CSC8501 Coursework 1 – 2018
The Safe Problem
Due 26th October 2017 at 10am

Set by Graham.Morgan@ncl.ac.uk

0 1 2 3 4

Specification (what you need to do): You will build a computer program in C++ to demonstrate strategies for constructing a simulation of a multi-lock safe

Clarifying the problem (approach to take)

- A combination lock has four numbers represented on dials
- Each number of the combination lock may be turned in sequence one movement at a time (i.e., to get from 1 to 7 you can either move through 2, 3, 4, 5, 6 or 0, 9, 8)
- When a button on a combination lock is pressed the lock will either open or remain closed (depending on the combination entered)
- A multi-lock safe consists of a series of 5 combination locks and to lock a safe each combination lock must be locked in turn (lock 0 first through to lock 4)
- To lock the first lock (lock 0, furthest left):
    - Generate a number known as the **ROOT**
    - Generate a number known as the **CN** by hashing the root**.** The **CN** is the combination of the lock and will unlock when displayed and button pressed
    - Generate a number known as the **LN** by hashing the **CN**. The **LN** is the number a lock displays to represent it is locked (i.e., any number but the **CN**)
    - Generate a number known as the **HN** by hashing the **LN**
- To lock subsequent locks (locks 1, 2, 3, and 4)
    - The **HN** of the lock to the left is used to generate a **CN** for the current lock
    - Generate the **LN** for the current lock from the **CN**
    - Generate the **HN** for the current lock from the **LN**
- Generating the **CN** uses the same hash function for all locks and is known as the *unlock hash function* (**UHF**)
- Generating the **LN** uses the same hash function for all locks and is known as the *lock hash function* **(LHF)**
- Generating the **HN** uses the same hash function for all locks and is known as the *pass hash function* (**PHF**)

Hash function rules

- In our multi-lock system a hash function applies a four digit number to four dials of a combination lock to derive another number. A positive number will turn a dial downwards and a negative number will turn a dial upwards. For example:
    - A hash function applying [+1 -4 +2 +9] to [9 0 9 8] would result in the new value [0 6 1 7]

Combination rules

- A **CN** must:
    - Not have repeating numbers
- BONUS MULTI_SAFE (only if everything else complete)
    - Ensure the sum of the digits on the combination lock to the left is less than the sum of the digits of the combination lock to the right
    - Ensure that the sum of numbers across all locks is even

Your program

- Create a program that will use randomisation to determine appropriate four digit inputs for each hash function and the **ROOT** value to generate a valid multi-lock configuration (the hash function inputs should be the same for all multi-lock safes but the number for the root should be different each time).
- Produce a text file outlining solutions known as the *key file*. Identify the number of solutions in the file and then list each solution to include the **ROOT** value followed by the four hash function inputs. This is an example of how you need to structure the file (this is not a valid solution but is used to illustrate file format):

NS 2
ROOT 6789
UHF +1,-4,+2,+5
LHF -4,+5,+2,-1
PHF -2,+3,+2,-3
ROOT 6749
UHF +1,-2,+2,+9
LHF -6,+5,+7,-1
PHF -1,+3,+2,-8

- Ensure your program can read in a *key file* and generate the equivalent **CN**, **LN** and **HN** values and output these in a text file (known as the *multi-safe file*) and identify if each entry is or is not a valid output given the requirements. Each solution should have 5 outputs for CN, LN and HN (one for each combination lock). This is an example of how you need to structure the file (this is not a valid solution but is used to illustrate the file format):

NS1 NOT VALID
CN0 7892, LN0 3446, HN0 1259,
CN1 7892, LN1 3456, HN1 3269
CN2 7292, LN2 3556, HN2 3759
CN3 9892, LN3 3056, HN3 7259
CN4 7892, LN4 3406, HN4 3959

Deliverables (what we want to see submitted):

- C++ source code authored by the student
- Executable file containing solution
- A *key file* containing 100 valid solutions (you may want to only output correct ones)
- A *multi-safe file* validating the solutions in your *key file*
- On Friday 26th October from 10am onwards students demonstrate solutions

Learning Outcomes (what we expect you to demonstrate in a general way)

- Be capable of designing and creating programs
- Realise inappropriate/appropriate usage of programming languages
- Understand how to manage memory
- To be able to create and use data structures
- To be able to use condition statements, loops and functions

Marking Scheme (what is worth what)

Marks are out of 25 and are awarded as follows:

- 10 Marks for achieving correct output
- 5 Marks for appropriate file input and output
- 3 Marks for user interface design
- 2 Marks for BONUS MULTI_SAFE
- 5 Marks for adherence to the 7 rules of programming (see lecture 1)

Additional direction (making the task clearer)

- **Task 1**: Construct a computer program that will generate a single combination lock output in a non-random way (**ROOT**, **CN**, **LN**, **HN**). Try to ensure that you can type in 4 numbers: 1 **ROOT** and the 3 numbers as input to the hash functions **UHF**, **LHF**, **PHF** and then generate the appropriate **CN**, **LN** and **HN** for a single combination lock
- **Task 2**: Extend your computer program to allow the outputs to be generated by randomisation (instead of typing the numbers in as you did in task 1) and include a suitable command line interface to make life easier
- **Task 3**: Extend your program to determine if the combination lock is showing a valid **CN** or not
- **Task 4**: Extend your program so you can decide how many iterations to run and capture valid iterations in a suitable data structure (and display them on screen) using an enhancement to your command line interface
- **Task 5**: Extend your program to a multi-safe version (with 5 locks)
- **Task 6**: Extend your program to allow validation of the outputs regarding the requirements of the multi-safe
- **Task 7**: Extend your program to produce the appropriate *key file* and *multi-safe* file
- **Task 8:** Think of how to make your multi-lock safes as difficult as possible to unlock