# Assignment: Algorithm Design and Problem Solving

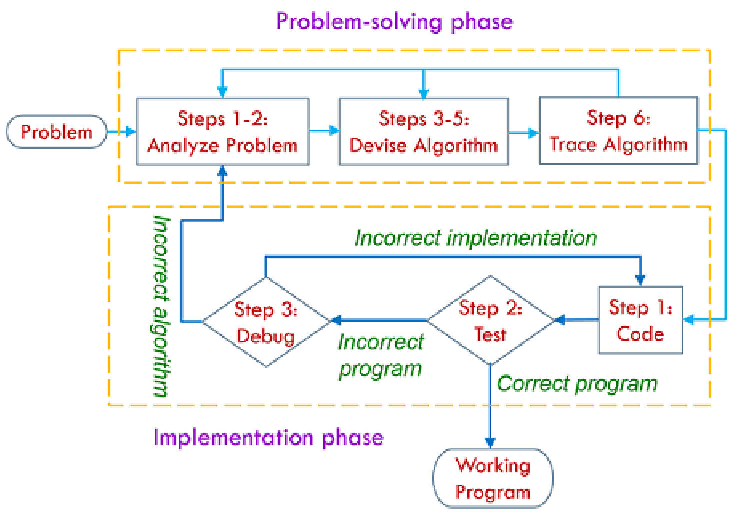
## Learning Outcomes:

- Implement algorithms
- Illustrate algorithms using flowcharts and pseudocode
- Understand the problem solving process

## Introduction

"Resist the temptation to code" is an old programming proverb. It is important to realize that the program's design, as represented by an algorithm, is done *before* you write your program. In this respect, it is like an architect's blueprint. You would not start to build your dream house without a detailed set of plans. Yet one of the most common errors of both experienced and new programmers alike is to start coding a program before the design is complete and fully documented. This rush to start is due in part to programmers' thinking they fully understand the problem and also due to their excitement about solving a new problem. In the first case, what they find is that they did not fully understand the problem. By taking the time to design the program, they will raise more questions that must be answered and therefore will gain a better understanding of the problem. The second reason programmers code before completing the design is just human nature. Programming is a tremendously exciting task. To see your design begin to take shape, to see your program creation working for the first time, brings a form of personal satisfaction that is a natural high. Remember, the primary intent of the design process is to increase quality (resulting in better grades for you), save time (resulting in more free time that you could devote to some downtime or studying for other courses), and in an industrial and professional setting to save money. Therefore, you must train yourself to always begin coding only after you've carefully analyzed the problem and have a blueprint that explicitly drives the coding process.

To structure program development, class lectures introduced the *program design process*. The program design process is divided into two general phases: the *problem-solving* phase and the *implementation* phase, each of which is further subdivided into multiple steps. A high-level view of the program design process is given in the following picture:



This tutorial is only concerned with the problem-solving phase; future tutorials will incorporate the implementation phase. The six steps of the problem-solving phase are:

1. Understand the problem clearly.
2. Describe the input and output information.
3. Work the problem by hand for a simple data set.
4. Decompose solution from previous step into step-by-step details.
5. Generalize the steps into an algorithm.
6. Test the algorithm with a broader variety of data.

The end result of the planning process is an algorithm that consists of a sequence of steps that are *unambiguous*, *executable*, and *terminating*. Unambiguous means each step has precise instruction for what to do and where to go next. Executable means that each step can be carried out in practice because there is well-defined data on which the precise instructions can be applied. Terminating means that the sequence of steps is finite and will eventually come to an end.

## Tasks

This tutorial requires you to complete two tasks. Partial solutions are provided for both tasks but you need to flesh out the details by progressing through each of the six steps in the problem-solving phase.

The submission details are listed here.

### Task 1

Consider the following investment problem:

*You put a certain amount into a bank account that earns a certain percentage of interest per year. How many years does it take for the account balance to be double the original amount?*

Design an algorithm consisting of both a *flowchart* and *pseudocode* using the six steps of the problem-solving phase. You're starting your journey as a computer scientist/engineer. Avoid short cuts. Instead, work your way through the six steps. It is tedious work but the reward is that you are understanding the most crucial aspects involved in problem-solving that you will be using and relying upon throughout your professional career.

### Partial solution for Task 1

1. Step $1$: As always, the first step is to write a concise statement that shows that you understand the problem. Something like this would be appropriate:

   > Given an annual percentage of interest, compute the number of years required to double an investment using compounded interest.

2. Step $2$: Using the problem statement from step $1$, in step $2$ you must identify the input(s) required by the problem and its output(s). It is clear that the investment (the amount in the bank account) and the annual percentage of interest are the inputs to the problem which then computes the number of years required to double the initial investment. Think about the types of values. Can the investment be values such as $100.45$ or $9{,}985.62$? Can annual interest rates have values such as $4.2\%$ or $2.63\%$? Without worrying about a solution to the problem, draw a picture that clearly identifies the inputs, output, and their types (integer or floating-point).

3. Step $3$ requires you to work the problem by hand or by using a calculator. Assume you have deposited $\$10{,}000$ in an account with an interest rate of $11\%$. Use the following table to compute the interest (using compounding interest) at the end of each year and grow the investment. You keep going until the balance is at least $\$20{,}000$.

| **year** | **interest ($\%$)** | **balance ($\$$)** |
|---|---|---|
| $0$ | | $10000$ |
| $1$ | $10000 \times 0.11 = 1100.00$ | $10000.00 + 1100.00 = 11100.00$ |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

Carrying out these computations is intensely boring but you're doing two things. First, you are able to provide a clear and unambiguous description of the steps necessary to design an algorithm. Second, you are building test data that can be used when testing both your algorithm and the code that implements this algorithm.

4. Step $4$: For this step, you must think about what you did to solve the problem in step $3$, and write down the individual steps to solve that particular instance. A partial listing of these individual steps would look like this:

$$\text{Input:} \quad B = \$10,000.00,\ I = \frac{11}{100} = 0.11\%$$
$$\text{Output: } Y(B, I) = 7$$

```
1. Compute balance $B$ at end of year 1:
    Compute $interest = B \times I = 10000.00 \times 0.11 = 1100.00$
    Add $interest$ to $balance$ : $10000.00 + 1100.00 = 11100.00$
    You get $balance$ at end of year 1: $B = 11100.00$
2. Compute balance $B$ at end of year 2:
    Compute $interest = B \times I = 11100.00 \times 0.11 = 1221.00$
    Add $interest$ to $balance$ : $11100.00 + 1221.00 = 12321.00$
    You get $balance$ at end of year 2: $B = 12321.00$
```

5. Step $5$: Now, you can begin to generalize the steps listed in step $4$ into an algorithm. The pattern to be discerned is that the computations must be repeated until the investment has at least doubled. The year in which the investment is at least double of original investment is the answer to the problem. Provide both the flowchart and pseudocode .

6. Step $6$: Test your algorithm by working through some sample data. If you're unable to get the same result (such as from an external source that computes the number of years for compounding interest to double an investment), then you'll have to go back and revisit your algorithm from step $2$. For example, how many years would it take for an initial investment

of $\$100$ to double if the interest rate is $6\%$? Is the answer $12$ years? What about if the interest rate is $12\%$? Is the answer $7$ years?

## Task 2

Consider the following car purchase problem:

*You have the choice of buying two new cars. One is more fuel efficient than the other, but also more expensive. You know the purchase price and fuel efficiency (in miles per gallon, mpg) of both cars. You plan to keep the car for $10$ years. Assume a price of $\$4$ per gallon of gas and usage of $15{,}000$ miles per year. You're wealthy and can afford to pay cash and not worry about financing costs. Which car is the better deal?*

Design a pseudocode algorithm (flowchart is not necessary) using the six steps of the problem-solving phase. Test your algorithm by working through some sample data.

## Partial solution for Task 2

1. Step $1$: The problem statement would look like this:

   > Given the purchase price and fuel efficiency of car $1$ and car $2$, which of these two cars will be more economical in terms of total ownership costs after $10$ years if there are no financing costs, the price per gallon of gas is $\$4$ and mileage per year is $15{,}000$ miles?

2. Step $2$: Using the problem statement from step $1$, in step $2$ you must identify the input(s) required by the problem and its output(s). Since the financing costs, price per gallon of gas, and mileage per year are constant and do not change, the problem has $2$ inputs per car (purchase price and fuel efficiency) for a total of $4$ inputs. Are these inputs integers, floating-point values, or a combination of both? Clearly, the output must be an integer value ($1$ or $2$) identifying the car that is more economical to purchase and operate over $10$ years. Without worrying about a solution to the problem, draw a picture that clearly identifies the inputs, output, and their types (integer or floating-point).
3. Step $3$: Could you solve this problem by hand? Sure - by breaking the problem into smaller tasks. For each car, you want to know the *total ownership cost*. The ownership cost is the sum of the purchase price and the operating cost for $10$ years. Do this computation separately for each car. Once you've the total cost for each car, you can then decide which car is the better deal. The purchase price for each car is provided as an input to the problem. You will need to device a formula for computing the operating cost for $10$ years using $\$4$ as the price per gallon of gas and mileage per year of $15{,}000$ miles.
4. For this step, you must think about what you did to solve the problem in step $3$, and write down the individual steps to solve that particular instance.
5. As with the previous task, generalize your hand-crafted solution into an algorithm that is described in the form of pseudocode.

## Submission

It is your responsibility to make sure that your flowchart (for task $1$) and pseudocodes (for both tasks) are clear and legible. What we can't read, we can't understand, and therefore we can't grade. Use the lab printer to scan your submission to a pdf file. If you decide to take a picture using your phone, you're then expected to use this free online tool to convert the jpeg file to pdf.

Combine your solutions to Task 1 and Task 2 into a single file. If your DigiPen login is **foo** (or something like **foo.boo**), then name your file `foo+ass1.pdf`. Upload this pdf file to the tutorial submission page.

## If you're done early and are bored ...

If you're done implementing the problem-solving phase with plenty of time to spare, try implementing the two algorithms in your favorite programming language.