CS1010E Programming Methodology
Semester 1 2016/2017

Week of 5 September – 9 September 2016
Tutorial 3 Suggested Answers
Control Structures – Selection and Repetition

1. Given the following program fragment.

```
int i=1;

while (i > 0) {
   i = i + 1;
}

printf("%d\n", i);
```

   (a) What do you think will happen?

   (b) Run the program, observe what happens and make your own deductions.
   The program terminates with the output -2147483648 after a short while. Due to the finite range of integer values from -2147483648 to 2147483647, incrementing beyond the upper bound causes an overflow error, resulting in a leap to the lower bound. Since this value is negative, the loop terminates.

2. Given the following program fragment.

```
int i, n, count2=0, count3=0, count5=0;

scanf("%d", &n);

for (i = 0; i <= n; i=i+1) {
   if (i%5 == 0) {
      count5 = count5 + 1;
      if (i%3 == 0) {
         count3 = count3 + 1;
      }
   } else {
      if (i%2 == 0) {
         count2 = count2 + 1;
      }
   }
}

printf("%d %d %d\n", count5, count3, count2);
```

(a) Perform a timeline trace of the `count2/3/5` variables from $i = 0$ to $i = 30$.

count2	0	:	:	:	:	:	1	:	:	2	:	:	:	:	3	:	:	4	:
count3	0	:	:	1	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
count5	0	:	1	:	:	:	:	:	:	:	:	2	:	:	:	:	:	:	:
i		0	:	:	1	2	:	3	4	:	5	:	:	6	:	7	8	:	9
count2	0	:	:	:	:	:	5	:	:	6	:	:	:	:	7	:	:	8	:
count3	0	:	:	:	:	:	:	:	:	:	:	:	2	:	:	:	:	:	:
count5	0	:	3	:	:	:	:	:	:	:	:	4	:	:	:	:	:	:	:
i		10	:	:	11	12	:	13	14	:	15	:	:	16	:	17	18	:	19
count2	0	:	:	:	:	:	9	:	:	10	:	:	:	:	11	:	:	12	:
count3	0	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
count5	0	:	5	:	:	:	:	:	:	:	:	6	:	:	:	:	:	:	:
i		20	:	:	**21**	22	:	23	24	:	25	:	:	26	:	27	28	:	29

(b) Using the trace in question 2a, predict the output of the program for input `321`.

(c) Verify your prediction by running the program.
Notice that the pattern of updates to the count variables repeats after every $2 \times 3 \times 5 = 30$ *loops. So for* 321 *loops, the pattern is repeated* 10 *times, with additional updates up to* $i = 21$.

Value of count5 is $6 \times 10 + 5 = 65$
Value of count3 is $2 \times 10 + 2 = 22$
Value of count2 is $12 \times 10 + 8 = 128$

3. Use loop constructs to generate the following number sequences:

   (a) Write a program that reads as input an integer $n(\geq 0)$ and outputs the sum of n terms of the series $1, 2, 3, 4, 5, \ldots$.

```
/*
   This program outputs the sum of first n terms
   of the series 1, 2, 3, 4, ...
*/
#include <stdio.h>

int main(void) {
   int n, i, sum=0;

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

   for (i = 1; i <= n; i++) {
      sum = sum + i;
   }

   printf("%d\n", sum);

   return 0;
}
```

   (b) Rewrite the program to output the sum of n terms of the series $1, 3, 5, 7, 9, \ldots$.

```
/*
   This program outputs the sum of first n terms
   of the series 1, 3, 5, 7, ...
*/
#include <stdio.h>

int main(void) {
   int n, i, sum=0;

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

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

   printf("%d\n", sum);
   return 0;
}
```

(c) Rewrite the program to output the sum of n terms of the series $1, -3, 5, -7, 9, \ldots$.

```
/*
   This program outputs the sum of first n terms
   of the series 1, -3, 5, -7, ...
*/
#include <stdio.h>

int main(void) {
   int n, i, sum=0, sign=1;

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

   for (i = 1; i <= n; i++) {
      sum = sum + sign * (2*i-1);
      sign = sign * -1;
   }

   printf("%d\n", sum);
   return 0;
}
```

(d) Rewrite the program to output the n terms approximation of π given by:

$$\frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \ldots$$

```
/* This program approximates the value of pi using the
   first n terms of the Taylor series expansion.
 */
#include <stdio.h>

int main(void) {
   int n, i, sign=1;
   double approx = 0.0;

   printf("Enter n: ");
   scanf("%d", &n);
   for (i = 1; i <= n; i++) {
      approx = approx + sign * (4.0/(2*i-1));
      sign = sign * -1;
   }
   printf("The approximate value of PI is %f\n", approx);
   return 0;
}
```

(e) Observe that as n increases, the π approximations get progressively closer. Rewrite the program to read as input a floating-point tolerance value δ. The program outputs the i^{th} approximation π_i for which the difference between the two most recent approximations (π_i and π_{i-1}) does not exceed δ. What are the π approximations for $t = 0.001$, $t = 0.0001$ and $t = 0.00001$?

```
/* This program approximates the value of pi using the Taylor series
   expansion.  The approximation stops when the difference between
   consecutive approximations are less than a specified threshold, t.
*/
#include <stdio.h>

int main(void) {
   int n=1, sign=1;
   double t, approx=0.0, prev;

   printf("Enter t: ");
   scanf("%lf", &t);
   do {
      prev = approx;
      approx = approx + sign * (4.0/n);
      n = n + 2;
      sign = sign * -1;
   } while ( ((approx-prev) > t) || ((prev-approx) > t) );

   printf("The approximate value of PI is %f\n", approx);
   return 0;
}
```

Alternatively,

```
#include <stdio.h>

int main(void) {
   int n=1, sign=1;
   double t, approx=0.0, term;

   printf("Enter t: ");
   scanf("%lf", &t);
   do {
      term = sign * (4.0/n);
      approx = approx + term;
      n = n + 2;
      sign = sign * -1;
   } while ((term > t) || (term < -t));

   printf("The approximate value of PI is %f\n", approx);
   return 0;
}
```

4. Write a program that takes in two positive integers and returns the greatest common divisor (*gcd*) of the two integers. For example, the *gcd* of 539 and 84 is 7. Two algorithms for determining the *gcd* is given below.

   (a) Set a variable `gcd` to be the smaller of the two values. If this value of `gcd` completely divides the two numbers, then return this value as the *gcd*. Otherwise, reduce the value of `gcd` by one and repeat the test.

```
#include <stdio.h>

int main(void) {
   int a, b, gcd;

   printf("Enter two positive numbers (> 0) separated by spaces: ");
   scanf("%d %d", &a, &b);

   if (a < b) {
      gcd = a;
   } else {
      gcd = b;
   }

   while ((a%gcd > 0) || (b%gcd > 0)) {
      gcd--;
   }

   printf("The gcd is %d.\n", gcd);
   return 0;
}
```

   (b) We apply the Euclidean algorithm. Let the two values be `a` and `b`. Replace `b` with the result of `a%b`, and `a` with the original value of `b` (before the replacement). Keep doing this until `b` becomes zero; the value of `a` is the *gcd*.

```
#include <stdio.h>

int main(void) {
   int a, b, r;

   printf("Enter two positive numbers (> 0) separated by spaces: ");
   scanf("%d %d", &a, &b);

   while (b > 0) {
      r = a%b;
      a = b;
      b = r;
   }

   printf("The gcd is %d.\n", a);
   return 0;
}
```

5. Every positive integer greater than one can be expressed **uniquely** as a product of primes. For example, the number 10 can be expressed as 2×5 and the number 20 can be expressed as $2 \times 2 \times 5$. This process is sometimes known as *prime factorization*.

   Write a program that reads an integer n (> 1) as user input and outputs all prime factors of n in increasing order. Sample runs of the program are given below. User input is underlined.

```
Enter n (> 1): 2
2
```

```
Enter n (> 1): 10
2 x 5
```

```
Enter n (> 1): 20
2 x 2 x 5
```

```
Enter n (> 1): 1010
2 x 5 x 101
```

```
Enter n (> 1): 223092870
2 x 3 x 5 x 7 x 11 x 13 x 17 x 19 x 23
```

```
/*
   This program performs prime factorization on a
   given integer n (> 1).
*/
#include <stdio.h>

int main(void) {
   int n, d;

   printf("Enter n (> 1): ");
   scanf("%d", &n);

   d = 2;
   while (d < n) {
      if (n%d == 0) {
         printf("%d x ", d);
         n = n / d;
      } else {
         d++;
      }
   }
   printf("%d\n", d);

   return 0;
}
```