CS 201 Notes

Introduction to Computer Science

Jaeden Bardati

September 24, 2021

0 Course Overview

September 10, 2021

0.1 What is Computer Science?

Computer science is the study of **algorithms**.

An **algorithm** is an effective method for solving a problem, expressed as a finite sequence of steps.

The development of algorithms works in this recurring order:

- **Design**
- **Analysis**
- **Implement**
- **Experiment**

In the **design** phase, we design an algorithm using pseudo-code. In the **analysis** phase, we need to analyze the correctness and the efficiency. That is to say, we make sure that our algorithm will work and completes in a reasonable amount of time. During the **implementation**, the algorithm is written in code on a computer. Finally, the algorithm is run and debugged in the **experiment** phase. This process repeats.

1 Introduction to Algorithms

1.1 Design

September 11, 2021

An algorithm is a step by step procedure to solve a problem. For example,

- Step 1: Do something
- Step 2: Do something
- ...
- Step n: Do something

There are three basic types of steps. Note that there is a fourth (recursion), but it is not covered in this course.

1.1.1 Sequential Steps

Do a single task.

For example: Let x be a variable. A sequential step could be to add 1 to x.

1.1.2 Conditional Steps

Ask a question that supports only **logic answers** (true or false answer).

For example: Let x be a variable. A conditional step could be to ask if $x > 0$. If so, add 1 to x; otherwise, subtract 1 from x.

1.1.3 Iterative Steps (loops)

Repeat a task until a certain condition is satisfied. This step links the sequential step to the conditional steps.

For example: If you have a recipe that you need to add water until its dry. An iterative step would be one where you add $\frac{1}{2}$ cup to mixture while mixture is dry.

1.2 Case Study: Addition Algorithm
September 11, 2021

Let's say we want to add 472 to 593. We would to it like so:

$$\begin{array}{cccc} & {\scriptstyle 1} & & \\ & 4 & 7 & 2 \\ + & 5 & 9 & 3 \\ \hline 1 & 0 & 6 & 5 \end{array}$$

If you know how to add these numbers, you know how to do it for any numbers. Why? Because we used a sequence of steps to solve it: We used an algorithm. What is this algorithm?

We know that we can break down the work for each of the digits. It is an iterative statement for each digit. What is the work we need to do at each iteration?

First, we add the digits plus the carry in (starts at 0). This is a **sequential step**. Then, we ask if it is greater than 9. If so, we set the resulting digit as the addition subtracted by 10 and set the carry out (which is te carry in for the next step) to 1; otherwise, we simply set the resulting digit as the addition and set the carry out to be 0. This is a **conditional step**. Then we repeat this process until we have no more digits to add. This is an **iterative step**.

Now, we need to conceptualize this. Let's let $m \geq 1$ be the number of digits. Let us define a_i (first number digits), b_i (second number digits) and c_i (resulting number digits) as follows:

	a_{m-1}	...	a_1	a_0
+	b_{m-1}	...	b_1	b_0
c_m	c_{m-1}	...	c_1	c_0

Let's write the steps of our algorithm:

Algorithm 1
Addition Algorithm

1: **get** m (provided by user)
2: **get** a_{m-1}, ... a_1, a_0 and b_{m-1}, ... b_1, b_0 (provided by user)
3: **set** $i = 0$ (digit index) and carry $= 0$
4: **while** $(i \leq m - 1)$ **do**
5: **set** $c_i = a_i + b_i +$ carry
6: **if** $(c_i \geq 10)$ **then**
7: **set** $c_i = c_i - 10$
8: **set** carry $= 1$
9: **else**
10: **set** carry $= 0$
11: **set** $i = i + 1$
12: **set** $c_m =$ carry
13: **print** c_{m-1}, ... c_1, c_0
14: **stop**

This can be programmed now using a programming language.

If you want to test your algorithm, you can perform a **trace**. A trace is when you go through the algorithm yourself step by step for a test case.

1.3 Pseudocode

September 11, 2021

Pseudocode is:

- Simplified
- A tradeoff between natural and programming languages
- Not unique

We will now look into the types of statements and the syntax we will use.

1.3.1 Sequential statements

- **Input**: Get "variable". E.g. Get m, Get radius
- **Computation**: Set variable = expression. E.g. Set area = $\pi\times$ radius^2
- **Output**: Print "variable". E.g. Print area.

Algorithm 2
Calculates the average of three numbers.

1: **get** x, y, z
2: **set** average = $\frac{x+y+z}{3}$
3: **print** average
4: **stop**

1.3.2 Conditional statements

- If (condition) Then
    operation T_1
    operation T_2
    ...
    operation T_m
  Else
    operation F_1
    operation F_2
    ...
    operation F_n

When a conditional statements within a conditional statement it is called a **nested** conditional statements (or nested ifs). This also applies to nested iterative statements (or nested loops).

Algorithm 3
Print average of three number if the first number is larger than 0, otherwise print an error message.

1: **get** x, y, z
2: **if** $(x \geq 0)$ **then**
3: **set** average = $\frac{x+y+z}{3}$
4: **print** average
5: **else**
6: **print** "Bad Data"
7: **stop**

1.3.3 Iterative statements

- While (condition) Do step i to step j
  - Step i: operation
  - Step $i + 1$: operation
  - ...
  - Step j: operation
- Stop

There are some considerations that you have to be careful of when writing a while loop:

If the condition is initially false, the loop will not execute at all.

If the condition is initially true, the loop is iterated until the condition is false. This means that *at least one step should change the condition at some point*. If this is forgotten, the loop will run forever (called an **infinite loop**)! This is considered a **fatal error**.

Algorithm 4
Compute and print the square of the first 100 integers.

1: **set** index=1
2: **while** (index $\leq$ 100) **do**
3: **set** square = index * index
4: **print** square
5: **set** index = index + 1
6: **stop**

1.3.4 The Do-While

September 23, 2021

The do-while is similar to the while-do, but you check the condition after the do section.

- Do
  - Step i: operation
  - Step $i + 1$: operation
  - ...
  - Step j: operation
- While (condition)

This will always execute at least once. It will only execute once if the condition is false, and multiple if it is true. For example,

Algorithm 5
Read a var x, print $\sqrt{x}$ and repeat the process as long as requested by user.

```
1: get x
2: do
3:     get x
4:     if (x ≥ 0) then
5:         set root = √x
6:         print root
7:     else
8:         print "Bad data"
9:     print "Do you want to continue? Y/N"
10:    get continue
11: while (continue == 'Y')
12: stop
```