CS1010 Programming Methodology

Week 11: Recursion

> Learning is not attained by chance,
> it must be sought for with ardor
> and attended to with diligence.
> ~ *Abigail Adams*

To students:

- Please be reminded that **lab #5 deadline is this Saturday, 6pm**.

1. **Tracing recursive codes**

(a) [CS1010 AY2010/2011 Semester 1 Exam, Q1.2]
Given the following function, what does **calculate(5)** compute?

```
// Precond: n >= 0
int calculate(int n)
{
    if (n == 0)
        return 0;
    else
        return (2 * n + calculate(n-1));
}
```

(b) Trace the function below manually, and write out the return value of **q(12)**.

```
// Precond: n >= 0
int q(int n)
{
   if (n < 3)
      return n+1;
   else
      return q(n-3) + q(n-1);
}
```

Exploration: Would you be able to write an iterative version of this function? Run both versions on a large input, such as 50. What do you observe?

(c) [CS1010 AY2011/2012 Semester 1 Exam, Q1.5]

What does the following function return?

```
int mystery(int x, int y)
{
   if (x == 0)
      return y;
   else if (x < 0)
      return mystery(++x, --y);
   else
      return mystery(--x, ++y);
}
```

A. It returns the value of y.

B. It returns the value of x – y.

C. It returns the value of x + y.

D. It returns the value of x * y.

E. It will give compile-time error.

2. **Summing digits in an integer.**
Summing digits in a non-negative integer *n* can be easily written using a loop. Is writing a recursive code for it just as easy? Write a recursive function **int sum_digits(int n)** to sum up the digits in *n*.

A sample run is shown below:

```
Enter a non-negative integer: 970517
Sum of its digits = 29
```

Download skeleton **Week11_Q2.c** from cs1010 account

3. **Recursion on array**

Study the program **Week11_Q3.c** below and trace the recursive function **mystery(int [] , int)**.

What is the smaller version of the problem on which the recursive call works? How does the original problem relate to this smaller problem? What does the function compute?

```
#include <stdio.h>
#define SIZE 8

void scan_array(int [], int);
int mystery(int [], int);

int main(void)
{
   int list[SIZE];
   scan_array(list, SIZE);
   printf("Answer = %d\n", mystery(list, SIZE));
   return 0;
}

// Read in values for array arr
void scan_array(int arr[], int size)
{
   int i;

   printf("Enter %d values: ", size);
   for (i=0; i<size; i++)
       scanf("%d", &arr[i]);
}

// Precond: n > 0
int mystery(int arr[], int n)
{
   int m;

   if (n == 1)
       return arr[0];
   else
   {
       m = mystery(arr, n-1);
       return (arr[n-1] > m) ? arr[n-1] : m;
   }
}
```

4. [CS1010 AY2010/2011 Semester 1 Exam, Q4]
Write a recursive function **int largest_digit_pairs(int n)** to determine the largest pair of digits of a positive integer ***n*** starting from the right to the left.

For example, if ***n*** is 5064321, then the pairs are 21, 43, 06 and 5, and hence the answer is 43.

Download skeleton **Week11_Q4.c** from cs1010 account

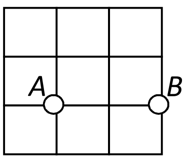
5. **Reversing an Array**
Write a function **void reverse_array(int arr[], int size)** to reverse an integer array using recursion.

For example, if the array contains { 6, 3, 0, 6, 8, 1, 5 }, then the reversed array is { 5, 1, 8, 6, 0, 3, 6 }. You should not use any additional array.

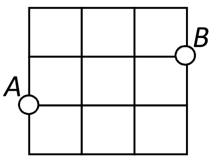
Answer: See **Week11_Q5.c**

Download skeleton **Week11_Q5.c** from cs1010 account

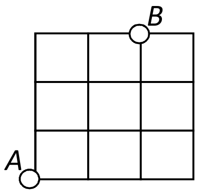
6. **North-East Paths**
In a special town where pedestrians are only allowed to move northwards or eastwards, each of the following examples shows the total number of unique NE-paths, **ne(x, y)**, to get from point *A* to point *B*, where *B* is *x* rows north and *y* columns east of *A*. Assume that *x* and *y* are non-negative integers. By convention, ne(0, 0) = 1.



ne(0, 2) = 1



ne(1, 3) = 4



ne(3, 2) = 10

Write a recursive function **int ne(int x, int y)** to compute the number of unique NE-paths between two points which are separated by *x* rows and *y* columns..

The following are some sample runs.

```
Enter rows and columns apart: 0 2
Number of NE-paths = 1
```

```
Enter rows and columns apart: 1 3
Number of NE-paths = 4
```

```
Enter rows and columns apart: 3 2
Number of NE-paths = 10
```