Lab 02

Basic I/O, Variables, Data Types, and Operators

Muhammad Mujtaba

CMD ID: 540040

mmujtaba.bese25seecs@seecs.edu.pk

Class: BESE 16B

Batch: 2k25

Task 1 [CLO 1]:

CODE:

```
#include <iostream>
#include <conio.h>

int main()
{
    // storing a constant for 8 stars `*`
    const char* STARS = "********";

    // printing the required pattern
    std::cout << STARS << "\t" << STARS << "\t" << STARS << "\t" << STARS << "\n";
    std::cout << "\t" << STARS << "\t" << STARS << "\t" << STARS << "\t" << STARS << "\n";
    std::cout << STARS << "\t" << STARS << "\t" << STARS << "\t" << STARS << "\n";

    // use to pause termination so we can see the output
    _getch();
    return 0;
}
```

OUTPUT:

```
c:\users\mmujtaba.bese25see

********        ********        ********        ********
        ********        ********        ********        ********
********        ********        ********        ********
```

Task 2 [CLO 1]:

CODE:

```
#include <iostream>
#include <conio.h>

// defining helper function to input a function
// var is taken by reference (&) and updated in the function and no value is
returned
void getValue(const char* varName, double& var)
{
    std::cout << "Enter " << varName << ": ";
    std::cin >> var;
    // next line will be inserted due to cin
    // std::cout << std::endl;
}

int main()
{
    // defining variables
    double r;
    double a;
    double b;
    double c;
    double d;

    // getting the values through input
    getValue("r", r);
    getValue("a", a);
    getValue("b", b);
    getValue("c", c);
    getValue("d", d);

    // calculating expression
    double answer = 4 / (3 * (r + 34)); // bracket is used after `/` to divide `4`
by `(3 * (r + 34))` as WHOLE
    // in `(3 * (r + 34))` brackets are used around `r + 34` so that `3 * r` is not
calculated first
    answer -= 9 * (a + b * c);
    // here `b * c` is calculated first and then added to a due to operator
precedence
```

```
    answer += (3 + d * (2 + a)) / (a + b*d);
    // brackets are used around `/` to remove operator precedence
    // on numerator `d * (2 + a)` is calculated first and then added to three
    // in denominator `b * d` is calculated first and then added to `a`

    std::cout << "Answer: " << answer;

    // pausing termination
    _getch();
    return 0;
}
```

OUTPUT:

```
c:\users\mmujtaba.bese25see
Enter r: 1
Enter a: 1
Enter b: 1
Enter c: 1
Enter d: 1
Answer: -14.9619
```

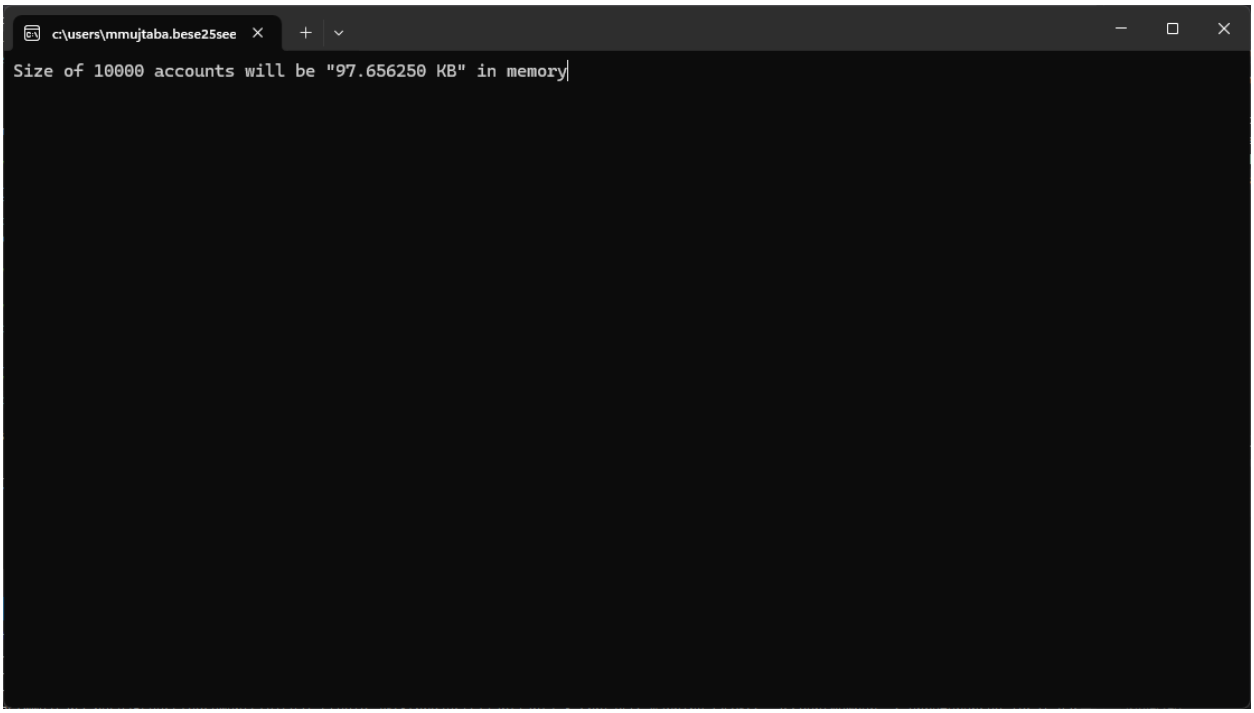
```
c:\users\mmujtaba.bese25see
Enter r: 6.12
Enter a: 5
Enter b: 3
Enter c: 4.5
Enter d: 6
Answer: -164.51
```

Task 3 [CLO 3]:

CODE:

```
#include <iostream>
#include <conio.h>
#include <string>

// using std::string to return formatted string, returns as 123 KB; 123 MB; 123 bytes
std::string getSizeString(int bytes)
{
    // more than 1 KB
    if (bytes > 1024)
    {
        if (bytes > (1024 * 1024)) // 1 MB = 1024 * 1 KB = 1024 * 1024 bytes
            return std::to_string(bytes / (1024.0 * 1024.0)) + " MB"; // .0 to invoke implicit
conversion
        else
            return std::to_string(bytes / 1024.0) + " KB";
    }
    else
        return std::to_string(bytes) + " bytes";
}

int main()
{
    // declaring variables
    int accountNumber;
    float transactionAmount;
    char accountStatus;
    bool activeStatus;
    // no initializing needed as memory has already been allocated
    // calculating size for one account
    const int accountSize_bytes = sizeof(accountNumber) + sizeof(transactionAmount) +
sizeof(accountStatus) + sizeof(activeStatus);

    int datasetCount = 10000;
    // calculating size for 10 000 accounts
    const int datasetSize_bytes = accountSize_bytes * datasetCount;

    std::cout << "Size of 10000 accounts will be \"" << getSizeString(datasetSize_bytes) <<
"\" in memory\n\n\n";

    _getch();
    return 0;
}
```

OUTPUT:


c:\users\mmujtaba.bese25see
Size of 10000 accounts will be "97.656250 KB" in memory

Task 4 [CLO 3]:

CODE:

```
#include <iostream>
#include <bitset>
#include <string>
#include <cstdint>
#include <iomanip>
#include <conio.h>

// defining helper function to input a function
// var is taken by reference (&) and updated in the function and no value is
returned
void getValue(const char* varName, uint8_t& outVar)
{
    int inputVar;
    // using do while loop to make sure user enters number between 0 and 255
    // `uint8_t` can only hold values from 0 and 255 inlcusive
    do
    {
        std::cout << "Input value for \"" << varName << "\" between 0 and 255: ";
        std::cin >> inputVar;
        // stopping the program from going crazy when alphabet is entered
        if (std::cin.fail()) // if error is detected in cin
        {
            std::cin.clear(); // clear error flag
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); //
discard the input
            // ^ says discard all input including newline character
        }
    } while (inputVar < 0 || inputVar > 255);
    outVar = inputVar; // updating var value
    // no checks needed as value is confirmed to be between 0 and 255
    // next line will be inserted due to cin
    // std::cout << std::endl;
}

// inline utility to print
inline void printBits(const std::string& prefix, const uint8_t& val)
{
    std::cout << std::left << std::setw(16) << prefix << std::bitset<8>(val) <<
"\n";
}
```

```
int main()
{
    // `uint8_t` is an unsigned int with 8 bits
    uint8_t firstNum;
    uint8_t secondNum;

    getValue("A", firstNum);
    getValue("B", secondNum);

    printBits("BITs A: ", firstNum);
    printBits("BITs B: ", secondNum);
    printBits("BITs A & B: ", (firstNum & secondNum));
    printBits("BITs A | B: ", (firstNum | secondNum));
    printBits("BITs A ^ B: ", (firstNum ^ secondNum));
    printBits("BITs A << 2: ", (firstNum << 2));
    printBits("BITs B >> 3: ", (secondNum >> 3));


    _getch();
    return 0;
}
```

OUTPUT:

```
c:\users\mmujtaba.bese25see
Input value for "A" between 0 and 255: 25
Input value for "B" between 0 and 255: 5
BITs A:         00011001
BITs B:         00000101
BITs A & B:     00000001
BITs A | B:     00011101
BITs A ^ B:     00011100
BITs A << 2:    01100100
BITs B >> 3:    00000000
```

```
c:\users\mmujtaba.bese25see
Input value for "A" between 0 and 255: a
Input value for "A" between 0 and 255: 300
Input value for "A" between 0 and 255: 3
Input value for "B" between 0 and 255: 255
BITs A:         00000011
BITs B:         11111111
BITs A & B:     00000011
BITs A | B:     11111111
BITs A ^ B:     11111100
BITs A << 2:    00001100
BITs B >> 3:    00011111
```