Name: Xuan Tuan Minh Nguyen
Student ID: 103819212

1.1P: Preparing for OOP – Answer Sheet

1. Explain the following terminal instructions:
   a. cd: Change directory to the designated destination

   b. ls: List the current files and child directories inside a directories

   c. pwd: Extract the full path of the current directory or the place where the directory stays

2. Consider the following kinds of information, and suggest the most appropriate data type to store or represent each:

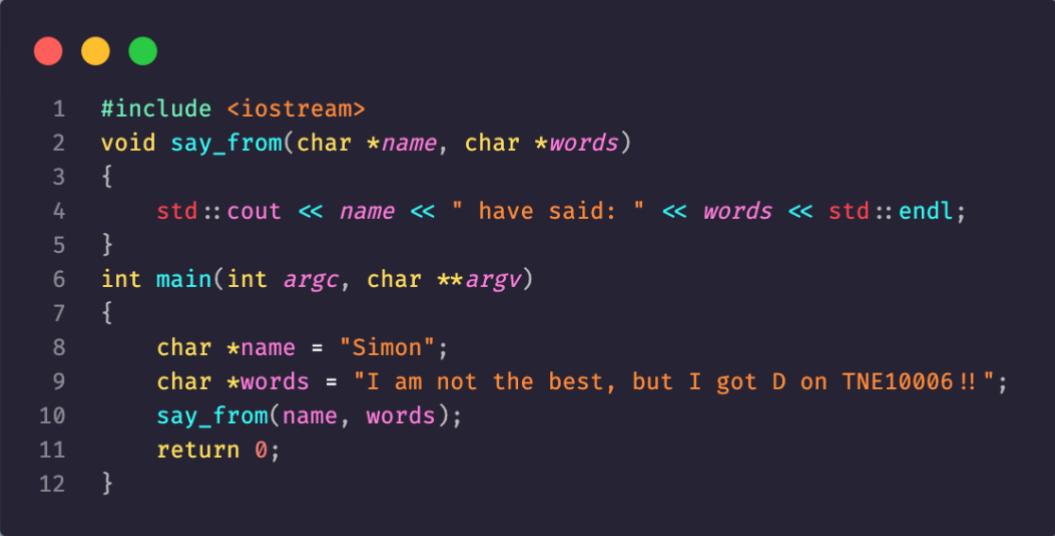| Information	Suggested Data Type
A person's name	String
A person's age in years	Integer
A phone number	String
A temperature in Celsius	Float
The average age of a group of people	Float
Whether a person has eaten lunch	Boolean

3. Aside from the examples already provided in question 2, come up with an example of information that could be stored as:

Data type	Suggested Information
String	An animal's information
Integer	A person's age
Float	The GPA of a Swinburne student
Boolean	Is his or her passed the TNE10006

4. Fill out the last two columns of the following table, evaluating the value of each expression and identifying the data type the value is most likely to be:

Expression	Given	Value	Data Type
6		6	Integer
True		True	Boolean
a	a = 2.5	2.5	Integer
1 + 2 * 3		7	Integer
a and False	a = True	False	Boolean
a or False	a = True	True	Boolean
a + b	a = 1 b = 2	3	Integer
2 * a	a = 3	6	Integer
a * 2 + b	a = 2.5 b = 2	7	Integer
a + 2 * b	a = 2.5 b = 2	6.5	Float
(a + b) * c	a = 1 b = 1 c = 5	10	Integer
"Fred" + " Smith"		Fred Smith	String
a + " Smith"	a = "Wilma"	Wilma Smith	String

5. Using an example, explain the difference between **declaring** and **initialising** a variable.
   - The difference between **declaring** and **initialising** is:
     - **Declaring**: Use for define which kind of data type (Float, Boolean, etc.) for the variable.
     - **Example**: public int x;
     - **Initialising**: Use for assign an initialise value for the variable
     - **Example**: x = 5;

6. Explain the term **parameter**. Write some code that demonstrates a simple of use of a parameter. You should show a procedure or function that uses a parameter, and how you would call that procedure or function.

   - A parameter or the parameters, is defined as variables that could be used for passing values into a function or a procedure

```cpp
#include <iostream>
void say_from(char *name, char *words)
{
    std::cout << name << " have said: " << words << std::endl;
}
int main(int argc, char **argv)
{
    char *name = "Simon";
    char *words = "I am not the best, but I got D on TNE10006 !! ";
    say_from(name, words);
    return 0;
}
```

Demonstration code for task 6

7. Using an example, describe the term **scope** as it is used in procedural programming (not in business or project management). Make sure you explain the different kinds of scope.
   - Scope is use for defining the region where variables can be accessed or referenced.
   - There are two kinds of scope
     - **Global scope** is a variable that could be used globally, in this case all functions or methods are able to access and referenced to that variable

```cpp
1    // Global variable
2    int x = 5;
3
4    int main(int argc, char **argv)
5    {
6        std::cout << x;
7        return 0;
8    }
```
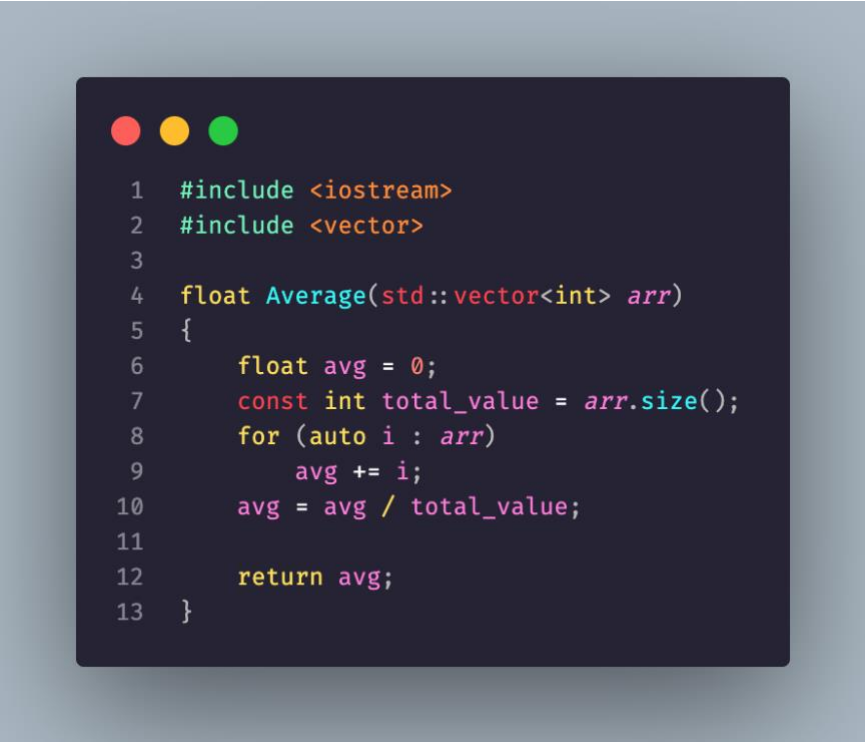
Example for Global Scope

   - **Local scope** is a variable that could only be used locally, which is a variable inside a functions or a method should only be used in that method or function.

```cpp
void function()
{
    // Local variable;
    int x = 5;
}

int main(int argc, char **argv)
{
    // Error
    std::cout << x;
    return 0;
}
```

Example for Local Scope

8. In a procedural style, in any language you like, write a function called Average, which accepts an array of integers and returns the average of those integers. Do not use any libraries for calculating the average. You must demonstrate appropriate use of parameters, returning and assigning values, and use of a loop. Note — just write the function at this point, we'll *use* it in the next task. You shouldn't have a complete program or even code that outputs anything yet at the end of this question.

```cpp
1    #include <iostream>
2    #include <vector>
3
4    float Average(std::vector<int> arr)
5    {
6        float avg = 0;
7        const int total_value = arr.size();
8        for (auto i : arr)
9            avg += i;
10       avg = avg / total_value;
11
12       return avg;
13   }
```

Demonstration code for task 8

9. In the same language, write the code you would need to call that function and print out the result.

```cpp
#include <functional>
float test_average(std::function<float(std::vector<int>)> arg_func, std::vector<int> arr)
{
    return arg_func(arr);
}

int main(int argc, char **argv)
{
    std::vector<int> arr{2, 5, 4, 2, 3};
    std::cout << test_average(&Average, arr);
    return 0;
}
```

Demonstration code for task 9

```
> g++-13  -o function function.cpp
> ./function
3.2%
```

Output for above code

10. To the code from 9, add code to print the message "Double digits" if the average is above or equal to 10. Otherwise, print the message "Single digits". Provide a screenshot of your program running.

```cpp
1   #include <iostream>
2   #include <vector>
3   #include <functional>
4
5   float Average(std::vector<int> arr)
6   {
7       float avg = 0;
8       const int total_value = arr.size();
9       for (auto i : arr)
10          avg += i;
11      avg = avg / total_value;
12      return avg;
13  }
14
15  void disalay_digits(std::function<float(std::vector<int>)> arg_func, std::vector<int> arr)
16  {
17      if (arg_func(arr) >= 10)
18          std::cout << "Double digits" << std::endl;
19      else
20          std::cout << "Single digits" << std::endl;
21  }
22
23  float test_average(std::function<float(std::vector<int>)> arg_func, std::vector<int> arr)
24  {
25      return arg_func(arr);
26  }
27
28  int main(int argc, char **argv)
29  {
30      std::vector<int> first_arr{50, 30, 80, 40, 20};
31      std::vector<int> second_arr{2, 3, 4, 5, 6};
32      std::cout << test_average(&Average, first_arr) << std::endl;
33      disalay_digits(&Average, first_arr);
34      std::cout << test_average(&Average, second_arr) << std::endl;
35      disalay_digits(&Average, second_arr);
36
37      return 0;
38  }
39
```

Demonstration code for task 10

```
❯ g++-13  -o function function.cpp
❯ ./function
44
Double digits
4
Single digits
```