COMP9024 19T3   Week 1 Problem Set   Data Structures and Algorithms
Elementary Data and Control Structures in C

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

Since most of you are new to C, you will get an extra "Study Week" (week 2) to learn the basics. For the remainder of the course it is vital that you understand and are able to solve all of the Exercises 1–7 below. In addition, you could use the C Study Week for further, self-guided programming practice. The following site, for example, offers many small programming puzzles for you to solve:

C Puzzles

(*Hint:* Relevant are puzzles E, L, 1D and 2D. Pointers and random numbers will be introduced later in COMP9024.)

1. (Numbers)

   There is a 5-digit number that satisfies $4 \cdot abcde = edcba$, 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.

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?

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.

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
3505	Sun 28/07/2019 13:17	T	Kings Cross to Strathfield	9	Travel Reward	$4.48	$2.92	-$1.56
3503	Fri 26/07/2019 18:00	B	Anzac Pde bf Addison St to Taylor Square	8		$3.73	$0.00	-$3.73
3501	Fri 26/07/2019 10:20	B	Flinders St af Oxford St to UNSW	7		$3.73	$0.00	-$3.73

You may assume that individual stops (such as "Anzac Pde bf Addison St") 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?

5. (Stack ADO)

a. Modify the Stack ADO from the lecture (Stack.h and Stack.c) to implement an integer stack ADO (IntStack.h and 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 19T3
#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
```

6. (ADO client)

A stack can be used to convert a positive decimal number *n* to a different numeral system with base *k* according to the following algorithm:

> **while** *n*>0 **do**
>   push *n%k* onto the stack
>   *n* = *n* / *k*
> **end while**

The result can be displayed by printing the numbers as they are popped off the stack. Example (*k*=2):

```
n = 13           --> push 1 (= 13%2)
n = 6  (= 13/2) --> push 0 (= 6%2)
n = 3  (=  6/2) --> push 1 (= 3%2)
n = 1  (=  3/2) --> push 1 (= 1%2)
n = 0  (=  1/2)
Result: 1101
```

Using your stack ADO from Exercise 5, write a C-program that implements this algorithm to convert to base *k*=2 a number given on the command line.

Examples of the program executing could be

```
prompt$ ./binary
Enter a number: 13
1101
prompt$ ./binary
Enter a number: 128
10000000
prompt$ ./binary
Enter a number: 127
1111111
```

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 three programs in the current directory:*

- *IntStack.h – your header file for the integer stack from Exercise 5*
- *IntStack.c – your implementation of the integer stack from Exercise 5*

○ *binary.c*

You can use dryrun as follows:

```
prompt$ 9024 dryrun binary
```

7. (Queue ADO)

Modify your integer stack ADO from Exercise 5 to an integer queue ADO.

Hint: A *queue* is a FIFO data structure (first in, first out). The principal operations are to *enqueue* and to *dequeue* elements. Elements are dequeued in the same order in which they have been enqueued. Below is a sample header file (IntQueue.h) to get you started.

IntQueue.h

```
// Integer Queue ADO header file ... COMP9024 19T3
#define MAXITEMS 10

void QueueInit();        // set up empty queue
int  QueueIsEmpty();     // check whether queue is empty
void QueueEnqueue(int);  // insert int at end of queue
int  QueueDequeue();     // remove int from front of queue
```

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 two files named* IntQueue.c *and* IntQueue.h *in the current directory that provide an implementation of a queue ADO with the four queue functions shown above. You can use dryrun as follows:*

```
prompt$ 9024 dryrun IntQueue
```

8. **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.

Assessment

This first problem set is meant to give you your first practice and will not count towards your mark for the weekly homework.

However, in order to familiarise yourself with the submission and auto-marking process, you can submit your solutions to **Exercise 5 and 6**.

You should submit your files using the following **give** command:

```
prompt$ give cs9024 week1 IntStack.h IntStack.c binary.c IntQueue.h IntQueue.c
```

Alternatively, you can submit through WebCMS3.

Make sure you spell the filenames correctly. You can run **give** and submit multiple times. Only your last submission will be marked.

The deadline for submission is **Monday, 30 September 11:00:00am**.

Auto-marking will be run by the lecturer several days after the submission deadline using different test cases than dryrun does. *Hint*: Do your own testing in addition to running dryrun.