**CS1010 Programming Methodology**
**Week 5: Repetition Statements**

> Learn as much as you can while you are young, since life becomes too busy later. ~ *Dana Stewart Scott*

***To students:***

Due to time constraint, not all the questions in this Discussion Sheet are discussed in class. Your DL has the discretion to choose the questions to discuss (or you may request your DL to discuss certain questions). Please do go through those left-over questions after class.

Please be reminded that **the deadline for Lab #2 is next Monday 12nn!**

## I. Revision on Selection Statements and Good Programming Habits

1. The following program works but is badly written. Download it from cs1010 account and improve it by simplifying logic and avoiding re-computation:

   **`cp ~cs1010/discussion/Week5_Q1.c .`**

```
#include <stdio.h>

int main(void)
{
   // declare the first input and second input
   double the_first_input, the_second_input;

   // ask user to enter two values
   printf("Enter two values: ");
   scanf("%lf %lf", &the_first_input, &the_second_input);

   if (the_first_input/the_second_input < 90.2)
   {
      if (the_first_input/the_second_input < 32.2)
         printf("Paper\n");
      else if (the_first_input/the_second_input >= 45.8)
         printf("Ruler\n");
      else
         printf("Pencil\n");
   }
   else
   {
      if (the_first_input/the_second_input >= 100.0)
         printf("Unknown\n");
      else if (the_first_input/the_second_input < 100.0)
         printf("Eraser\n");
      else
         printf("Clip\n");
```

```
   }

   return 0;
}
```

2. Study the following program **Week5_Q2.c**.

   You see a new data type here: the **char** type, which stands for character. Character constants are enclosed in single quotes, for example, 'A', 'w', '+'. The format specifier for character in a `printf` statement is **%c**.

   What is the output if the user enters 12? Replace the `switch` statement with an equivalent `if` statement.

```
// To convert switch statement into if statement.
#include <stdio.h>

int main(void)
{
   int score;
   char grade;

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

   switch (score)
   {
      case 10:
      case 9:
      case 8:  grade = 'A';
               break;
      case 7:
      case 6:  grade = 'B';
               break;
      case 5:  grade = 'C';
               break;
      default: grade = 'F';
               break;
   }

   printf("Grade is %c.\n", grade);

   return 0;
}
```

## IV. Repetition Statements I

3. **Conversion of loop construct.**

You have learned 3 loop constructs: **`for`**, **`while`** and **`do-while`**. For each of the following parts, a particular loop construct is used. Rewrite the loop construct using the other 2 loop constructs (without adding any **`if`** statement), and indicate if there is any limitation on the new codes. You may assume that all variables are of type **`int`**.

(a)

```
int sum = 0;
int i = -5;
do {
    sum += i;
    i += 5;
} while (i < 100);
printf("sum = %d\n", sum);
```

(b)

```
int i = 0;
printf("Enter n: ");
scanf("%d", &n);
while (i < n)
{
    printf("*");
    i += 3;
}
printf("\n");
```

(c) The following **`for`** loop may look strange, but after converting it into the other 2 loop constructs you should understand what it does. Can you describe what the code does in words?

```
printf("Enter n: ");
for (scanf("%d", &n); n < 0; scanf("%d", &n))
    printf("Enter n: ");

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

Suppose the valid range of values for `score` in question 2 should be 0 to 10 inclusive. Would you now be able to write a code to check that the user's input is valid (and keep asking if it is invalid) before going to the `switch` statement?

4. **Manual tracing.**

(a) Spot an error in this program and correct it. Then manually trace the program and write out its output.

```
int i, sum;

for (i = 1; i < 1000; i*=2)
    sum += i;

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

(b) Manually trace the program and write out its output.

```
int a = 10, b = 200;

while ((a*a) < (a+b))
{
    printf("a = %d, b = %d\n", a, b);
    a++;
    b+=10;
}
```

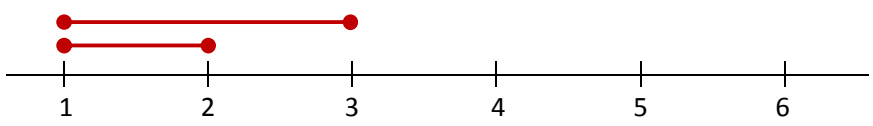
(c) Manually trace the program and write out its output.

```
int x, y, count = 0;

for (x = 1; x <= 6; x++)
    for (y = x + 1; y <= 6; y++)
        count++;

printf("count = %d\n", count);
```

(d) Suppose we have a function **draw_line(int a, int b)** which draws a horizontal line segment from a to b. If you replace the statement **count++;** in part (c) above with the statement **draw_line(x, y);**, how would the line segments be drawn? The diagram below shows the first two line segments drawn (a new line segment is drawn above an old one). Complete the diagram.



(e) Manually trace the program and write out its output.

```
int x, y, count = 0;

for (x = 1; x <= 5; x++)
    for (y = x; y <= 5; y++)
        for (z = x; z <= y; z++)
            count++;

printf("count = %d\n", count);
```

5. Adam and Brusco each wrote the following functions to calculate **Greatest Common Divisor (GCD)**. Study their functions.

Adam's code:

```
// Returns the GCD of a and b
// Pre-cond: a>=0, b>=0 and not both = 0
int adam_gcd(int a, int b)
{
    int divisor;

    if (a == 0)
        return b;
    else if (b == 0)
        return a;

    if (a < b)
        divisor = a;
    else
        divisor = b;

    while (1)
    {
        if ((a%divisor == 0) && (b%divisor == 0))
            return divisor;
        divisor--;
    }
    return 1;
}
```

Brusco's code:

```
// Returns the GCD of a and b
// Pre-cond: a>=0, b>=0 and not both = 0
int brusco_gcd(int a, int b)
{
    int temp, remainder;

    // Swap a with b if a < b
    if (a < b) {
        temp = a; a = b; b = temp;
    }

    while (b != 0) {
        remainder = a % b;
        a = b;
        b = remainder;
    }

    return a;
}
```

(a) Compare the two functions. Which one do you think is better? Have an empirical verification by downloading the following program:

**`cp ~cs1010/discussion/Week5_Q5.c .`**

and test it on this pair of data: **987654321** and **987654322**. What do you observe?

(c) Brusco actually drew the idea from the famous Euclid's algorithm which is shown below.

```
// Returns the GCD of a and b
// Pre-cond: a>=0, b>=0 and not both = 0
int euclid_gcd(int a, int b)
{
    int remainder;

    while (b != 0)
    {
        remainder = a % b;
        a = b;
        b = remainder;
    }
    return a;
}
```

Convince yourself that above algorithm works no matter whether *a* is bigger than *b* or vice versa.