

Week 1 Problem Set

Elementary Data and Control Structures in C

[\[Show with no answers\]](#) [\[Show with all answers\]](#)

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 $4 \cdot abcde = edcba$, 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.

[\[show answer\]](#)

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.

[\[show answer\]](#)

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:

- $\text{Fib}(1) = 1$
- $\text{Fib}(2) = 1$
- $\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2)$ 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.

[\[show answer\]](#)

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		Parramatta to Toongabbie		Day Cap	\$3.61	\$3.61	\$0.00
3963	Sun 03/05/2020 15:30		Kings Cross to Parramatta	2	Day Cap	\$5.15	\$2.35	-\$2.80
3961	Mon 27/04/2020 18:27		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?

[\[show answer\]](#)

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
```

[\[show answer\]](#)

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.

[\[show answer\]](#)