Monirul Islam Mahmud
ID: A22286395
Project: 02

The source code can be found here: https://github.com/Monirules/Programming-Language.git

The source code meets all the requirements below:

- **No mutable variables**

The code uses only immutable lists and functional constructs such as, *List.map, List.map2, List.fold2,* and *recursion*. No mutable keyword or imperative mutation is present.

- **Looping is done by tail recursion**

Functions such as *toBin, addHelper,* and *askInputType* use recursion instead of traditional loops. toBin used in *unsignedToBin* and addHelper used in *addBinaryLists* are implemented in a tail-recursive manner.

- **All Functions Work on Binary Lists**

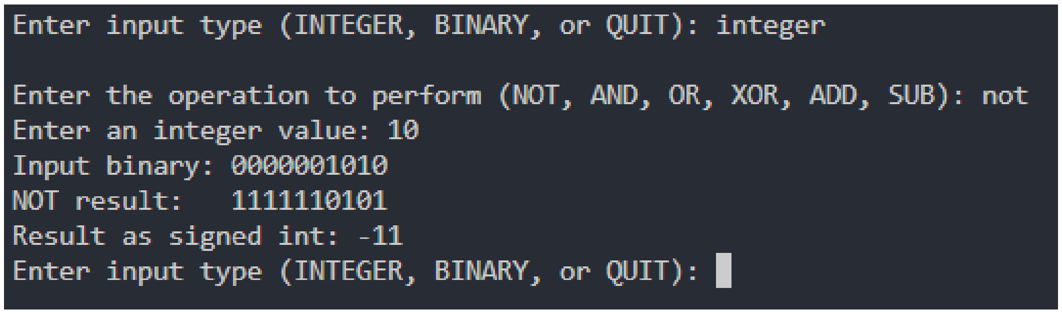
The primary data structure is an *int list*, representing binary numbers. Operations like, *notBinary*, *addBinaryLists, twosComplement*, and bitwise operations - *andBinary, orBinary, xorBinary* all work on binary lists.

- **Conversions to and from integer to binary lists follow shared algorithms**

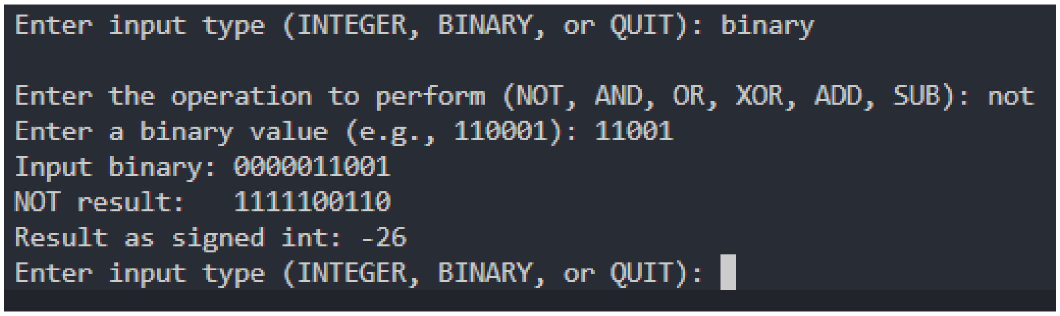
unsignedToBin and *signedToBin* correctly convert integers to binary lists. *binToSignedInt* accurately converts binary lists back to integers, handling two's complement representation. The logic follows standard binary conversion principles.

Example Output

1. **NOT Operation**

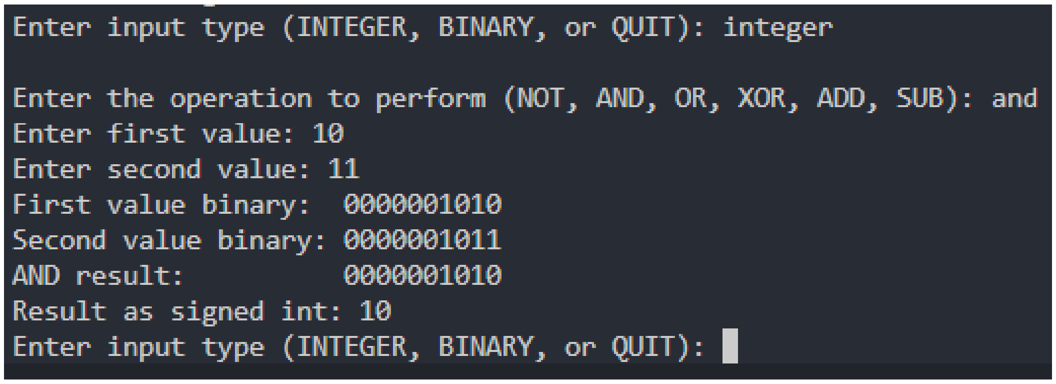


```
Enter input type (INTEGER, BINARY, or QUIT): integer

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): not
Enter an integer value: 10
Input binary: 0000001010
NOT result:   1111110101
Result as signed int: -11
Enter input type (INTEGER, BINARY, or QUIT):
```

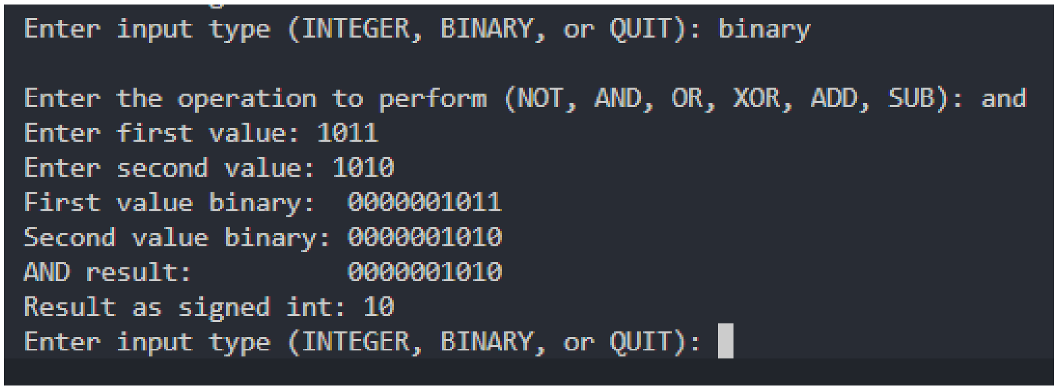


```
Enter input type (INTEGER, BINARY, or QUIT): binary

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): not
Enter a binary value (e.g., 110001): 11001
Input binary: 0000011001
NOT result:   1111100110
Result as signed int: -26
Enter input type (INTEGER, BINARY, or QUIT):
```

2. **AND Operation**

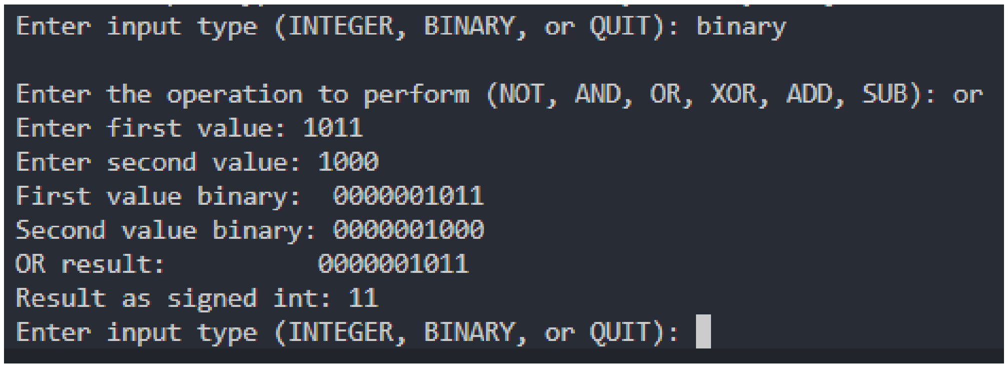


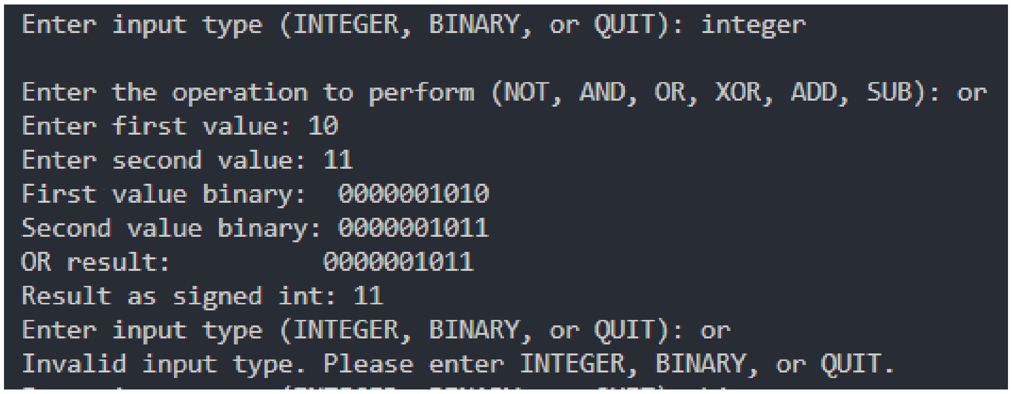
```
Enter input type (INTEGER, BINARY, or QUIT): integer

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): and
Enter first value: 10
Enter second value: 11
First value binary:  0000001010
Second value binary: 0000001011
AND result:          0000001010
Result as signed int: 10
Enter input type (INTEGER, BINARY, or QUIT):
```



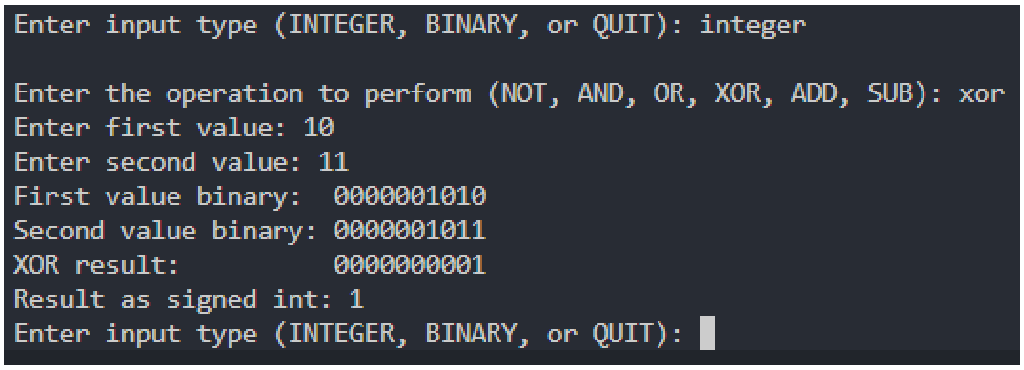
```
Enter input type (INTEGER, BINARY, or QUIT): binary

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): and
Enter first value: 1011
Enter second value: 1010
First value binary:  0000001011
Second value binary: 0000001010
AND result:          0000001010
Result as signed int: 10
Enter input type (INTEGER, BINARY, or QUIT):
```

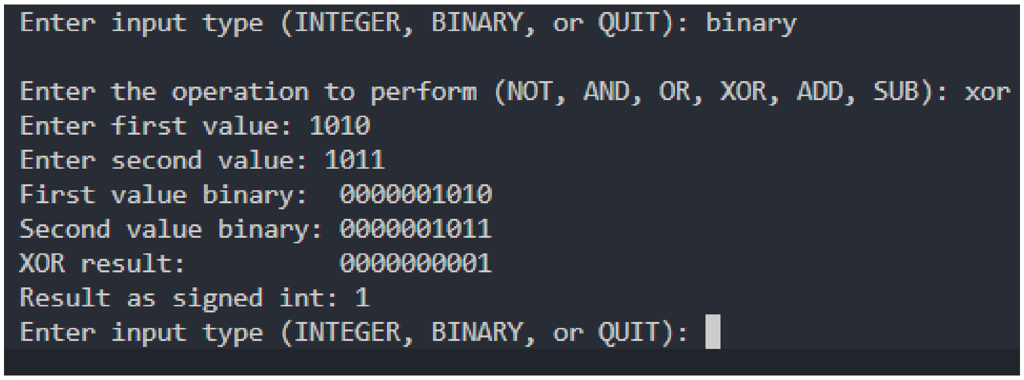
3. **OR Operation**


```
Enter input type (INTEGER, BINARY, or QUIT): binary

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): or
Enter first value: 1011
Enter second value: 1000
First value binary:  0000001011
Second value binary: 0000001000
OR result:          0000001011
Result as signed int: 11
Enter input type (INTEGER, BINARY, or QUIT):
```

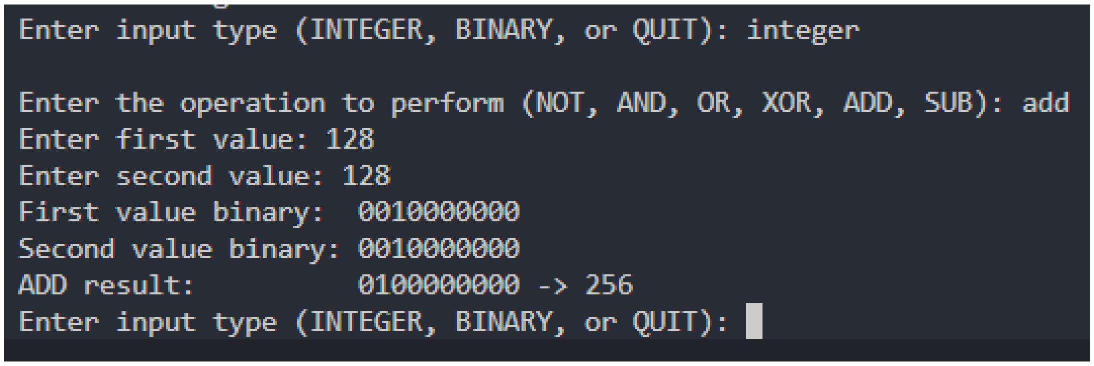

```
Enter input type (INTEGER, BINARY, or QUIT): integer

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): or
Enter first value: 10
Enter second value: 11
First value binary:  0000001010
Second value binary: 0000001011
OR result:          0000001011
Result as signed int: 11
Enter input type (INTEGER, BINARY, or QUIT): or
Invalid input type. Please enter INTEGER, BINARY, or QUIT.
```

4. **XOR Operation**

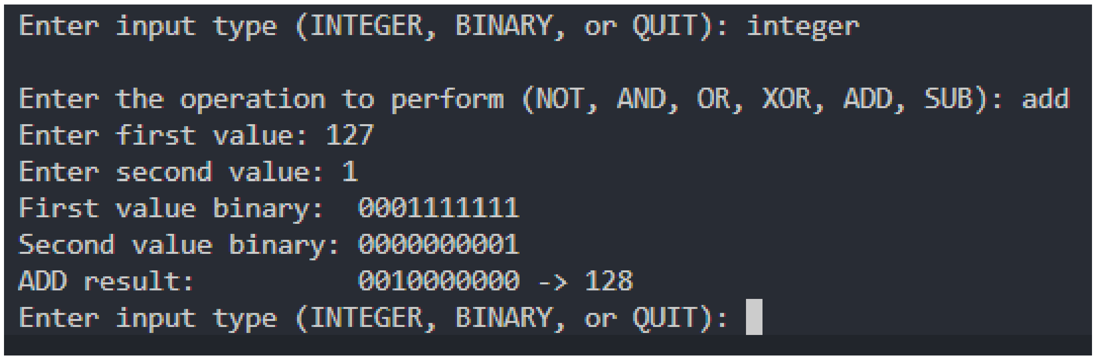

```
Enter input type (INTEGER, BINARY, or QUIT): integer

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): xor
Enter first value: 10
Enter second value: 11
First value binary:  0000001010
Second value binary: 0000001011
XOR result:          0000000001
Result as signed int: 1
Enter input type (INTEGER, BINARY, or QUIT):
```


```
Enter input type (INTEGER, BINARY, or QUIT): binary

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): xor
Enter first value: 1010
Enter second value: 1011
First value binary:  0000001010
Second value binary: 0000001011
XOR result:          0000000001
Result as signed int: 1
Enter input type (INTEGER, BINARY, or QUIT):
```

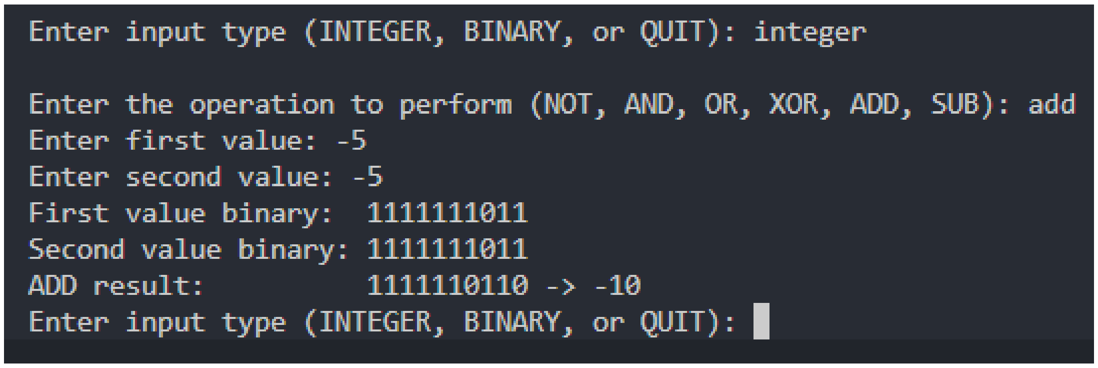
5. **ADD Operation**



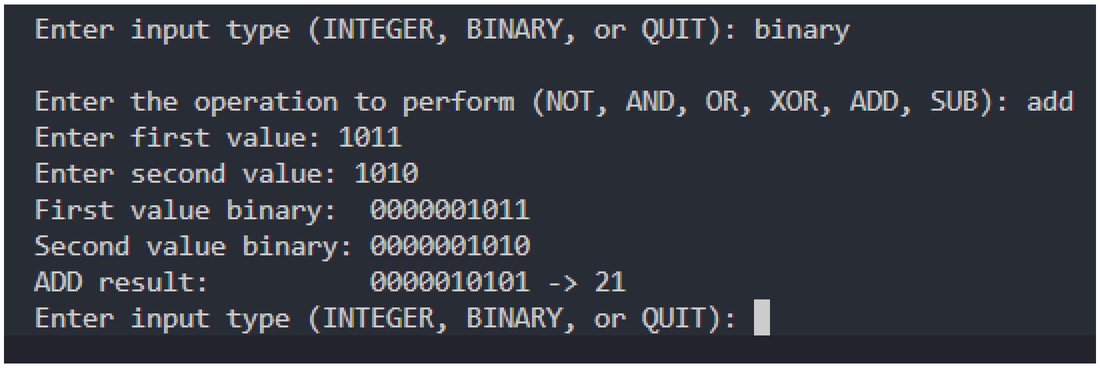
```
Enter input type (INTEGER, BINARY, or QUIT): integer

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): add
Enter first value: 128
Enter second value: 128
First value binary:  0010000000
Second value binary: 0010000000
ADD result:          0100000000 -> 256
Enter input type (INTEGER, BINARY, or QUIT):
```



```
Enter input type (INTEGER, BINARY, or QUIT): integer

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): add
Enter first value: 127
Enter second value: 1
First value binary:  0001111111
Second value binary: 0000000001
ADD result:          0010000000 -> 128
Enter input type (INTEGER, BINARY, or QUIT):
```

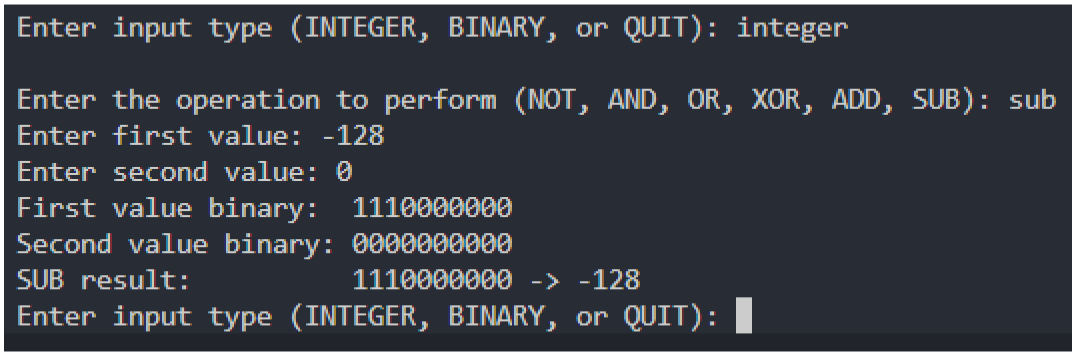


```
Enter input type (INTEGER, BINARY, or QUIT): integer

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): add
Enter first value: -5
Enter second value: -5
First value binary:  1111111011
Second value binary: 1111111011
ADD result:          1111110110 -> -10
Enter input type (INTEGER, BINARY, or QUIT):
```

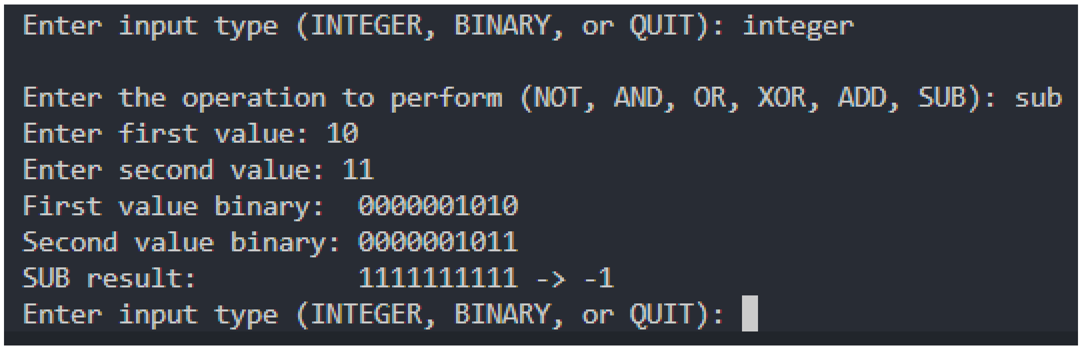


```
Enter input type (INTEGER, BINARY, or QUIT): binary

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): add
Enter first value: 1011
Enter second value: 1010
First value binary:  0000001011
Second value binary: 0000001010
ADD result:          0000010101 -> 21
Enter input type (INTEGER, BINARY, or QUIT):
```

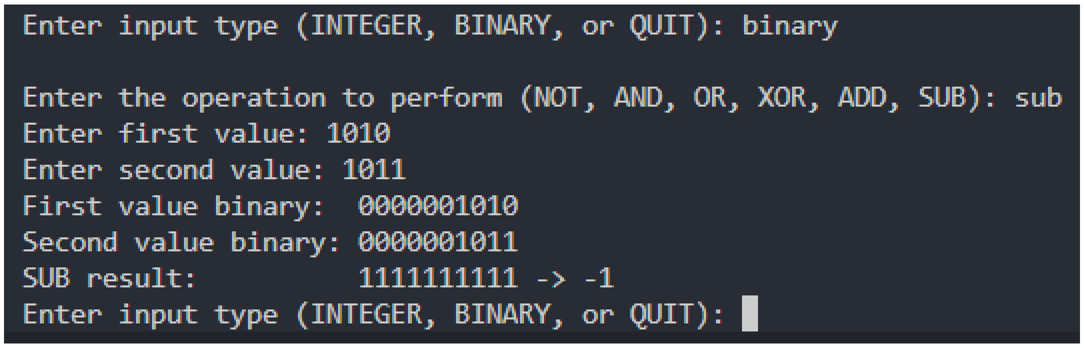
6. SUB Operation



```
Enter input type (INTEGER, BINARY, or QUIT): integer

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): sub
Enter first value: -128
Enter second value: 0
First value binary:  1110000000
Second value binary: 0000000000
SUB result:          1110000000 -> -128
Enter input type (INTEGER, BINARY, or QUIT):
```



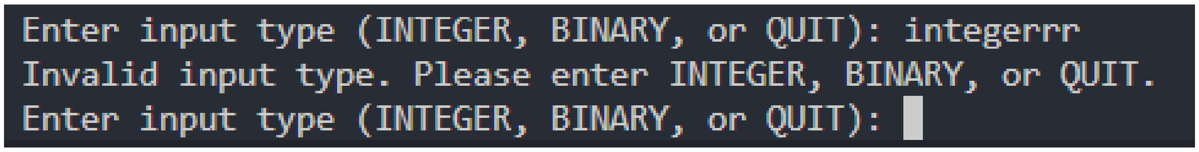
```
Enter input type (INTEGER, BINARY, or QUIT): integer

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): sub
Enter first value: 10
Enter second value: 11
First value binary:  0000001010
Second value binary: 0000001011
SUB result:          1111111111 -> -1
Enter input type (INTEGER, BINARY, or QUIT):
```



```
Enter input type (INTEGER, BINARY, or QUIT): binary

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): sub
Enter first value: 1010
Enter second value: 1011
First value binary:  0000001010
Second value binary: 0000001011
SUB result:          1111111111 -> -1
Enter input type (INTEGER, BINARY, or QUIT):
```

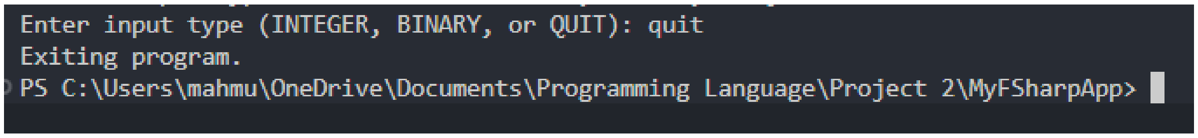
Handling User Errors

- **Handling misspelling while choosing binary or integer value**



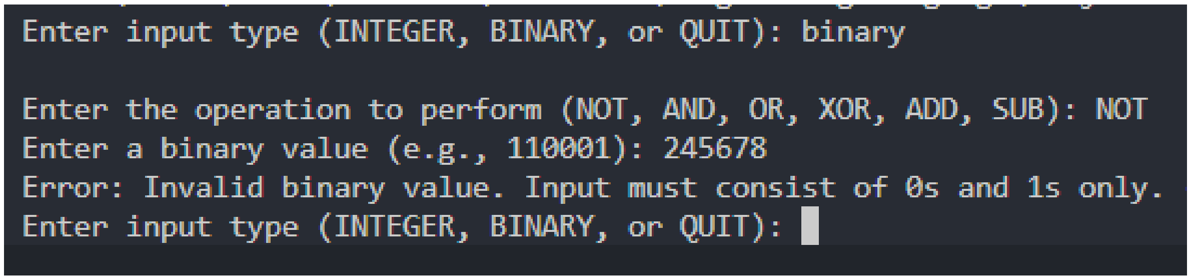
```
Enter input type (INTEGER, BINARY, or QUIT): integerrr
Invalid input type. Please enter INTEGER, BINARY, or QUIT.
Enter input type (INTEGER, BINARY, or QUIT):
```

- **QUIT operation will exit the program**



```
Enter input type (INTEGER, BINARY, or QUIT): quit
Exiting program.
PS C:\Users\mahmu\OneDrive\Documents\Programming Language\Project 2\MyFSharpApp>
```

- **Handling errors in binary values**



```
Enter input type (INTEGER, BINARY, or QUIT): binary

Enter the operation to perform (NOT, AND, OR, XOR, ADD, SUB): NOT
Enter a binary value (e.g., 110001): 245678
Error: Invalid binary value. Input must consist of 0s and 1s only.
Enter input type (INTEGER, BINARY, or QUIT):
```

5 Key Points of this project

1. The code adheres to functional programming principles, using only immutable variables and tail recursion for looping.
2. It correctly implements binary list operations, including bitwise (NOT, AND, OR, XOR), arithmetic (ADD, SUB), and two's complement representation for signed integers.
3. The functions *unsignedToBin* and *signedToBin* efficiently convert integers to binary lists, while *binToSignedInt* accurately reconstructs signed integers from binary.
4. Functions like *addBinaryLists* and *subtractBinaryLists* use tail recursion to perform binary addition and subtraction efficiently without stack overflow issues.
5. Most importantly, The code handles errors in binary input using *parseBinary* to check if a string contains only *0s* and *1s*, and safely handles exceptions in conversions and operations using *try-catch*.