COMP9024 18s2 Data Structures and Algorithms

Week 02 Problem Set
Elementary Data and Control Structures in C

1. (Arithmetic)

   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.

   Answer:

```
#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;
}
```

   Solution: 21978

2. (Arrays)

   a. Write a C-function that returns the *inner product* of two *n*-dimensional vectors **a** and **b**, encoded as 1-dimensional arrays of *n* floating point numbers.

      Use the function prototype **float innerProduct(float a[], float b[], int n)**.

      Hint: The inner product of two vectors is calculated as $\sum_{i=1..n} \mathbf{a}_i \cdot \mathbf{b}_i$

   b. Write a C-function to compute **C** as the *matrix product* of matrices **A** and **B**.

      Use the function prototype **void matrixProduct(float a[M][N], float b[N][P], float c[M][P])**.
      You can assume that *M, N, P* are given as symbolic constants, e.g.

```
#define M 3
#define N 4
#define P 4
```

      Hint: The product of an *m*×*n* matrix **A** and an *n*×*p* matrix **B** is the *m*×*p* matrix **C** such that $\mathbf{C}_{ij} = \sum_{k=1..n} \mathbf{A}_{ik} \cdot \mathbf{B}_{kj}$

      for all $i \in \{1..m\}$ and $j \in \{1..p\}$.

   Answer:

```
float innerProduct(float a[], float b[], int n) {
   int i;
   float product = 0.0;

   for (i = 0; i < n; i++)
      product += a[i] * b[i];

   return product;
}

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];
         }
      }
   }
}
```

3. (Characters)

   Write a C-program that outputs, in alphabetical order, all strings that use each of the characters 'c', 'a', 't', 'd', 'o', 'g' exactly once.

   How many strings does your program generate?

   Answer:

   There are 6! = 720 permutations of "catdog".

   A straightforward solution is to use six nested loops and a conditional statement to filter out all strings with duplicate characters. The following program includes a counter to check how many strings have been generated.

```
#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;
}
```

4. (Elementary control structures)

   a. 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, then $n \leftarrow n/2$
      - if *n* is odd, then $n \leftarrow 3*n+1$

   b. The Fibonacci numbers are defined as follows:

      - Fib(1) = 1
      - Fib(2) = 1
      - Fib(*n*) = Fib(*n*-1)+Fib(*n*-2) for $n \geq 3$

      Write a C program `fibonacci.c` that applies the process described in Part a. 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
```

We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to the problem set and week, i.e. prob02 for this week. It expects to find a program named* `fibonacci.c` *in the current directory. You can use dryrun as follows:*

```
prompt$ ~cs9024/bin/dryrun prob02
```

Note: Please ensure that your output follows exactly the format shown above.

Answer:

```
#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;
}
```

5. (Elementary data structures)

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

Transaction number	Date/time	Mode	Details	Journey number	Fare Applied	Fare	Discount	Amount
2013	Mon 30/07/2018 10:16	B	Flinders St af Oxford St to Anzac Pde D opp UNSW	1		$1.46	$0.00	-$1.46
2011	Mon 30/07/2018 10:05	B	Victoria St at Liverpool to Oxford St op Palmer St	1		$2.20	$0.00	-$2.20
2009	Sun 29/07/2018 17:35	T	Bondi Junction to Kings Cross		Day Cap	$3.54	$3.54	$0.00

   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?

Answer:

There are of course many possible ways in which this data can be structured; the following is just one example:

```
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.

6. **Challenge Exercise**

Write a C-function that takes 3 integers as arguments and returns the largest of them. The following restrictions apply:

- You are not permitted to use **if** statements.
- You are not permitted to use loops (e.g. **while**).
- You are not permitted to call any function.
- You are only permitted to use data and control structures introduced in Week 2's lecture.

Answer:

The following makes use of the fact that a true condition has value 1 and a false condition has value 0:

```
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
}
```