Lab exercises

Task

1. **Use Scratch and familiarize with the followings:**
    a. **Scratch character & backdrop**
    b. **Control program execution**
    c. **Move the scratch character – forward, backward, up & down**
2. **Using Scratch – develop a animation to read values for three numbers and show the sum of three numbers.**
3. **Using Scratch read two numbers and then show the followings:**
    a. **Sum**
    b. **Difference**
    c. **Times**
    d. **Division**

Note: No Submission for week-1.

Tutorial Questions (No need to submit)

1. An Engineering Problem-Solving Methodology

Problem solving is a key part of engineering courses, as well as courses in computer science, mathematics, physics, and chemistry. Therefore, it is important to have a consistent approach to solving problems. It is also helpful if the approach is general enough to work for all these different areas, so that we do not have to learn one technique for mathematics problems, a different technique for physics problems, and so on. The **problem-solving process** that we present works for engineering problems and can be tailored to solve problems in other areas as well; however, it does assume that we are using the computer to help solve the problem.

The process or methodology for problem solving that we will use throughout this text has five steps:

1. State the problem clearly.
2. Describe the input and output information.
3. Work the problem by hand (or with a calculator) for a simple set of data.
4. Develop a solution and convert it to a computer program.
5. Test the solution with a variety of data.

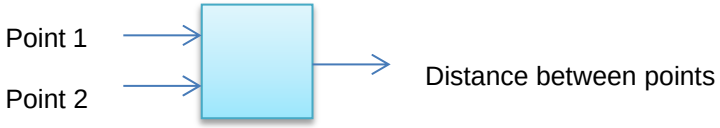
We now discuss each of these steps using an example of computing the distance between two points in a plane.

i. PROBLEM STATEMENT

The first step is to state the problem clearly. It is extremely important to give a clear, concise problem statement to avoid any misunderstandings. For this example, the problem statement is the following: Compute the straight-line distance between two points in a plane.

ii. INPUT/OUTPUT DESCRIPTION

The second step is to carefully describe the information that is given to solve the problem and then identify the values to be computed. These items represent the input and the output for the problem and collectively can be called input/output (I/O). For many problems, a diagram that shows the input and output is useful. At this point, the program is an “abstraction” because we are not defining the steps to determine the output; instead, we are only showing the information that is used to compute the output. The **I/O diagram** for this example follows:



iii. HAND EXAMPLE

The third step is to work the problem by hand or with a calculator, using a simple set of data. This is a very important step, and should not be skipped even for simple problems. This is the step in which you work out the details of the problem solution. If you cannot take a simple set of numbers and compute the output (either by hand or with a calculator), then you are not ready to move on to the next step; you should read

the problem again and perhaps consult reference material. The solution by hand for this specific example is as follows:

Let the points p1 and p2 have the following coordinates:

p1=(1,5); p2=(4, 7).

We want to compute the distance between the two points, which is the hypotenuse of a right triangle, as shown in Figure 1.4. Using the Pythagorean theorem, we can compute the distance with the following equation: distance = sqrt ((side1)^2 + (side2)^2) = sqrt ((4-1)^2 + (7-5)^2) = 3.61.

iv. ALGORITHM DEVELOPMENT

Once you can work the problem for a simple set of data, you are ready to develop an **algorithm**, or a step-by-step outline, of the problem solution. For simple problems such as this one, the algorithm can be listed as operations that are performed one after another. This outline of steps decomposes the problem into simpler steps, as shown by the following outline of the steps required to compute and print the distance between two points:

Decomposition Outline

1. Give values to the two points.
2. Compute the lengths of the two sides of the right triangle generated by the two points.
3. Compute the distance between the two points, which is equal to the length of the hypotenuse of the triangle.
4. Print the distance between the two points.

This **decomposition outline** is then converted to C commands so that we can use the computer to perform the computations. From the following solution, you can see that the commands are very similar to the steps used in the hand example. The details of these commands are explained in Chapter 2.

```
/*-------------------------------------------------------------*/
/* Program chapter1_1
*/
/*
*/
/* This program computes the
*/
/* distance between two points.
*/
#include <stdio.h>
#include <math.h> int
main(void)
{
/* Declare and initialize variables. */ double x1=1,
y1=5, x2=4, y2=7,side_1, side_2, distance;
/* Compute sides of a right triangle. */
side_1 = x2 - x1; side_2 = y2 - y1;
distance = sqrt(side_1*side_1 + side_2*side_2);
 /* Print distance. */ printf("The distance between the two points is
""%5.2f \n",distance);
```

```
/* Exit program. */
return 0;

}
/*-------------------------------------------------------------*/
```

v. TESTING

The final step in our problem-solving process is testing the solution. We should first test the solution with the data from the hand example because we have already computed the solution. When the C statements in this solution are executed, the computer displays the following output:

The distance between the two points is 3.61

This output matches the value that we calculated by hand. If the C solution did not match the hand calculated solution, then we should review both solutions to find the error. Once the solution works for the hand-calculated example, we should also test it with additional sets of data to be sure that the solution works for other valid sets of data. The set of steps demonstrated in this example are used in developing the programs in the Problem Solving Applied sections in the chapters that follow.