CS1010E Programming Methodology
Semester 1 2016/2017

Week of 12 September – 16 September 2016
Tutorial 4 Suggested Answers
Nested Control Structures

1. As mentioned during the lecture, printing patterns is a good way to test your understanding of nested loop constructs.

   (a) Write a program that takes as input an odd integer n and prints a diamond. For example, if `n` is 5, the following is printed

```
  *
 ***
*****
 ***
  *
```

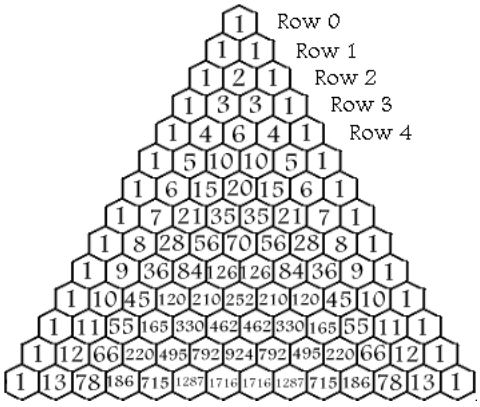
```
#include <stdio.h>

int main(void) {
   int n, i, j;

   printf("Enter n: ");
   scanf("%d", &n);

   n = n / 2 + 1;

   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n - i; j++) {
         printf(" ");
      }
      for (j = 1; j <= 2 * i - 1; j++) {
         printf("*");
      }
      printf("\n");
   }

   i = i - 2;
   while (i >= 1) {
      for (j = 1; j <= n - i; j++) {
         printf(" ");
      }
      for (j = 1; j <= 2 * i - 1; j++) {
         printf("*");
      }
      printf("\n");
      i--;
   }

   return 0;
}
```

(b) Write a program that takes as input an integer n and prints a hollow up-pointing triangle. For example, if `n` is 5, the following is printed

```
    *
   * *
  *   *
 *     *
*********
```

```
#include <stdio.h>

int main(void) {
   int n, i, j, blank1, blank2;

   printf("Enter n: ");
   scanf("%d", &n);

   /* print top line */
   for (i = n - 1; i > 0; i--)
      printf(" ");
   printf("*\n");

   /* print middle part */
   blank1 = n - 2;
   blank2 = 1;
   for (j = 2; j < n; j++) {
      for (i = 1; i <= blank1; i++)
         printf(" ");

      printf("*");

      for (i = 1; i <= blank2; i++)
         printf(" ");

      printf("*\n");

      blank1 = blank1 - 1;
      blank2 = blank2 + 2;
   }

   /* print bottom line if n > 1 */
   if (n > 1) {
      for (i = 1; i <= 2 * n - 1; i++)
         printf("*");
      printf("\n");
   }

   return 0;
}
```

2. Pascal's Triangle was originally developed by the ancient Chinese, but Blaise Pascal was the first person to discover the importance of all of the patterns it contained.



The tip of the Pascal's Triangle is the zeroth row (i.e. $n = 0$), the next row is $n = 1$, and so on. Each row contains $n + 1$ elements, with each element numbered starting with $r = 0$ on the left to $r = n$ on the right. So $n = 5$, $r = 3$ will denote the element 10. As it turns out, each element can be computed combinatorially as

$$\begin{aligned}
\binom{n}{r} &= \frac{n!}{r!(n-r)!} \\
&= \frac{1 \times 2 \times \cdots \times (n-1) \times n}{(1 \times 2 \times \ldots (r-1) \times r) \times (n-r)!} \\
&= \frac{(r+1) \times (r+2) \times \cdots \times (n-1) \times n}{(n-r)!} \\
&= \frac{\prod_{i=r+1}^{n} i}{(n-r)!}
\end{aligned}$$

Follow the steps below to develop a program that reads an input integer n, and outputs the corresponding Pascal's Triangle. Below is a Pascal's Triangle of $n = 10$.

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
```

(a) Write a program that takes as input the integer value n and outputs the factorial of n, denoted $n! = 1 \times 2 \times \ldots n$.

(b) Modify the program in part 2a such that it now takes as input the integer values n and r ($\leq n$) and outputs the factorial of $(n-r)!$.

(c) Extend the program in part 2b so that in addition to $(n-r)!$, it now outputs the product

$$\prod_{i=r+1}^{n} i = (r+1) \times (r+2) \times \cdots \times n$$

(d) Using the computations in parts 2b and 2c, output the term given by

$$\frac{\prod_{i=r+1}^{n} i}{(n-r)!}$$

Note that the the empty product $\prod_{\varnothing} = 1$.

(e) Instead of a single term as in part 2d, output the entire n-th row of the Pascal's Triangle. *Rather than reading the input for r, loop the values of r from 0 to n.*

(f) Instead of a single row as in part 2e, output the entire Pascal's Triangle for a given integer input n. What happens when the input n is more than 12?

```
#include <stdio.h>

int main(void) {
   int N, n, r, i, den, num, term;

   scanf("%d", &N);

   for (n = 0; n <= N; n++) {
      for (r = 0; r <= n; r++) {

         num = 1;
         for (i = r + 1; i <= n; i++) {
            num = num * i;
         }

         den = 1;
         for (i = 2; i <= n - r; i++) {
            den = den * i;
         }

         term = num / den;
         printf("%d ", term);
      }

      printf("\n");
   }

   return 0;
}
```

3. Given a set $\mathbf{T}$ consisting of n values t_i, the normalized value of t_i is defined as

$$\bar{t}_i = \frac{t_i - \min_{\mathbf{T}}}{\max_{\mathbf{T}} - \min_{\mathbf{T}}}$$

where $\min_{\mathbf{T}}$ and $\max_{\mathbf{T}}$ represent the minimum and maximum values among all n values in $\mathbf{T}$. For example, the set of values $\mathbf{T} = \{1, 2, 3, 4, 5\}$ upon normalizing will become $\bar{\mathbf{T}} = \{0, 0.25, 0.5, 0.75, 1\}$ since $\max_{\mathbf{T}} = 5$ and $\min_{\mathbf{T}} = 1$. Moreover, the normalized mean (or normalized average) is defined as

$$\bar{t}_{mean} = \frac{\sum_i \bar{t}_i}{n}$$

With the set of normalized values $\bar{\mathbf{T}}$ generated, the normalized mean can be easily computed to be 0.5. Notice that to compute each $\bar{t}_i$ requires the value of t_i to be read and $\min_{\mathbf{T}}$ and $\max_{\mathbf{T}}$ to be known beforehand; however, they will not be known until all values of t_i are read. *Hint: you will need to re-express the normalized mean.*

In the process of computing the normalized mean

$$\bar{t}_{mean} = \frac{\sum_i \bar{t}_i}{n}$$

we require $min_{\mathbf{T}}$ and $max_{\mathbf{T}}$ to be known beforehand. However, these values will not be known until all t_i are read. So, the normalized mean needs to be re-expressed as

$$\begin{aligned}
\bar{t}_{mean} &= \frac{\sum_i \bar{t}_i}{n} \\
&= \frac{\sum_i \frac{t_i - \min_{\mathbf{T}}}{\max_{\mathbf{T}} - \min_{\mathbf{T}}}}{n} \\
&= \frac{\frac{\sum_i t_i - n\min_{\mathbf{T}}}{\max_{\mathbf{T}} - \min_{\mathbf{T}}}}{n} \\
&= \frac{\frac{\sum_i t_i}{n} - \min_{\mathbf{T}}}{\max_{\mathbf{T}} - \min_{\mathbf{T}}}
\end{aligned}$$

(a) Write a program that reads as input an integer n denoting the number of data values, followed by each of the n floating-point data values. The program then computes the corresponding normalized mean. Sample runs are shown below. User input is underlined.

```
10
10 9 8 7 6 5 4 3 2 1
Normalized mean: 0.500000
```

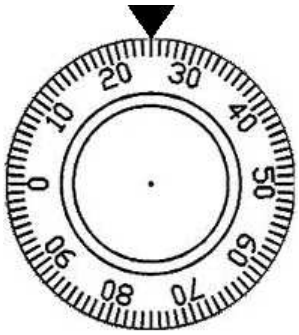
```
6
0.023438 0.070313 -0.039063 -0.039063 0.046875 0.101563
Normalized mean: 0.472223
```

(b) Extend the program in part 3a so as to allow the user to repeatedly input sets of values. For each set of values, the normalized mean is output. The program stops when the user inputs the value of $n \leq 0$. A sample run is given below. User input is underlined.

```
10
10 9 8 7 6 5 4 3 2 1
Normalized mean: 0.500000
6
0.023438 0.070313 -0.039063 -0.039063 0.046875 0.101563
Normalized mean: 0.472223
0
```

```
#include <stdio.h>

int main(void)
{
   int n, i;
   double data, sum, min, max, norm;

   scanf("%d", &n);
   while (n > 0) {

      scanf("%lf", &data);
      max = data;
      min = data;
      sum = data;

      for (i = 1; i < n; i++) {
         scanf("%lf", &data);

         if (data < min) {
            min = data;
         } else {
            if (data > max) {
               max = data;
            }
         }
         sum = sum + data;
      }

      norm = ((sum / n) - min) / (max - min);
      printf("Normalized mean: %f\n", norm);

      scanf("%d", &n);
   }

   return 0;
}
```

4. A safe is typically used to secure valuable objects against theft or damage. Opening a safe requires a dial to be spun in clockwise and anti-clockwise revolutions according to a fixed combination of numbers. The diagram below shows a dial with one hundred graduations ranging from 0 to 99, and is currently set at position 25. Assume that the number of graduations of the dial is fixed at 100.



There are two ways to spin the dial.

- **Anti-spin** — Suppose the dial is spun anti-clockwise starting from position 25 for 30 graduations. The resulting position of the dial is 55. Spinning the dial anti-clockwise starting from position 25 for 80 graduations will result in the dial ending at position 5.
- **Spin** — Suppose the dial is spun clockwise starting from position 25 for 20 graduations. The resulting position of the dial is 5. Spinning the dial clockwise starting from position 25 for 40 graduations will result in the dial ending at position 85.

(a) Develop your program incrementally by implementing anti-spin and spin functionalities separately. Test each functionality by devising suitable test cases.

(b) Have the program read in a sequence of positive integers that represent the number of graduations, terminated by zero. By alternating anti-spins and spins, determine the final position of the dial. You may make the following assumptions:

- The dial is set to position 0 at the start.
- Begin with an anti-spin.
- The value of `num` may be greater than 100, hence requiring more than one complete turn of the dial.

A sample run of the program is given below. User input is underlined.

```
123 234 345 0
The dial is at position 34.
```

Possible test cases:

- `0`: no turn.
- `1 0`: one graduation.
- `99 0`: just before one full anti-spin.
- `100 0`: exactly one full anti-spin;

- `101 0`: one click more than full anti-spin;
- `100 1 0`: one full anti-spin and one graduation spin;
- `100 99 0`: one full anti-spin and just before one full spin;
- `100 100 0`: one full anti-spin and one full spin;
- `100 101 0`: one full anti-spin and jsut after one full spin;
- *other combinations you deem fit*

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

int main(void)
{
   int num, pos = 0;
   bool antiSpin = true;

   scanf("%d", &num);
   while (num > 0) {
      if (antiSpin) {
         while (num > 0) {
            pos = pos + 1;
            if (pos == 100) {
               pos = 0;
            }
            num = num - 1;
         }
         /* alternatively, pos= = (pos+num)%100; */
      } else {
         while (num > 0) {
            pos = pos - 1;
            if (pos == -1) {
               pos = 99;
            }
            num = num - 1;
         }
         /* alternatively, pos = (pos + (100-(num%100)))%100; */
      }

      antiSpin = !antiSpin;
      scanf("%d", &num);
   }

   printf("The dial is at position %d.\n", pos);

   return 0;
}
```