Basic documentation for Digital Integrated Circuits console simulator

The Digital Integrated Circuits console simulator is a modular and flexible application designed for defining and analyzing logical circuits. Its structured memory management and namespace segregation ensure scalability and reliability.

1. How to run the code on Windows:

A. Ensure that a C++ compiler is installed (GCC via MinGW, or MSVC).
B. Open a terminal or Command Prompt and navigate to the directory containing the source files.
C. Compile the program using:

```
g++ -o simulator.exe main.cpp -std=c++17
```

D. Run the compiled program:

```
simulator.exe
```

2. How to run the code on Linux

A. Install a C++ compiler such as GCC.
B. Open a terminal and navigate to the directory with the source code.
C. Compile the program using:

```
g++ -o simulator main.cpp -std=c++17
```

D. Execute the program:

```
./simulator
```

3. Download the project's archive from MNKnowledge and use VS

A. Download and unarchive the .zip file
B. Open the project (*StrypesLab2024_FinalProject.sln*) at Visual Studio and run the file *main.cpp*

4. Main code architecture

The code is structured into multiple namespaces and modules to manage complexity:

A. Namespaces:
   a. *ProjectConstants*: Defines constants used across the application.
   b. *CustomizedString* and *CustomizedVector*: Custom data structures for managing arrays of characters and integers, respectively.
   c. *CustomizedStack*: Implements a stack for logical expression evaluation.

B. Modules:
   a. *IntegratedCircuit*: Handles creation, deletion, and validation of circuits.
   b. *IntegratedCircuitInput*: Handles creation and manipulation of user’s input.
   c. *IntegratedCircuitStorage*: Provides a container for created integrated circuits.
   d. *MemoryManagement*: Provides functions for dynamic memory manipulation.
   e. *TruthTable*: Handles creation, deletion and manipulation of truth tables.
   f. *LogicalExpressionHandler*: Manages logic expression parsing and evaluation.
   g. *Main* & *Utils*: Provides the entry point and basic user interface.

5. Fundamental functions and their logic

A. *readIntegratedCircuit()*
   a. Reads and creates an integrated circuit from the input stream.
   b. Parses the name of the circuit and its arguments.
   c. Reads the logical expression and parses it into tokens.
   d. Validates the expression against the arguments using *validateIntegratedCircuit()*.

B. *parseCommandRUN()*
   a. Parses RUN command containing the name of the circuit and specific argument values.

C. *convertInfixToPostfix()*
   a. Infix logical expression conversion to postfix using the Shunting Yard algorithm for computation simplification.

D. *postfixExpressionEvaluation()*
   a. Computes the value of a logical expression in postfix form using a stack. It supports &(AND), |(OR), and !(NOT) operators.

E. *executeCommandRUN()*

   a. Takes a circuit and specific input values, substitutes arguments with the values, and calculates the result.
F. *executeCommandFIND()*
   a. Calculates a logical expression for a circuit represented by a truth table. It uses logical synthesis by "1".
G. *executeCommandALL()*
   a. Generates all input-output combinations for a circuit represented in its complete truth table.

6. Memory management

Memory is managed using:

A. Custom functions such as *allocArrayMemory*, *freeArrayMemory*, *deleteCircuitStorage* etc. for dynamic memory allocation and deallocation.
B. Array resizing method like *reallocArrayMemory* for dynamic array's size modifications.

7. Main program

The main program (*main.cpp*) implements a command-driven interface. Supported commands include:

A. **DEFINE**: Creates a logic integrated circuit with a given name and parameters.
B. **RUN**: Executes any of defined integrated circuits with defined input values.
C. **ALL**: Enables execution and analysis of an integrated circuit in a truth table.
D. **FIND**: Allows to find an integrated circuit by a given truth table.
E. **PRINT**: Prints all defined integrated circuits in the storage.
F. **EXIT**: Allows to stop the program and exit.

8. Sample Command Flow

A. *DEFINE* a Circuit:

```
DEFINE A(x,y) "(x & y)"
```

a. Defines a circuit named *A* with inputs *x* and *y* and the logical expression *(x & y)*.
b. Steps:
   1) Parses circuit name (*A*) and arguments (*x*, *y*).
   2) Tokenizes the logical expression into (*x* & *y*).
   3) Validates the logical expression for correctness - ensures all tokens match arguments.
   4) Checks if a circuit with the same name already exists in the CircuitStorage.
   5) Adds the circuit to storage if validation passes.

B. *RUN* a Circuit:

```
RUN A(1,0)
```

a. Executes circuit *A* with *x=1* and *y=0*.
b. Steps:
   1) Retrieves circuit *A* from storage.
   2) Replaces *x* and *y* in the expression with input values: *(x & y) → (1 & 0)*
   3) Evaluates the expression: *Result = 0*

C. *ALL* Analysis:

```
ALL A
```

Analyzes all possible input combinations for circuit *A*.

D. *FIND* by Truth Table:

```
FIND "truth_table.txt"
```

Searches for the file named “truth_table.txt” and generates a logical expression using the table defined in the file.

9. Additions

A. Error Handling:
   a. Assertions ensure invalid operations (e.g., accessing empty arrays or invalid tokens) are caught during runtime.
   b. Input validation ensures malformed commands are reported to the user.

B. Scalability:
   a. The use of dynamic memory allocation supports variable-sized inputs and outputs, enhancing flexibility.

C. Extensibility:

a. Adding new logical operations or data structures can be achieved with minimal modifications due to the modular code design.

10. Example inputs, outputs, and expected behavior

COMMAND	DESCRIPTION	SAMPLE INPUT	EXPECTED OUTPUT
DEFINE	Define a new circuit with a logical expression.	*DEFINE A(x,y) "(x & y)"*	Next command input or error message
RUN	Execute a circuit with specific input values.	*RUN A(1,0)*	Result: 0
ALL	Analyze all combinations of a circuit's inputs.	*ALL A*	Table with all combinations of specific inputs and outputs for relevant integrated circuit.
FIND	Search for circuits matching a given truth table.	*FIND "truth_table.txt"*	Find the file and do the operations or error message
PRINT	Display all defined circuits.	*PRINT*	List of all defined circuits
EXIT	Exit the simulator.	*EXIT*	Program terminates

11. Input format for each command

Here's a detailed description of the exact input format expected by each command in the simulator:

A. *DEFINE*: Define a new circuit

a. Command Syntax:

```
DEFINE <Name>(<Argument1>,<Argument2>,...,<ArgumentN>) "<LogicalExpression>"
```

b. Input Details

1) *<Name>*: A single alphanumeric character representing the circuit name.
2) *<ArgumentX>*: One or more unique alphanumeric characters representing input variables.
3) *<LogicalExpression>*: A valid logical expression using:
   - Logical operators: *& (AND)*, *| (OR)*, *! (NOT)*.
   - Parentheses for precedence: *()*.
   - Arguments must match the defined *<ArgumentX>* list.

c. Constraints:
   1) *<Name>* must be unique and cannot already exist in storage.
   2) *<LogicalExpression>* must use only the listed arguments.

B. *RUN*: Execute a circuit with specific inputs
   a. Command Syntax:

```
RUN <Name>(<Value1>,<Value2>,...,<ValueN>)
```

   b. Input Details:
      1) *<Name>*: The name of an existing circuit.
      2) *<ValueX>*: Binary values (0 or 1) matching the number of arguments defined for the circuit.
   c. Constraints:
      1) *<Name>* must exist in storage.
      2) The number of input values must match the number of arguments defined for the circuit.

C. *ALL*: Analyze all input-output combinations for a circuit
   a. Command Syntax:

```
ALL <Name>
```

   b. Input Details:
      1) *<Name>*: The name of an existing circuit.
   c. Constraints:
      1) *<Name>* must exist in storage.

D. *FIND*: Find circuits matching a truth table
   a. Command Syntax:

```
FIND "<FilePath>"
```

   b. Input Details:
      1) *<FilePath>*: The path to a file containing the truth table in the following matrix format:

```
<Value11> <Value12> ... <Value1N>
...
<ValueM1> <ValueM2> ... <ValueMN>
```

   c. Constraints:
      1) File must be accessible and correctly formatted.

E. *PRINT*: Display all defined circuits
   a. Command Syntax:

```
PRINT
```

b. Input Details:
   1) No additional input required.

F. *EXIT*: Terminate the simulator
   a. Command Syntax:

```
EXIT
```

   b. Input Details:
      1) No additional input required.

12. Handling errors and invalid inputs

A. **Error Case 1:** *DEFINE* with Invalid Expression
   a. Command:

```
DEFINE B(x, y) "(x & z)"
```

   b. Error Description:
      1) Expression contains *z*, which is not a defined argument.
      2) Validation fails, and the circuit is rejected.
   c. Output:

```
Found NOT valid token {z}.
Invalid expression entered. Skip DEFINE command.
```

B. **Error Case 2:** *RUN* Nonexistent Circuit
   a. Command:

```
RUN C(1, 0)
```

   b. Error Description:
      1) Circuit *C* is not found in storage.
   c. Output:

```
Digital Integrated Circuit 'C' does NOT exist. Instead, skip RUN
command or DEFINE the circuit.
```

C. Error Case 3: *FIND* with Missing File
   a. Command:

```
FIND "missing_file.txt"
```

   b. Error Description:

        1) The specified truth table file does not exist or cannot be opened.
    c. Output:

```
Failed to parse truth table from file 'missing_file.txt'. Check if the file name is not wrong or if the file is missing. Skip 'FIND' command.
```

D. Error Case 4: *ALL* for Nonexistent Circuit
    a. Command:

```
ALL D
```

    b. Error Description:
        1) Circuit *D* is not found in storage.
    c. Output:

```
Digital Integrated Circuit 'D' does NOT exist. Instead, skip ALL command or DEFINE the circuit.
```