Objective:

- Develop recursive functions in C
- Understand memory for functions (the stack)
- Use pointers

Learning Outcomes:

- The acquisition, application and integration of knowledge
- Critical thinking and problem-solving skills
- Literacy and numeracy skills
- Creativity and aesthetic appreciation

Part 1: Recursive Functions (40 marks)

In this part, for each function that is described, create the non-recursive and recursive solution. All functions are to be placed in the same file and called from main (using any provided test values), responses printed, and tested using assert. The values to test are the provided in each question, all asserts should pass unless otherwise stated. Any and all inputs (scanning) and outputs (printing) is performed by main; the functions should never print. All recursive functions should be efficient and minimize the number of recursive calls inside the function.

Part 1 A: Fibonacci (10 marks)

Develop a function to compute the Fibonacci series. The function receives a positive integer, N, and computes/returns the Nth Fibonacci number. Main should print the result.

The fibonacci series is computed as follows:

Fib(N) = Fib(N-1) + Fib(N-2)

Test values [*hint: program must work when examples are provided*]:

N = 0, 1, 2, 3, 5, 10, 30, 47

Part 1 B: Palindrome (10 marks)

Develop a function to check if a string is a palindrome. The function can receive a string, its size, and a position, and return whether the string is a palindrome. Ignore any characters that are not A-Z or a-z.

Test values:

"abcba", "racecar", "noon"

Part 1 C: Bubble Sort (10 marks)

Develop a function for Bubble Sort. You must create an array in main (sized 30), populate it with random values between 0 and 1000, pass the array to the Bubble Sort function which will then sort the array. Main should print the unsorted and sorted array. Test the function on 2 randomly initialized arrays (per function). The method to validate the sorted arrays is in Part 1 D below.

Part 1 D: Testing Bubble Sort (10 marks)

Develop a function to validate Bubble Sort's sorted array. The function receives the sorted array and returns if the array is sorted (no return value) or throws an error using assert. The function simply checks for each pair of adjacent values if they are in order. Test the function on the sorted arrays on part 1 C and finally complete the program by testing a randomly initialized array which should throw an error.

Part 2: Memory (30 marks)

For the recursive functions in Part 1 A and B, describe the stack frames for the provided test values. Show which stack frames are popped and which remain on the stack while the function processes up until the final base case is returned (meaning no further processing when the final base case is reached).

Part 2 A: (15 marks)

Fibonacci where N=4

Part 2 B: (15 marks)

Palindrome for "abcdba"

Part 3: Pointers (30 marks)

Create a program and set up the variables as shown in the diagram. Complete the following steps for the program:

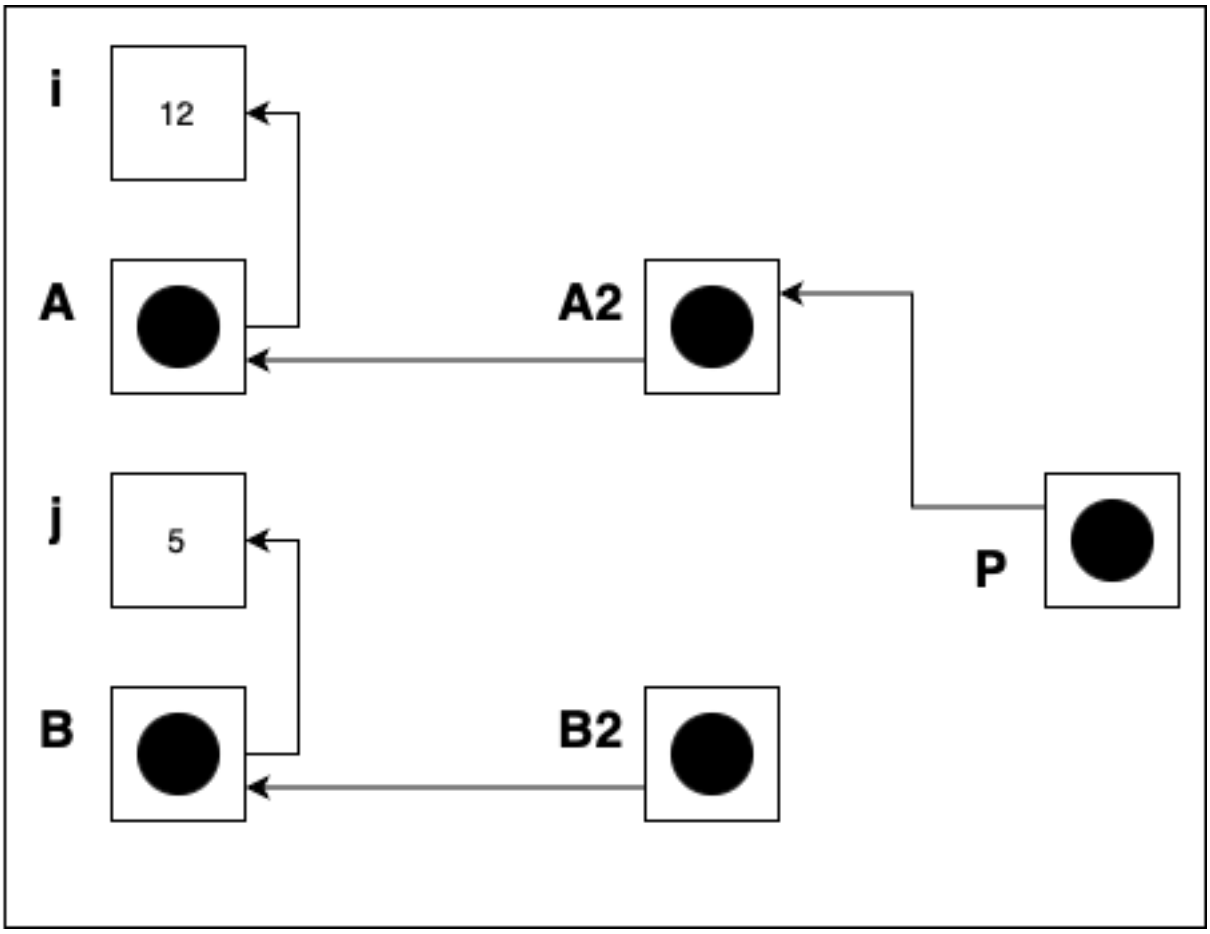
Part 3 A:

1. Create a function 'setter' with parameters P and a new value k (integer). The function should follow the pointer and change the integer at the end of the chain to the new value. **[8 marks]**
2. Call 'setter' in main with k=2. When it returns, demonstrate that the new value of i was changed (*hint: print i*). **[2 marks]**
3. Create a swap function with parameters A2 and B2. The function should swap them such that A2 now points at B and B2 now points to A. Call this function with A2/B2 in main. **[8 marks]**

4. Call 'setter' with k=8.
5. Now print all variables (dereferenced to i or j) in the following format: **[2 marks]**

```
i     : 2
A     : 2  (hint: you must print using the pointer)
B2    : 2

j     : 8
B     : 8
A2    : 8
P     : 8
```



Part 3 B: Finally, draw the new diagram (like above) for the new variable setup. **[10 marks]**

Grading:

- Programs should be named **part1_a1.c** and **part3_a1.c** **(note:** 'a1' represents 'assignment1'). All code for the relevant part should be placed in these files as one C program each.
- Submit both C programs (only as .c), the stack frames in part2 (as .txt, .png, .jpg, .pdf, word, etc.), and the pointer diagram from part3B (as .png, .jpg, .pdf, etc.).
- Programs should be compiled using "gcc -Wall *program.c*" and tested using the appropriate compiler (while not the only option, you can use the compiler available on the university's server accessed through NoMachine or SSH).
- Any programs that do not compile (produce compile errors) automatically **receive a -50% penalty**. This means you must ensure you compile and test your program using the appropriate software.
  - Note that this means if a compile error was generated when the program was tested through 'online compilers' (which is not appropriate software), they will **receive the -50% penalty**.
  - Fatal and non-fatal run-time errors will **receive mark deductions based on assignment grading scheme** (depending on the portion of code causing the error and its severity).
- Programs should be documented properly, meaning you must have comments for each function defined.
- All programs must use the C extension. Formats such as "txt" or other text formats will **receive a partial penalty**. Word documents, PDF formats, or Image formats (like PNG and JPG) which cannot be converted to a C extension will **receive 0 for the associated parts/ questions**. We cannot compile images into programs, so they are not accepted.
- Late assignments **receive -25% for each day that it is late and 0% after 3 days**. In other words:
  - if(submitted_date > submission_deadline + 3 days)
    grade = 0;
  - else if(submitted_date > submission_deadline) {
    int days_late = submitted_date - submission_deadline + 1;
    grade *= 1 - 0.25*days_late;
    }
  - If the assignment is a few hours late due to submission issues, we generally don't apply the late penalty. This is about a 2-3 hour grace period.