Dry-Running and **Pseudocode**

(self reading)

Learning Programming

- Introducing two methods that will help you in learning programming
  1. **Dry-run**: To run a program in your brain
  2. **Pseudocode**: An informal, usually step-by-step description of a program, written in a human readable form
- If you master these two methods, you will gain extra power in learning programming!

Dry Running

- How to best understand the program code?
  - Think like a computer!
    - The computer executes the C program line by line
    - You can also read it line by line
  - This is called a “dry run”
    - You may do it on a piece of paper
    - You may use NotePad/TextEdit or other ways too
    - NEVER read just the starting and ending lines of code and attempt to guess the outcome of the program!

Dry Running with a Trace Table

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5     int side, perimeter, area;
6
7     side = 3;
8     perimeter = 4 * side;
9     area = side * side;
10
11    printf("Side : %d\n", side);
12    printf("Perimeter: %d\n", perimet
13    printf("Area : %d\n", area);
14
15    return 0;
16  }
```

```
Dry Run Table
**********************


side perimeter area
```

The dry run table allow us to track how each variable change as we run the program

Example from Lecture 01b: Computing the perimeter and the area of a square.

Dry Running with a Trace Table

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5     int side, perimeter, area;
6
7     side = 3;
8     perimeter = 4 * side;
9     area = side * side;
10
11    printf("Side : %d\n", side);
12    printf("Perimeter: %d\n", peri
13    printf("Area : %d\n", area);
14
15    return 0;
16  }
```

```
Dry Run Table
*********************

side perimeter area
  ?       ?       ?
```

Variables start with unknown values, before we assign values to them

Example from Lecture 01b: Computing the perimeter and the area of a square.

Dry Running with a Trace Table

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5     int side, perimeter, area;
6
7     side = 3;
8     perimeter = 4 * side;
9     area = side * side;
10
11    printf("Side : %d\n", side);
12    printf("Perimeter: %d\n", peri
13    printf("Area : %d\n", area);
14
15    return 0;
16  }
```

```
Dry Run Table
**********************


side perimeter area
  ?       ?       ?
  3
```

We step forward as if we are the computer, line by line, and write down how the variables will change

Example from Lecture 01b: Computing the perimeter and the area of a square.

Dry Running with a Trace Table

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5     int side, perimeter, area;
6
7     side = 3;
8     perimeter = 4 * side;
9     area = side * side;
10
11    printf("Side : %d\n", side);
12    printf("Perimeter: %d\n", peri
13    printf("Area : %d\n", area);
14
15    return 0;
16  }
```

```
Dry Run Table
********************


side perimeter area
  ?       ?        ?
  3
          12
```

We step forward as if we are the computer, line by line, and write down how the variables will change

Example from Lecture 01b: Computing the perimeter and the area of a square.

Dry Running with a Trace Table

Repeat until the program is finished

- You may even add a column for the `printf` output. Try it!

```
1   #include <std
2
3   int main(void
4   {
5     int side, perimeter, area;
6
7     side = 3;
8     perimeter = 4 * side;
9     area = side * side;
10
11    printf("Side : %d\n", side);
12    printf("Perimeter: %d\n", perimeter);
13    printf("Area : %d\n", area);
14
15    return 0;
16  }
```

```
Dry Run Table
********************


side perimeter area
  ?       ?        ?
  3
          12
                   9
```

Example from Lecture 01b: Computing the perimeter and the area of a square.

Pseudocode

- Maybe your problem is not in understanding a program, but writing one?
  - For some beginners, it is very hard to start writing C from nothing
- Programmers may be very good at thinking like a computer, but we are *still* human beings!
  - As a human, we use **logic** and **natural language**, which are largely programming language independent
- To express our logic and communicate with others, programmers often use a human readable form of programming called **pseudocode**

Pseudocode

- **Example**: I wish to write a program to calculate the sum of an arithmetic sequence from x_1 to x_n
- Recall that an arithmetic sequence is a sequence of numbers with a constant difference between consecutive terms, e.g.

  5, 8, 11, 14, 17, 20, 23

- We may all know that the formula is:

  $$sum = (n * (x_1 + x_n)) / 2$$

  where *n* is the number of terms in the sequence
- In the above example, the sum of the sequence is *7 * (5 + 23) / 2 = 98*

Pseudocode

- Let's attempt to write the program in pseudocode
- There's no restriction in pseudocode syntax, but it should "look like" your target language, i.e. C
- For example, we may write the pseudocode like this. Notice how readable it is?
  1. Declare integers x1, xn, n
  2. Ask for values of x1, xn, n from user (assume valid)
  3. Declare integer sum
  4. Assign sum to be (n * (x1 + xn)) /2
  5. Print out sum

Pseudocode

```
1   #include <stdio.h>
2   int main(void)
3   {
4     // declare integers x1, xn, n
5     // ask for values of x1, xn, n from user
6     // declare integer sum
7     // sum = (n * (x1+xn))/2
8     // print sum
9
10    return 0;
11  }
12
13
14
15
16
```

A good trick is to write down your pseudocode as comments into your program first.

Notice that your pseudocode should be separated into multiple statements, which could be executed step by step like a computer program, to achieve our objective (hopefully).

We use terminology similar to, but not exactly same as, features of C, such as declaring integers (implying integer variable), and printing (implying `printf`).

Pseudocode

```
1   #include <stdio.h>
2   int main(void)
3   {
4     // declare integers x1, xn, n
5     int x1,xn,n;
6     // ask for values of x1, xn, n from user
7     scanf("%d%d%d", &x1, &xn, &n);
8     // declare integer sum
9     int sum;
10    // sum = (n * (x1+xn))/2
11    sum = (n * (x1+xn))/2;
12    // print sum
13    printf("Sum = %d\n", sum);
14
15    return 0;
16  }
```

I can then "translate" each line of my pseudocode and insert one or more valid C statement(s) under each.

It is not necessary a one-to-one translation, but the order of each statement of your pseudocode must be preserved.

Do you think the program is correct?

Pseudocode

```
// declare integers x1, xn, n
// ask for values of x1, xn, n from user
// declare integer sum
// sum = (n * (x1+xn))/2
// print sum
```

A

Pseudocode is very flexible. Here, both A and B are valid pseudocode for the sample program, but B is more concise, and is targeted towards more mature programmers

```
// ask for integers x1, xn, n from user
// declare integer sum = (n * (x1+xn))/2
// print sum
```

B

In B, the first step clearly implies you need to declare x1, xn and n as integers first. In cases where the pseudocode is very concise or ambiguous, it is up to the programmer to use their knowledge and be smart!

Pseudocode

- If you are a beginner programmer and you feel afraid to write in C
  - Write in pseudocode as comments first
- If you face a hard programming problem of C
  - It is also a good idea to write in pseudocode so that you can put your thinking "on paper"
  - You can even discuss with your classmate on its correctness, and he/she will understand your logic better than through a C program

Pseudocode in Lab Exercises

- In your Lab Exercises, sometimes we will give you the correct pseudocode as a hint to the problems!
  - Yay! It's almost like cheating!
  - As long as you can "translate" from the pseudocode to C correctly, your program will be correct!
  - However, as a beginner, you might sometimes translate the pseudocode into C incorrectly
    - In such a case, please make sure you check your C syntax and see if you understand how to express your logic in C properly
  - Future… Other methods to represent an algorithm:
    - Flowchart, Nassi–Shneiderman diagram, etc.