
```

While in *putRibbon*, we calculated the first index directly, her we keep it in a local variable — both ways can be used in both functions.

The argument list is that of *putRibbon* with *slant* added as new second-last argument.

```
void putSlantedRibbon(char ar[][WIDTH], int height,
                      int startHeight, int slant, char c)
{
  int j;
  for ( j = 0 ; j < WIDTH ; j++ ) {
    startHeight = rem (startHeight, height);
    squeeze( ar[ startHeight ], WIDTH, j, c );
    startHeight += slant;
  }
}
```

Initialisation will be needed below; we supply the initialisation value as *c*:

```
void initCharArray(char ar[][WIDTH], int height, char c)
{
  int i,j;
  for ( i = 0 ; i < height ; i++ )
    for ( j = 0 ; j < WIDTH ; j++ )
      ar[i][j] = c;
}
```

It is important to initialise the array with spaces (which are not zero-values), and to re-initialise it for boxes 3 and 4.

```
int main()
{
  char ar[ HEIGHT ][ WIDTH ];

  initCharArray( ar, HEIGHT, ' ');
  putRibbon( ar, HEIGHT, 2, '*');          /* asterisks */
  printCharArray( ar, HEIGHT);             /* Box 1 */

  putRibbon( ar, HEIGHT, 3, '+');          /* plusses */
  printCharArray( ar, HEIGHT);             /* Box 2 */

  initCharArray( ar, HEIGHT, ' ');         /* Clear for box 3 */
  putSlantedRibbon( ar, HEIGHT, 2, 2, 'A'); /* As */
  printCharArray( ar, HEIGHT);             /* Box 3 */
```

```
  initCharArray( ar, HEIGHT, ' ');          /* Clear for box 4 (As have to be last) */
  putRibbon( ar, HEIGHT, 2, '*');           /* asterisks */
  putRibbon( ar, HEIGHT, 3, '+');           /* plusses */
  putSlantedRibbon( ar, HEIGHT, 2, 2, 'A'); /* As (squeezing) */
  printCharArray( ar, HEIGHT);              /* Box 4 */

  return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  int i = atoi(argv[1]), j = atoi(argv[2]);
  printf("%4d %% %4d = %5d\n", i, j, i % j);
  return 0;
}
```

Exercise 3.2 — Simulation of C Program Execution (40% of Midterm 2, 2003)

Simulate execution of the following **correct ANSI C** program; show the intermeidiate steps and show which output is produced:

```
1   #include <stdio.h>
2   #define SIZE 5
3   int q[SIZE];
4
5   void printq() {
6     int i;
7     printf("[ ");
8     for ( i=0; i < SIZE; i++)
9       printf("%4d ", q[i]);
10    printf("]\n");
11  }
12
13  int s(int i, int j) {
14    int k = q[i];
15    q[i] = j;
16    return k;
17  }
18  int f(int m, int n) {
19    int h, r, mm = m + 1, nn = n – 1;
20    printf("f(%d,%d) --- ", m, n);
21    printq();
22    if (m >= n) return q[m];
23    h = s(mm, q[m]);
24    r = f(mm,nn);
25    q[m] = s(nn, q[n]);
26    q[n] = h;
27    return r;
28  }
29  int main() {
30    int i;
31    for ( i=0; i < SIZE; i++)
32      q[i] = 11* (i + 1);
33    printf("%d\n", f(0,SIZE–1));
34    printq();
35    return 0; }
```

Solution Hints

```
f(0,4) --- [   11   22   33   44   55 ]
f(1,3) --- [   11   11   33   44   55 ]
f(2,2) --- [   11   11   11   44   55 ]
11
[   33   11   44   55   22 ]
```

Exercise 3.3 — Compilation Phases (8% of Midterm 1, 2004)

Name the phases of compilation — **give a short description, too** — and the result of each phase.

Solution Hints

- **Preprocessing:** including includ files, replacing preprocessor macro calls, (eliminating comments): ***preprocessed source***
- **Compilation:** translating higher-level source language into lower-level target language: ***assembly code***
- **Assembly:** transliterating mnemonic assembly instrctions into machine code: ***object file***, machine code objects
- **Linking:** Integrating object files with libraries and run-time environment, resolving symbolic references: ***executable***

Exercise 3.4 — Find Errors (16% of Midterm 1, 2004)

In each of the following programs or program segments,

- **Find and describe the error**. If the error can be corrected, explain how.
- Mark any unclear or unintuitive use of C features, **explain the problem**, and propose improvements.

(a)
```
int p=1, q=2.3;
p = q = 7;
printf( "q = %s\n", q );
```

(b)
```
int funny( int n, int k ) {
  return n ? k * funny (n–1) : 1;
}
```

(c)
```
int strange(int q, int r) {
  int m;
  if (q < r)
    m = r;       /* set m          */
  else           /* to minimum */
    m = q;       /* of q and r     */
  return m * m + q * r;
}
```

(d)
```
#include <stdio.h>
int main() {
  int i, count;
  printf("How often?\n");
  scanf("%d", count);
  for( i=1; i ≤ count; i++) {
    printf("Hello!\n");
    main();
  }
  return 0;
}
```

Solution Hints

(a) Error: Format specification "%s" must be replaced by "%d" since *q* is of type int (otherwise probable segmentation fault).

Problems: Dubious initialisation of int variable *q* with floating-point literal; dubious re-initial-

isation of *p* and *q*.

(b) Error: Recursive call to *funny* has only one argument instead of two.

Potential problem: C syntax for conditional expressions is not very readable.

(c) Error: In the *scanf* call, the second argument must to be prefixed with & (otherwise probable segmentation fault).

Problem: Calling *main* recursively is unusual — better do this in a separate function.

(d) Error: Comment disagrees with implementation. Find out which, if any, is correct …
