SOFTWARE DEVELOPMENT

Assignment 1

ZEYNEB ARIF
1813730
UNIVERSITY OF WALES TRINITY SAINT DAVID
Cloud Computing

Table of Contents

Part A....2
Constant....2
Variable....2
Data Types....3
Value Type....3
Reference Type....3
Dynamic Type....3
Part B....4
Part C....8
Task 1....8
Task 2....9
Task 3....10
References....12

Figure 1 Constant Example....2
Figure 2 Menu....4
Figure 3 Main....4
Figure 4 Display Header....5
Figure 5 Display Menu....5
Figure 6 Read Number....5
Figure 7 Process Menu....6
Figure 8 Get Numbers....6
Figure 9 Find Smallest....6
Figure 10 Find Largest....7
Figure 11 Is Quitting....7
Figure 12 Write Text....7
Figure 13 Clear Text....7
Figure 14 Task 1 Code....8
Figure 15 Task 1 Screenshot....8
Figure 16 Task 2 Code....9
Figure 17 Task 2 Screenshot....9
Figure 18 Task 3 Code....10
Figure 19 Task 3 Screenshot....11

Part A

Constant

Constants are static properties defined at the time of initialization, which do not change the program's existence. Constants are declared with the modifier constant. Only the built-in types of C # (except System. Object) can be declared a const. User-defined types can't be constant, including classes, structures, and arrays. Using the read only modifier to build a file, or list that is initialized once at runtime (e.g. in a function Object() { [native code] }), which can' t be modified afterwards. (Microsoft, 2020)

```
class Calendar1
{
    public const int Months = 12;
}
```

Figure 1 Constant Example

For this case, the constant months are still 12, and even the class itself can't change it. In addition, when the compiler finds a constant variable in C # source code (e.g. months), it replaces the literal value explicitly with the intermediate language (IL) tag it produces. As at runtime there is no variable address associated with a constant, constant fields can't be transferred by proxy, and can't exist in an expression as a l-value.

Variable

A vector is nothing but a name given to a storage area which can be exploited by our programs. -- variable in C # has a particular form, which specifies the size and configuration of the memory of the variable, the number of values that can be stored in that memory, as well as the collection of operations that can be applied to it.

Syntax for variable definition in C# is – `<data_type> <variable_list>;`

Data type must be a valid C # type of data, like char, int, float, double, or any user-defined type of data, and variable list that consist of one or more comma-separated identifier names.

Some valid definitions of the variable are given here –

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

Data Types

Value Type

Variables of the form of value can be applied explicitly to a variable. They derive from the System. ValueType class.

The forms of values include data straight away. Some examples are int, char, and float which respectively stores numbers, alphabets, and floating-point numbers. If you declare a type int, the program assigns memory to store the attribute. (TutorialsPoint, 2020)

You may use the scale approach to get the exact size of a category or element on a specified board. The size(type) statement yields the entity or type in bytes storage space. Following is an example for having the int sort size on every system.

```
using System;

namespace DataTypeApplication
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Size of int: {0}", sizeof(int));
            Console.ReadLine();
        }
    }
}
```

Reference Type

The reference types do not include the exact data contained in a variable, but a reference to the variables is given.

In other words, they refer to a place in the brain. The comparison types may refer to a memory location through the use of multiple variables. If one of the variables updates the data in the memory position the other variable will immediately represent this value adjustment. Examples of implemented reference types are: object, dynamic, and set. (TutorialsPoint, 2020)

```
object obj;
obj = 100; // this is boxing
```

Dynamic Type

Every kind of value can be contained in the dynamic data form variable. Type verification for these variables happens at run-time.

```
dynamic <variable_name> = value;
```

Part B

```
*****************************************************************************
* This application is designed to allow you to choose numbers *
* and finds the biggest and the smallest value *
*****************************************************************************



                    Select an option
                    1. Enter the numbers
                    2. Find the smallest
                    3. Find the largest
                    4. Display all numbers
                    5. Quit
                    Please choose an option: _
```

Figure 2 Menu

The first thing we see when we open the program is the menu with 5 options.

1. Lets us enter 3 numbers.
2. Shows us the smallest number.
3. Shows us the largest number.
4. Says that it should display the entered numbers but instead asks us to quit the program.
5. Asks us to quit the program.

```
const int MAXNUMBERS = 3;
0 references
static void Main(string[] args)
{
    int[] theNumbers = new int[MAXNUMBERS];
    int chosenOption = 0;
    bool quit = false;
    InitialiseNumbers(theNumbers);
    while (quit == false)
    {
        DisplayHeader();
        DisplayMenu();
        chosenOption = ReadNumber("Please choose an option: ");
        quit = ProcessMenu(chosenOption, theNumbers);
        Console.Clear();
    }
}
```

Figure 3 Main

The main method declares some of the variables and calls 4 functions:

```
static void DisplayHeader()
{
    WriteText("*****************************************************************************", 0, 0);
    WriteText("* This application is designed to allow you to choose numbers *", 0, 1);
    WriteText("* and finds the biggest and the smallest value *", 0, 2);
    WriteText("*****************************************************************************", 0, 3);
}
```

Figure 4 Display Header

DisplayHeader shows the header of the program on the console.

```
static void DisplayMenu()
{
    WriteText("Select an option", 20, 8);
    WriteText("1. Enter the numbers", 20, 9);
    WriteText("2. Find the smallest", 20, 10);
    WriteText("3. Find the largest", 20, 11);
    WriteText("4. Display all numbers", 20, 12);
    WriteText("5. Quit", 20, 13);
}
```

Figure 5 Display Menu

DisplayMenu shows the menu of the program on the console.

```
static int ReadNumber(string prompt)
{
    string text;
    int number;
    WriteText(prompt, 20, 14);
    text = Console.ReadLine();
    number = int.Parse(text);
    ClearText(20, 14, prompt.Length + text.Length);
    return number;
}
```

Figure 6 Read Number

ReadNumber returns the user input as a integer.

```
static bool ProcessMenu(int option, int[] numbers)
{
    bool quit = false;
    switch (option)
    {
        case 1:
            GetNumbers(numbers);
            break;
        case 2:
            WriteText(string.Format("The smallest value is {0}", FindSmallest(numbers)), 20, 15);
            Console.ReadKey(); // Pause
            break;
        case 3:
            WriteText(string.Format("The largest value is {0}", FindLargest(numbers)), 20, 15);
            Console.ReadKey(); // Pause
            break;
        case 4:
            quit = IsQuitting();
            break;
        case 5:
            quit = IsQuitting();
            break;
        default:
            WriteText("Unknown option value entered", 20, 15);
            Console.ReadKey(); // Pause
            break;
    }
    return quit;
}
```

Figure 7 Process Menu

ProcessMenu is the function that decides which option of the menu has been selected and executes the appropriate code. Here we can see why option 4 is asking us to quit the program.

```
static void GetNumbers(int[] numbers)
{
    for (int index = 0; index < MAXNUMBERS; index++)
    {
        numbers[index] = ReadNumber("Enter number: ");
    }
}
```

Figure 8 Get Numbers

GetNumbers asks the user 3 times to enter a number and stores the numbers into an array for later use.

```
static int FindSmallest(int[] numbers)
{
    int smallest = numbers[0];
    for (int index = 0; index < MAXNUMBERS - 1; index++)
    {
        if (numbers[index + 1] < smallest)
        {
            smallest = numbers[index + 1];
        }
    }
    return smallest;
}
```

Figure 9 Find Smallest

FindSmallest function checks every element of the array and returns the lowest number.

```
static int FindLargest(int[] numbers)
{
    int largest = numbers[0];
    for (int index = 0; index < MAXNUMBERS - 1; index++)
    {
        if (numbers[index + 1] > largest)
        {
            largest = numbers[index + 1];
        }
    }
    return largest;
}
```

Figure 10 Find Largest

FindLargest function checks every element of the array and returns the biggest number.

```
static bool IsQuitting()
{
    string response;
    bool quit = false;
    WriteText("Do you really wish to quit? ", 20, 13);
    response = Console.ReadLine();
    if (response == "YES")
    {
        quit = true;
    }
    return quit;
}
```

Figure 11 Is Quitting

IsQuitting function return a Boolean value of true or false depending on the user's decision to quit the program or not. That value is then used in the while loop in the main function to continue or break the loop.

```
static void WriteText(string text, int x, int y)
{
    Console.CursorLeft = x;
    Console.CursorTop = y;
    Console.Write(text);
}
```

Figure 12 Write Text

WriteText is used by the program to display the all text in console on a specified position.

```
static void ClearText(int x, int y, int length)
{
    WriteText(new String(' ', length), x, y);
}
```

Figure 13 Clear Text

ClearText is used to fill a portion of line with blank spaces.

Part C

Task 1.

```
using System;

namespace Assignment_1_Exercise_1
{
    0 references
    class MainClass
    {
        0 references
        public static void Main(string[] args)
        {
            int price;
            int quantity;
            Console.Write("Enter the name of the product: ");
            string name = Console.ReadLine();

        GetPrice:
            Console.Write("Enter the price of the product: ");
            if (!int.TryParse(Console.ReadLine(), out price))
            {
                Console.WriteLine("Invalid Input. Please try again.");
                goto GetPrice;
            }
        GetQuantity:
            Console.Write("Enter the quantity of the product: ");
            if (!int.TryParse(Console.ReadLine(), out quantity))
            {
                Console.WriteLine("Invalid Input. Please try again.");
                goto GetQuantity;
            }
            Console.WriteLine("The total price for {0} {1} is {2}£", quantity, name, price * quantity);
        }
    }
}
```

Figure 14 Task 1 Code

To solve the problem, we need to get from the user 3 variables and display the product of 2 of them on the console. Two of the inputs are type validated with a combination of a TryParse method and the use of labels.

```
Enter the name of the product: Apple
Enter the price of the product: asd
Invalid Input. Please try again.
Enter the price of the product: 2
Enter the quantity of the product: asd
Invalid Input. Please try again.
Enter the quantity of the product: 5
The total price for 5 Apple is 10£
```

Figure 15 Task 1 Screenshot

Task 2.

```
using System;

namespace Assignment_1_Exercise_2
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            int x;
            int y;
        GetX:
            Console.Write("Enter X: ");
            if (!int.TryParse(Console.ReadLine(), out x))
            {
                Console.WriteLine("Invalid Input. Please try again.");
                goto GetX;
            }
        GetY:
            Console.Write("Enter Y: ");
            if (!int.TryParse(Console.ReadLine(), out y))
            {
                Console.WriteLine("Invalid Input. Please try again.");
                goto GetY;
            }
            int result = ((x - 7) * 3) + ((y - 3) * (x - 7)) + 17;
            Console.WriteLine("The area is: {0}", result);
        }
    }
}
```

Figure 16 Task 2 Code

In this problem we need to solve the area of an odd shape when given just 2 parameters with the following formula: `((x - 7) * 3) + ((y - 3) * (x - 7)) + 17`. X and Y need to be integers so we can use TryParse and labels to ensure the correct data type.

```
Enter X: asd
Invalid Input. Please try again.
Enter X: 20
Enter Y: asd
Invalid Input. Please try again.
Enter Y: 15
The area is: 212
```

Figure 17 Task 2 Screenshot

Task 3.

```
using System;

namespace Assignment_1_Exercise_3
{
    0 references
    class Program
    {
        const int x = 30;
        const int y = 8;
        0 references
        static void Main(string[] args)
        {
            Random rnd = new Random();
            int score = 0;
            int win = 0;
            Console.SetCursorPosition(x-5, y-5);
            Console.WriteLine("Press enter to Spin. Type \"quit\" to quit the game");
            while (true)
            {
                int a = rnd.Next(0, 10);
                int b = rnd.Next(0, 10);
                int c = rnd.Next(0, 10);

                string input = Console.ReadLine();
                input.ToLower();

                if (input == "quit")
                {
                    break;
                }
                if (a == b && b == c && a == 9)
                {
                    win = 2000;
                    score += win;
                }
                else if ((a == b && a == 9) || (b == c && b == 9) || (a == c && a == 9))
                {
                    win = 200;
                    score += win;
                }
                else if (a == b && b == c)
                {
                    win = 15;
                    score += win;
                }
                else if (a == b || b == c || a == c)
                {
                    win = 5;
                    score += win;
                }
                else
                {
                    win = 0;
                }
                Console.SetCursorPosition(x, y+1);
                Console.WriteLine(new String(' ', 35));
                Console.SetCursorPosition(x, y+1);
                if (win == 0)
                {
                    Console.WriteLine("You didn't win anything. Try again.");
                }
                else
                {
                    Console.WriteLine("You win {0} points.", win);
                }
                Console.SetCursorPosition(x, y);
                Console.WriteLine("Numbers: {0}, {1}, {2}    Score: {3}", a, b, c, score);
                Console.SetCursorPosition(x, y+2);
            }
        }
```

Figure 18 Task 3 Code

For the slot machine game, we need 3 random independent numbers to be generated on every cycle, a score variable to store the score and a win variable streamline the logic. Using the idea from the assignment appendix to display the text in a different position we can create a user interface for our slot machine game displaying the random numbers, score, winnings and a prompt text.

```
Press enter to Spin. Type "quit" to quit the game


Numbers: 0, 3, 3    Score: 10
You win 5 points.
quit
```

Figure 19 Task 3 Screenshot

References

Microsoft, 2020. *Constants.* [Online]
Available at: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constants

TutorialsPoint, 2020. *CSharp Data Types.* [Online]
Available at: https://www.tutorialspoint.com/csharp/csharp_data_types.htm