1. **[48 points]** Write a Java program that will simulate **First Come First Serve**, **Shortest Remaining Time**, and **Round Robin** scheduling algorithms. For each algorithm, the program should compute the turnaround time and the waiting time of every job as well as the average turnaround time and average waiting time.

   The process information will be read from an input file. The format is as follows:

   ```
   Process_id,Arrival_time,CPU_time
   ```

   All fields are integers where:
   `Process_id` is a unique numeric process ID
   `Arrival_time` is the time when the process arrives in milliseconds
   `CPU_time` is the amount of estimated CPU time requested by the process
   All the fields will be delimited by a comma.

   An example of a sample input file is as follows:

   ```
   1,0,8
   2,1,4
   3,2,9
   4,3,5
   ```

   The program will accept command line arguments. The program usage will be as follows:

   ```
   ./Question1 input_file [FCFS|SRT|RR] [time_quantum]
   ```

   where, `Question1` is name of the java program and `input_file` is the input file containing the process information. FCFS refers to the First Come First Serve policy and is non-pre-emptive. SRT refers to Shortest Remaining Time and is pre-emptive. RR refers to Round Robin, which is pre-emptive and the time quantum only applies to this algorithm. Ignore context switching time. Note the last argument is only needed in case of Round Robin.

   Suppose you want to run the SRT algorithm on a process list contained in input file called `process.txt`, then you would run your program as follows:

   ```
   ./Question1 process.txt SRT
   ```

   If you want to run the RR algorithm on the same input file with a time quantum of 5 milliseconds, then you would run your program as follows:

```
./Question1 process.txt RR 5
```

Turnaround time = completion time – arrival time
Waiting time = Turnaround time – CPU time taken by the process

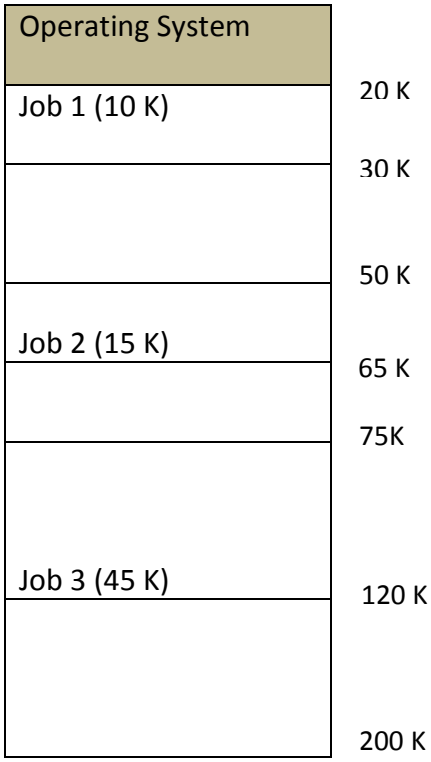
When the program is run, it should print statistical information to the screen formatted in the following fashion:

```
==========================================
Process ID      | Turnaround time     | Waiting time
==========================================
                |                     |
------------------------------------------------------------------
                |                     |
------------------------------------------------------------------
……..
==========================================
Avg.            |                     |
==========================================
```

For example, if you run the program with RR scheduling using a time quantum of 4 milliseconds on `process.txt`, then the output to the screen should be as follows:

```
==========================================
Process ID      | Turnaround time     | Waiting time
==========================================
1               | 20                  |12
------------------------------------------------------------------
2               |7                    |3
------------------------------------------------------------------
3               |24                   |15
------------------------------------------------------------------
4               |22                   |17
==========================================
Avg.            |18.25                |11.75
==========================================
```

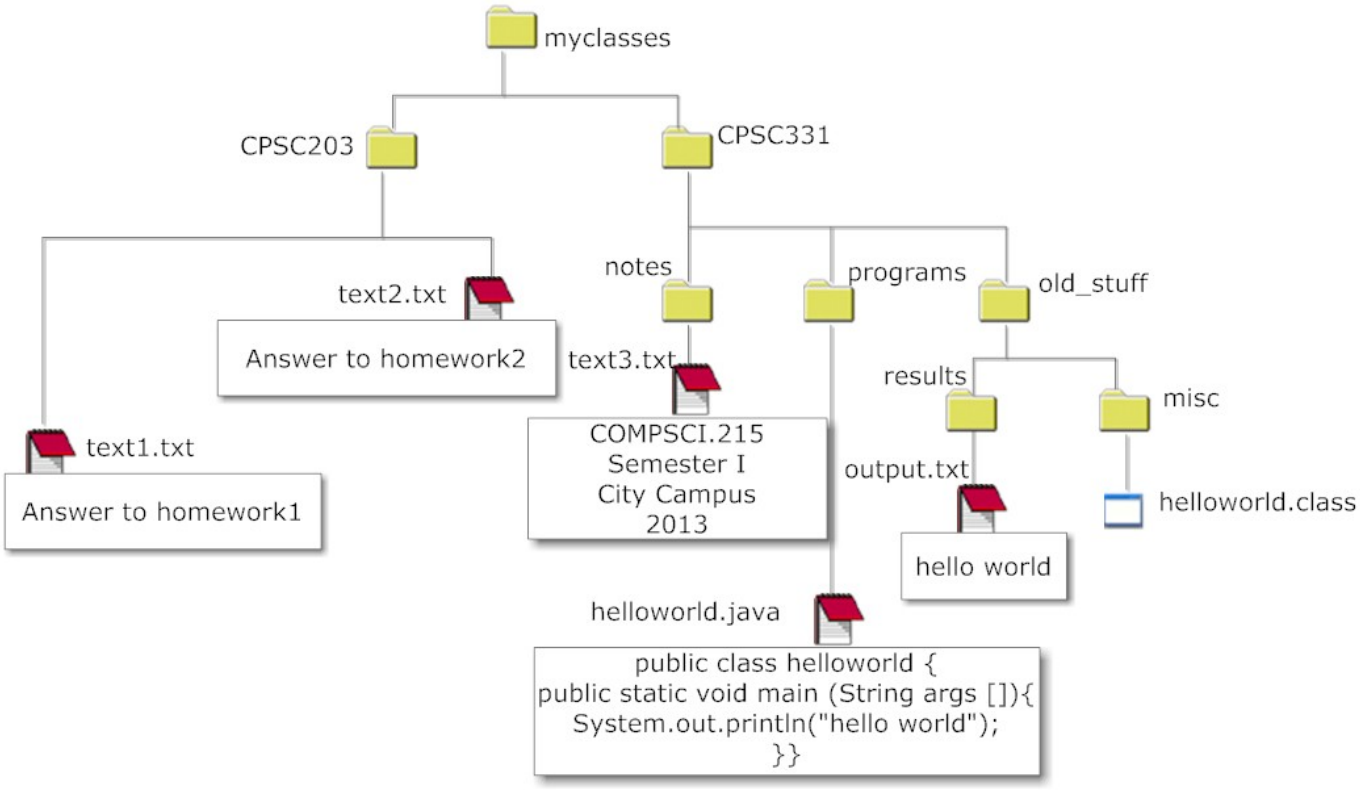
2. **[14 points]** Given the memory configuration in the figure, answer the following questions. At this point, Job 4 arrives requesting a block of 100 K.
    a. Can Job 4 be accommodated? Give reasons for your answer.
    b. If relocation is used, what are the contents of the relocation registers for Job 1, Job 2, and Job 3 after compaction?
    c. What are the contents of the relocation register for Job 4 after it has been loaded into memory?
    d. An instruction that is part of Job 1 was originally loaded into memory location 22 K. What is its new location after compaction?
    e. An instruction that is part of Job 2 was originally loaded into memory location 55 K. What is its new location after compaction?
    f. An instruction that is part of Job 3 was originally loaded into memory location 80 K. What is its new location after compaction?
    g. If an instruction was originally loaded into memory location 110 K, what is its new location after compaction?



3. **[10 mark]** You will implement conditional processing and nest if statements. The firm named TJ Watson and Associates needs a shell script written. The script needs to accept two values, Status and Years. The status must be either “S”, “H”, or “C” (for “salary,” “hour,” and “contract” respectively); otherwise, an error message is displayed. An employee with a status of “H” or “S” and who has been employed for at least 2 years will receive 50 shares of company stock. A contractor who has been working with TJ Watson and Associates for more than three years will receive a $100 bonus. A message must be displayed for employees or contractors not meeting the requirements saying they are not eligible. The user will be prompted to enter the Status and Year, and the script will print the appropriate output to the screen.

4. **[14 mark]** You will create a script that generates a random number using the shell’s built in RANDOM variable. The user will be prompted to enter a number between 1 and 10, inclusive. The loop will continue until the user guesses the correct answer. The $RANDOM statement will generate a random number between 0 and 32,767; therefore, a $RANDOM statement within the loop should ensure a random number between 1 and 10, inclusive is generated by the system.

5. **[14 mark]** Write a shell script that, when executed, builds the directory tree and files shown below. The diagram shows the required directory structure and files within those directories. Most files are text files. One is a Java file and its compiled class file. The names of the directories and files are shown besides the icons. The contents of the files are shown in the textboxes below the file icons.



Note that the text in text2.txt, text1.txt, and text3.txt should appear exactly as shown in the figure. That is, “Answer to homework2” and “Answer to homework1” should appear on a single line in text2.txt and text1.txt, respectively. In case of text3.txt, the contents should appear on each line separately.

6. **[OPTIONAL, BONUS: 20 points]** After executing the script in Problem 5, copy `dirtree.sh` into `myclasses`. The script `dirtree.sh` recursively searches the contents of your current directory and outputs a tree of its contents. When you run `dirtree.sh` you will get the following output (Note the indentation of the directory names and the file names):

```
/afs/ec.auckland.ac.nz/users/a/m/amah811/unixhome/myclasses
`---CPSC203
|   `---test1.txt
|   `---test2.txt
`---CPSC331
|   `---notes
|   |   `---text3.txt
|   `---old_stuff
|   |   `---misc
|   |   |   `---helloworld.class
|   |   `---results
|   |   |   `---output.txt
|   `---programs
|   |   `---helloworld.java
`---dirtree.sh
```

Write a shell script to create a tree structure of the contents of a directory. In other words, you have to write a script that does the same job as `dirtree.sh`.

Maximum points: 100 (10% of the total grade)

Deliverables:

- Java source code for Problem1. Provide a readme (problem1.txt) to explain the usage of the program.
- Answers to Problem 2 should be provided in a PDF called "Problem2.pdf"
- Script for Problem 3 should be named problem3.sh
- Script for Problem 4 should be named problem4.sh
- Script for Problem 5 should be named problem5.sh
- Script for Problem 6 should be named problem6.sh
- Put all the files in a zip file called 215A1_UPI.zip and upload to ADB (replace UPI with your UPI).
- To get credit for your work, all programs and scripts should run on *login.cs.auckland.nz* or *login01.fos.auckland.ac.nz*

Marking Scheme:

Problem 1.

- 16 points per policy implementation. Full points, if the code compiles and produces correct output.
- 12 points per policy implementation for partial policy implementation such as the program does not compile or produces incorrect results. Note the code should be well developed for the marker to know that the student put in reasonable effort to get the code working.
- 8 points per policy for attempting to write the program. The student needs to show that he/she tried to solve the problem.

Problem 2.

- 2 points per sub-question for correct answer. No partial points awarded.

Problem 3.

- Full points for script that executes as per the problem statement.
- 5 points per policy for attempting to write the program. The student needs to show that he/she tried to solve the problem.

Problem 4.

- Full points for script that executes as per the problem statement.
- 5 points per policy for attempting to write the program. The student needs to show that he/she tried to solve the problem.

Problem 5.

- Full points for script that executes as per the problem statement.
- No partial points awarded for incomplete script or a script that does not fully automate the creation of the directories and files, the compiling of the java program, and moving of the files.

Problem 6.

- Full points for script that executes as per the problem statement.
- 10 points if the script can only produce the tree structure for myclasses/;i.e., the script does not produce correct tree structure for any other directory.