# JIT121 Programming Tutorial 4

## Learning Objectives

1. Use of **enum** (enumeration type)
2. Use of **switch** statement
3. Understanding Short Circuit Evaluation
4. Using various searching and sorting algorithms
5. Designing algorithms and writing code on PAPER

## Assumptions

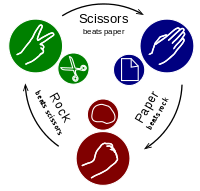
That you are already familiar with;

* Lectures 1 – 4 & Tutorials 1 - 3

## Activity Overview

1. Rock-paper-scissors game - **enum**
2. Choose a car – **switch** and **enum**
3. Implementing a Linear Search algorithm
4. Short Circuit Evaluation
5. Tracing Insertion Sort algorithm by hand
6. Implementing BubbleSort algorithm

**Activity 1: Rock-paper-scissors Game**

Rock-paper-scissors is a game usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand. These shapes ae “rock” (a closed fist), “paper” (a flat hand) and “scissors” (a fist with index and middle fingers forming a V).

The game has only three possible winning outcomes: a player who plays rock will beat the other player who has played scissors (“rock crushes scissors”) but will lose if the other player has played paper (“paper covers rock”); a play of paper will lose to a play of scissors (“scissors cut paper”). If both players play the same shape, the game is tied and is immediately replayed.

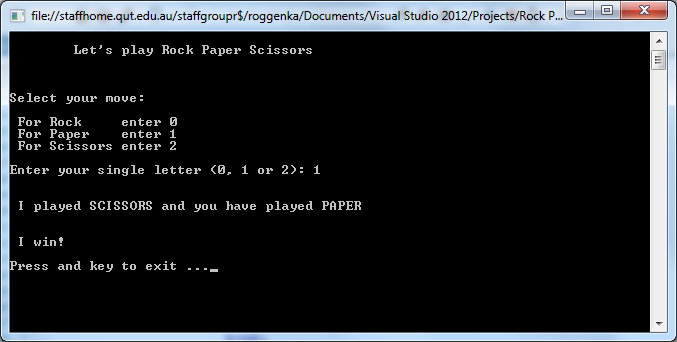
Write a program which simulates the rock-paper-scissors game where one player is the computer and the other is the user. The computer will randomly select their shape whilst the user will enter a number (0, 1 or 2) to select their shape.

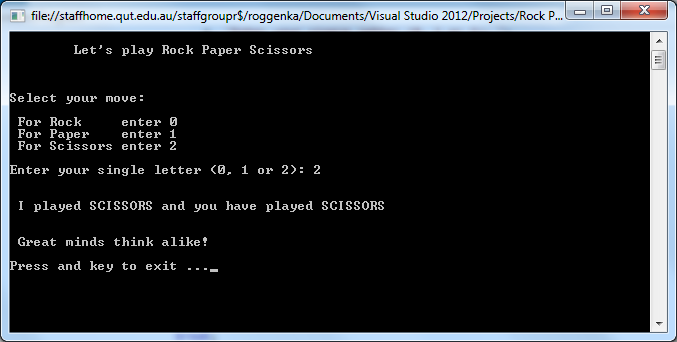
Initially implement code to play a single round of the game. Use the following declarations as Class (global) variables and use both in your solution.

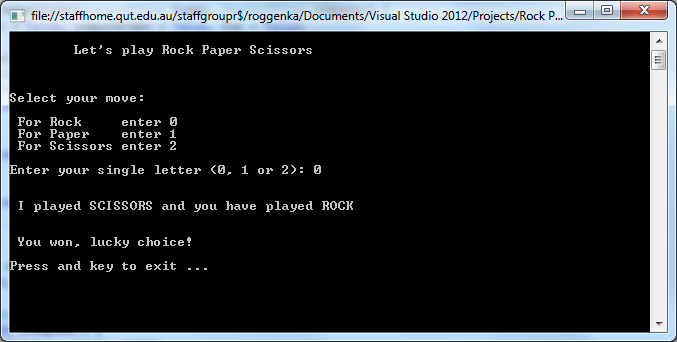
static Random randomChoice = new Random(100);

enum StateOfPlay {ROCK, PAPER, SCISSORS}

The following screenshots shows three executions of the initial implementation with the computer always selecting “scissors” due to using a “seed” parameter of 100 in the declaration of the **Random** variable randomChoice and the user selecting each of the possible values.



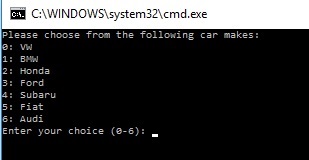


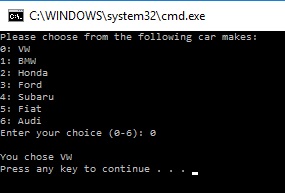


Develop and implement the code incrementally, so that in the event of a tie the game is replayed immediately otherwise the user is asked if they wish to play another game. You could keep a count of the number of games won by each player and report this when the user decides to stop playing. Only add error checking on the user inputs as the last modification.

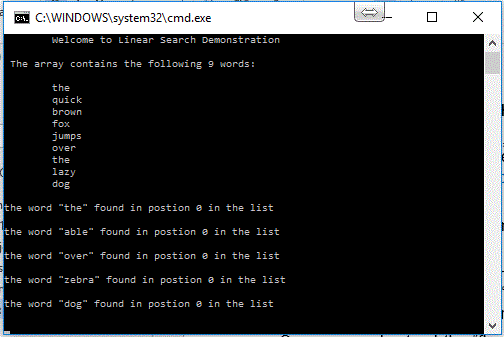
**Activity 2: Choose a Car**

1. Write a program (just in **Main** will do) to:

* Display a list of cars and ask the user to choose one of them by typing an integer (see screen shot)

1. Use a **switch** block to output a message to the screen, depending on the car chosen by the user e.g.:
2. Modify your solution in (a) above to use an **enum** to store the car values (7 of them in the example given above). Replace the **switch** block by creating a variable of that **enum** type and assigning to it the value of the user's choice, cast to that **enum** type. (See Lecture 4 for an example.)

### Activity 3: Searching Algorithm

Extract the file **LinearSearch.zip**. Open the **LinearSearch** folder, then open the solution file, **LinearSearch.sln** in Visual Studio and examine the source code.It contains a **Main** method, and various other complete methods **DisplayArray,**  **OutputSearchResult, OutputMessage** and an incomplete method **LinearSearch.**

**Main** is a simple “driver” method: code to test your eventual implementation of the **LinearSearch** method.

Run the program as it is given to you. The output should be as shown in the first screen shot on the right:

The program will output that each word is found at the same position: 0. That is because at the moment the **LinearSearch** methodis currently a “stub”. A stub is minimal code to ensure that the code compiles but does not necessarily perform the required functionality at all.

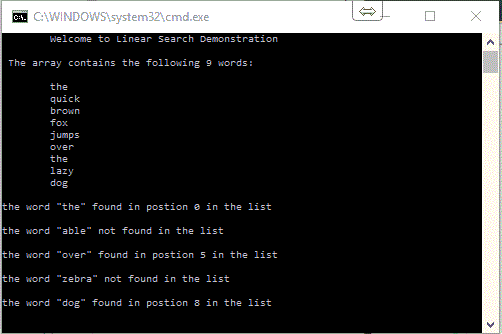
Once you understand the “flow of control” of this program you are to finish implementing the body of **LinearSearch** by implementing the algorithm supplied as a comment.

Figure 1 - supplied

Your program should then produce the output shown in the screen shot on the right when executed:

### Activity 4: Short Circuit Evaluation

Using your program from the previous activity.

Change the position of the operands of the **while** loop in your program from the implementation of the given algorithm:

//while (position < length of list) & (element != key)

to the implementation of this algorithm instead:

//while (element != key) & (position < length of list)

Compile your program and execute it. You should see an “Index Out Of Range” exception.

Why does the ordering of these operands make a difference ??

### Activity 5: Tracing Insertion Sort Algorithm

1. Trace the action of the **insertion sort** algorithm (see the lecture slides for the algorithm) on the following sequence of 9 integers. Show the relative order of the values after each pass of the algorithm. There will be 8 passes altogether.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **31** | **15** | **41** | **28** | **87** | **55** | **22** | **46** | **32** |
| pass 1 | **15** | **31** | 41 | 28 | 87 | 55 | 22 | 46 | 32 |
| pass 2 | **15** | **31** | **41** | 28 | 87 | 55 | 22 | 46 | 32 |
| pass 3 | **15** | **28** | **31** | **41** | 87 | 55 | 22 | 46 | 32 |
| pass 4 | **15** | **28** | **31** | **41** | **87** | 55 | 22 | 46 | 32 |
| pass 5 | **15** | **28** | **31** | **41** | **55** | **87** | 22 | 46 | 32 |
| pass 6 | **15** | **22** | **28** | **31** | **41** | **55** | **87** | 46 | 32 |
| pass 7 | **15** | **22** | **28** | **31** | **41** | **46** | **55** | **87** | 32 |
| pass 8 | **15** | **22** | **28** | **31** | **32** | **41** | **46** | **55** | **87** |

### Activity 6: Bubble Sort

For this activity, you will practice writing solutions on PAPER.

1. Write down the algorithm for the Bubble Sort method, leaving enough space on your paper between each step to eventually write the code. Hint: use a pencil so you can make changes as necessary.
2. On another piece of paper, trace your algorithm using the following array:

**int**[] arr = **new** **int**[10] { 23, 2, 3, 34, 6, 1, 24, 45, 78, 8};

1. Under each step of your algorithm, write C# code for the Bubble Sort method. (The method should have one parameter, an unsorted integer array.)
2. Write code to sort an integer array of 10 elements in ascending order using the **bubble sort** algorithm with this array and method call: