Week 1 Problem Set
Elementary Data and Control Structures in C

The purpose of the first problem set is mainly to familiarise yourself with programming in C.

For the remainder of the course it is vital that you understand and are able to solve all of the Exercises 1 – 5 below. In addition, you could gain further, self-guided programming practice by solving small puzzles from this website:

C Puzzles

(*Hint:* Recommended are L11–L20, L31–L40, D11–D20, DD11–DD17. Pointers and random numbers will be introduced later in COMP9024.)

1. (Numbers)

   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.

   Hint: Only use arithmetic operations; do not use any string operations.

   Answer:

```c
#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. (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?

   Hint: Find a simple algorithm to solve this specific problem only. Do not use any functions from the string library.

   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.

```c
#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;
}
```

3. (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 this exercise. It expects to find a program named* `fibonacci.c` *in the current directory. You can use dryrun as follows:*

```
prompt$ 9024 dryrun fibonacci
```

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

Answer:

```c
#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;
}
```

4. (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
3965	Sun 03/05/2020 17:26	T	Parramatta to Toongabbie		Day Cap	$3.61	$3.61	$0.00
3963	Sun 03/05/2020 15:30	T	Kings Cross to Parramatta	2	Day Cap	$5.15	$2.35	-$2.80
3961	Mon 27/04/2020 18:27	B	Taylor Square to Kings Cross Station	1		$0.00	$0.00	$0.00

You may assume that individual stops (such as "Kings Cross Station") 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 "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.

5. (Stack ADO)

a. Modify the Stack ADO from the lecture (`Stack.h` and `Stack.c`) to an implementation of a stack of *integers*. Below is the header file (`IntStack.h`) for your ADO:

`IntStack.h`

```
// Integer Stack ADO header file ... COMP9024 20T2

#define MAXITEMS 10

void StackInit();      // set up empty stack
int  StackIsEmpty();   // check whether stack is empty
void StackPush(int);   // insert int on top of stack
int  StackPop();       // remove int from top of stack
```

Your task is to implement these functions in a program called `IntStack.c`.

b. Complete the test program below (`StackTester.c`) and run it to test your integer stack ADO. The tester

- initialises the stack
- prompts the user to input a number *n*
- checks that *n* is a positive number
- prompts the user to input *n* numbers and push each number onto the stack
- *uses the stack to output the n numbers in reverse order* (needs to be implemented)

StackTester.c

```
// Integer Stack ADO tester ... COMP9024 20T2
#include <stdio.h>
#include <stdlib.h>
#include "IntStack.h"

int main(void) {
   int i, n;
   char str[BUFSIZ];

   StackInit();

   printf("Enter a positive number: ");
   scanf("%s", str);
   if ((n = atoi(str)) > 0) {    // convert to int and test if positive
      for (i = 0; i < n; i++) {
         printf("Enter a number: ");
         scanf("%s", str);
         StackPush(atoi(str));
      }
   }

   /* NEEDS TO BE COMPLETED */

   return 0;
}
```

An example of the program executing could be

```
Enter a positive number: 3
Enter a number: 2019
Enter a number: 12
Enter a number: 25
25
12
2019
```

Answer:

a. IntStack.h

```
// Integer Stack ADO header file

#define MAXITEMS 10

void StackInit();     // set up empty stack
int  StackIsEmpty();  // check whether stack is empty
void StackPush(int);  // insert int on top of stack
int  StackPop();      // remove int from top of stack
```

IntStack.c

```
// Integer Stack ADO implementation
#include "IntStack.h"
#include <assert.h>

typedef struct {
   int item[MAXITEMS];
   int top;
} stackRep;                // defines the Data Structure

static stackRep stackObject;  // defines the Data Object

void StackInit() {         // set up empty stack
   stackObject.top = -1;
}

int StackIsEmpty() {      // check whether stack is empty
   return (stackObject.top < 0);
}

void StackPush(int n) {   // insert int on top of stack
   assert(stackObject.top < MAXITEMS-1);
   stackObject.top++;
   int i = stackObject.top;
   stackObject.item[i] = n;
}

int StackPop() {          // remove int from top of stack
   assert(stackObject.top > -1);
   int i = stackObject.top;
   int n = stackObject.item[i];
   stackObject.top--;
   return n;
}
```

b. StackTester.c

```
// Integer Stack ADO tester ... COMP9024 20T2
#include <stdio.h>
#include <stdlib.h>
#include "IntStack.h"

int main(void) {
   int i, n;
   char str[BUFSIZ];

   StackInit();

   printf("Enter a positive number: ");
   scanf("%s", str);
   if ((n = atoi(str)) > 0) {    // convert to int and test if positive
      for (i = 0; i < n; i++) {
         printf("Enter a number: ");
         scanf("%s", str);
         StackPush(atoi(str));
      }
   }
   while (!StackIsEmpty()) {
      printf("%d\n", StackPop());
   }
   return 0;
}
```

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 1'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
}
```