Homework One

Q1. There is a 5-digit number that satisfies 4*abcde = edcba, that is, when multiplied by 4 yields the same number read backwards. Write a C-program to find this number.

Solution:

```
#include <stdio.h>
#define MIN 10000
#define MAX 24999    // solution has to be <25000
int main(void) {
  int a, b, c, d, e, n;
  for (n = MIN; n <= MAX; n++) {
    a = (n / 10000) % 10;
    b = (n / 1000) % 10;
    c = (n / 100) % 10;
    d = (n / 10) % 10;
    e = n % 10;
    if (4*n == 10000*e + 1000*d + 100*c + 10*b + a) {
      printf("%d\n", n);
    }
  }
  return 0;
}
```

Q2. Write a C program to compute the matrix product of two matrices A and B.

Solution:

```
#include <stdio.h>
#define M 4
#define N 4
#define P 4

// Function matrixProduct  computes a[][]*b[][], and stores the result in c[][]

 void matrixProduct(float a[M][N], float b[N][P], float c[M][P]) {
  int i, j, k;
  for (i = 0; i < M; i++) {
    for (j = 0; j < P; j++) {
          c[i][j] = 0.0;
          for (k = 0; k < N; k++) {
            c[i][j] += a[i][k] * b[k][j];
          }
    }
  }
}
```

```
int main()
{
   float a[M][N] = { {1, 1, 1, 1}, {2, 2, 2, 2},   {3, 3, 3, 3},  {4, 4, 4, 4}};
   float b[N][P] = { {1, 1, 1, 1}, {2, 2, 2, 2},  {3, 3, 3, 3}, {4, 4, 4, 4}};
   float c[M][P]; // To store result
   int i, j;

   matrixProduct(a, b, c);
   printf("Result matrix is \n");
   for (i = 0; i < M; i++)
     {
        for (j = 0; j < P; j++)
          printf("%d ", c[i][j]);
        printf("\n");
    }
   return 0;
}
```

Q3. Write a C program that outputs, in alphabetical order, all the distinct strings that use each of the characters 'c', 'a', 't', 'd', 'o', 'g' exactly once. How many strings does the program generate?

Solution:

```
#include <stdio.h>
int main(void) {
  char catdog[] = { 'a','c','d','g','o','t' };

  int count = 0;
  int i, j, k, l, m, n;
  for (i=0; i<6; i++)
    for (j=0; j<6; j++)
          for (k=0; k<6; k++)
            for (l=0; l<6; l++)
              for (m=0; m<6; m++)
                    for (n=0; n<6; n++)
                      if (i!=j && i!=k && i!=l && i!=m && i!=n &&
                             j!=k && j!=l && j!=m && j!=n &&
                             k!=l && k!=m && k!=n &&
                             l!=m && l!=n && m!=n) {
                              printf("%c%c%c%c%c%c\n", catdog[i], catdog[j],
                                                            catdog[k], catdog[l],
                                                            catdog[m], catdog[n]);
                              count++;
                      }
  printf("%d\n", count);
  return 0;
}
```

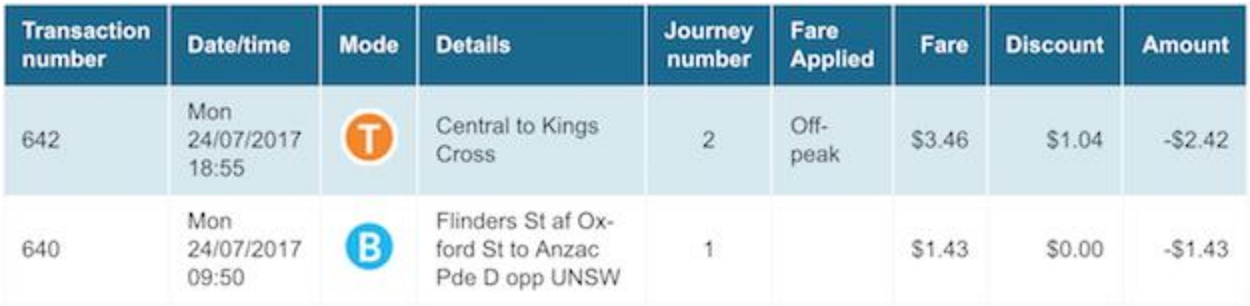
Q4. Write a C function that takes a positive integer n as argument and outputs a series of numbers according to the following process, until 1 is reached:

- If n is even, set n to n/2
- If n is odd, set n to 3*n+1

Solution:

```
void collatz(int n) {  // named after the German mathematician who invented this problem
  printf("%d\n", n);
  while (n != 1) {
    if (n % 2 == 0) {
          n = n / 2;
    } else {
          n = 3*n + 1;
    }
    printf("%d\n", n);
  }
}
```

Q5. Define a data structure to store all information of a single ride with the Opal card. Here are two sample records:



Transaction number	Date/time	Mode	Details	Journey number	Fare Applied	Fare	Discount	Amount
642	Mon 24/07/2017 18:55	T	Central to Kings Cross	2	Off-peak	$3.46	$1.04	-$2.42
640	Mon 24/07/2017 09:50	B	Flinders St af Oxford St to Anzac Pde D opp UNSW	1		$1.43	$0.00	-$1.43

You may assume that individual stops (such as "Anzac Pde D opp UNSW") require no more than 31 characters.

Determine the memory requirements of your data structure, assuming that each integer and floating point number takes 4 bytes.

If you want to store millions of records, how would you improve your data structure?

Solution:

```
typedef struct {
   int day, month, year;
} DateT;

typedef struct {
   int hour, minute;
} TimeT;

typedef struct {
   int   transaction;
   char  weekday[4];       // 3 chars + terminating '\0'
   DateT date;
   TimeT time;
   char  mode;             // 'B', 'F' or 'T'
   char  from[32], to[32];
   int   journey;
   char  faretext[12];
   float fare, discount, amount;
} JourneyT;
```

Memory requirement for one element of type JourneyT: 4 + 4 + 12 + 8 + 1 (+ 3 padding) + 2·32 + 4 + 12 + 3·4 = 124 bytes.

The data structure can be improved in various ways: encode both origin and destination (from and to) using Sydney Transport's unique stop IDs along with a lookup table that links e.g. 203311 to "Anzac Pde Stand D at UNSW"; use a single integer to encode the possible "Fare Applied" entries; avoid storing redundant information like the weekday, which can be derived from the date itself.

Q6. The Fibonacci numbers are defined as follows:

Fib(1) = 1
Fib(2) = 1
Fib(n) = Fib(n-1)+Fib(n-2) for n⩾3

Write a C program fibonacci.c that applies the process described in Q4 to the first 10 Fibonacci numbers. The output of the program should begin with

```
Fib[1] = 1
1
Fib[2] = 1
1
Fib[3] = 2
2
1
Fib[4] = 3
3
10
5
16
8
4
```

```
2
1
```

Solution:

```
#include <stdio.h>
#define MAX 10

void collatz(int n) {  // named after the German mathematician who invented this problem
  printf("%d\n", n);
  while (n != 1) {
    if (n % 2 == 0) {
          n = n / 2;
    } else {
          n = 3*n + 1;
    }
    printf("%d\n", n);
  }
}

int main(void) {
  int fib[MAX] = { 1, 1 };     // initialise the first two numbers
  int i;
  for (i = 2; i < MAX; i++) {  // compute the first 10 Fibonacci numbers
    fib[i] = fib[i-1] + fib[i-2];
  }

  for (i = 0; i < MAX; i++) {  // apply Collatz's process to each number
    printf("Fib[%d] = %d\n", i+1, fib[i]);
    collatz(fib[i]);
  }

  return 0;
}
```

Q7. Write a C function that takes 3 integers as arguments and returns the largest of them. Your C function cannot use any control construct.

Solution:

```
int max(int a, int b, int c) {
  int d = a * (a >= b) + b * (a < b);   // d is max of a and b
  return c * (c >= d) + d * (c < d);   // return max of c and d
}
```

Q8. Write a C program that takes a sequence of integers from the keyboard, sorts them, and displays the sorted sequence on the screen, one integer per line. A non-integer indicates the end of sequence.

Solution:

```
#include <stdio.h>
#define SIZE 250

void insertionSort(int array[], int n) {
  int i;
  for (i = 1; i < n; i++) {
    int element = array[i];           // for this element ...
    int j = i-1;
    while (j >= 0 && array[j] > element) {  // ... work down the ordered list
      array[j+1] = array[j];          // ... moving elements up
      j--;
    }
    array[j+1] = element;             // and insert in correct position
  }
}

int main(void) {
  int numbers[SIZE];
  int i, n=0;
  int done=1,rev;

 while (done) // Initialize the array numbers[] by receiving integers from keyboard
   {
    if (n==SIZE-1)
      break;
     printf("Type in a number \n");
     rev=scanf("%d", &numbers[n]);
     printf("numbers[%d]=%d\n", n, numbers[n]);
     if (rev<=0) // not an integer
        done=0;
     else n++;
   }
  insertionSort(numbers, n);
  for (i = 0; i <n; i++)
    printf("%d\n", numbers[i]);

  return 0;
}
```