Week 7 – Session 1: Programming Challenges

This week we will gain familiarity with memory management pointers and references in C++ and their use

Challenge 1: Warm up

Consider the following code snippet. This is an incorrect implementation, why?

```
19  using namespace std;
20
21  int* GetPointer();
22
23  int main(int argc, const char * argv[]) {
24
25      int* intPtr = GetPointer();
26
27      cout << *intPtr << endl;
28
29      return 0;
30  }
31
32  int* GetPointer() {
33      int x{ 5 };
34      return &x;
35
36  }
37
```

Figure 1: Incorrect implementation of returning a variable from a function

Task 1 Fix it FELIX! - duration 10

Fix the code sample above correcting the implementation issue.

Challenge 2: Stack growth – duration 30 minutes

Which way does the stack grow for data allocated on the stack? Up towards higher addresses or down towards lower addresses?

Write a program that finds out. To do this:

- Declare an array of 10 integers with values from [1 ... 10] on the stack.
- Declare four integer variables [a ... d] with values [1 ... 4]
- Write a function `PrintStackArrayAddresses()` that print out the array index, element, address in hex, address in decimal for each item in the array.

Hint: you can use the `std::dec` and `std::hex` stream output manipulators to output values in decimal or hex as required

Challenge 3: Heap growth – duration 20 minutes

Repeat challenge 2 for the heap. This time dynamically allocate 10 floats on the heap. The values should be 1.1, 2.2, 3.3, …, 10.10.

- You should have a function `SetFloatValues(float* floatArray)` that passes in a float pointer to the array of elements and uses the pointer to initialise each value
- `PrintHeapArrayAddresses(float* floatArray)` should print out the index, element, address in hex, address in decimal for each item in the array.

Challenge 4: More raw pointers

Task 1 Simple input - duration 10 minutes

Write a function called `int HowMany()` that asks the user how many integer values they would like to record and returns the value.

Task 2 Allocating memory off the heap - duration 5 minutes

Write a function `int * CreateInts(int num)` that allocates memory for an array of integers corresponding to the size entered by the user. The function should return a pointer to the array.

Note: Remember to delete the array that you allocated off the heap!

Task 3 Using heap - duration 5 minutes

Write a function `ReadInts(int* arrayInt, int arrIdx)` that asks the user for integer values for the array and writes them to the memory locations reserved in the `CreateInts()` function.

Task 4 Display ints - duration 10 minutes

Write a function `DisplayInts(int* arrayInt, int arrIdx)` that prints the integer values of the array.

Task 5 Create a find highest function - duration 10 minutes

Write a function `FindMaxInts(int* arrayInt, int arrIdx)` that prints the highest integer value in the array.

Challenge 5: Unique pointers

Task 1 Initialise unique pointers - duration 10 minutes

Initialise a unique pointer `uniquePtr1` with a value of 32. Print out:

- The value stored in the memory location the pointer points to
- The address of the unique pointer
- The address of the memory location the pointer points to

Task 2 Changing ownership - duration 5 minutes

Now initialise a second unique pointer `uniquePtr2` and assign it ownership of the memory location where 32 is stored.

Task 3 Print out the details - duration 5 minutes

Print out the details for each pointer as above in task 1. Note what happens when you attempt to dereference pointer 1.

Challenge 6: Shared pointers

Task 1 Initialise a shared pointer - duration 5 minutes

Declare a shared pointer (`sharedPointerA`) that points to a new string. Read input from the user and place it into the string for the shared pointer.

Task 2 Print shared pointer details - duration 10 minutes

Print the following:

- The address (&) of the shared pointer
- The address of the memory location the pointer points to (&*)
- The value (*) of the shared pointer
- The reference count of the shared pointer

Task 3 Sharing is fun - duration 5 minutes

Declare a 2nd and 3rd shared pointers and point them at A. Also print their details.

Task 4 Sweet release - duration 5 minutes

Free `sharedPointerA` from the responsibility of managing the string. Print out the details of the other pointers including the reference count.

Challenge 7: Vectors

Task 1 Vectors - duration 20 minutes

Write an application that populates two vectors v1 and v2 with 100 randomly generated integers within the range [0 … 1000]. You should pass the vectors to a function called:
`PopulateVector(vector <int> &v, int size();`

Task 2 size and capacity - duration 5 minutes

Print out the size and capacity of the vectors before and after they have been populated.

Task 3 Sort the vectors - duration 10 minutes

Sort the contents of each vector in ascending order. Hint, you don't have to do this manually. Google it!

Task 4 Vector memory management - duration 5 minutes

You should note that after populating each vector the capacity increases beyond the number of actual elements. The vector class implementation will dynamically allocate a little more memory that it actually needs. Find out how to resize the vectors back to the size required for the number of elements actually contained in each vector.

Task 5 Vector memory management - duration 30 minutes

This task has little to do with memory management but will help hone your programming and problem-solving skills. Modify vector 2 so that it contains only numbers not found in vector 1.