Project Documentation

System name:	Boolean Expressions (main)
Students:	Prinsloo, H.F.
	10022806

Declaration:

We declare that this is our own, original work and that we understand what plagiarism entails according to the policy of the University of Pretoria. We also declare that this is the only instance of this project that each of us are working on.

____
Prinsloo H.F.
10022806

____
Date

Description of Project:

This project's main purpose is to evaluate and display Boolean algebraic expressions through the use of logic gates. It has a user-friendly menu navigation system which allows the user to be in total control and be fully aware of what he/she is doing at any specific time. The user may enter any expression by making use of the predefined keywords. This expression is analysed and a tree structure thereof is created. The user may then choose to get the result of the expression, change the values of variables inside the expression or view the expression in three different ways (Boolean algebra, Tree structure or Truth table). The program also has the capability to store and load external functions to text files.

Phases Attempted:

Phase 0 – Getting started

Phase 1 – Complex Boolean Expressions

Phase 2 – Different Views

Phase 3 – Predefined Boolean Operators

Phase 4 – File Input/output

Phase 5 – Component Circuits and elementary simulation

Phase 6 – Only partially as it is only a console navigation system

Phase 0 – Getting started

In phase 0 we implemented the Adapter design by using encapsulation to convert the return type of the functions used in the original BooleanOperation class from string to Boolean.

Adapter Design Pattern

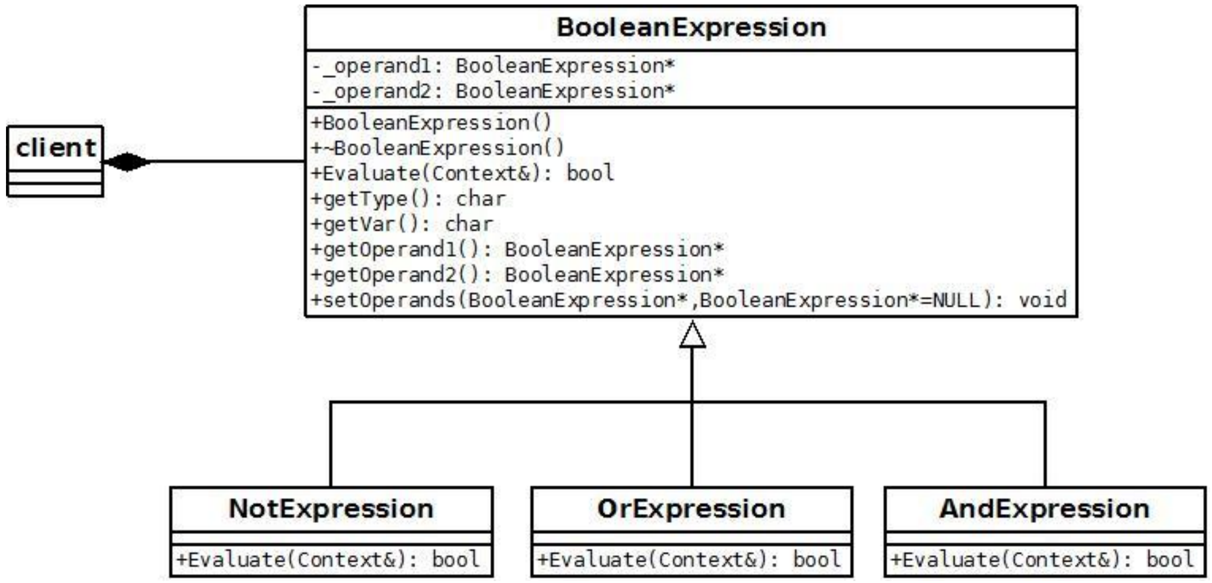
Intent: "Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."[0]

Prototype Design Pattern

We then created prototypes from which we can create Boolean expressions.

AndExpression	**&&**
OrExpression	**||**
NotExpression	**!**

Intent: "Define the skeleton of an algorithm in an operation, deferring some steps to client subclasses."[0]

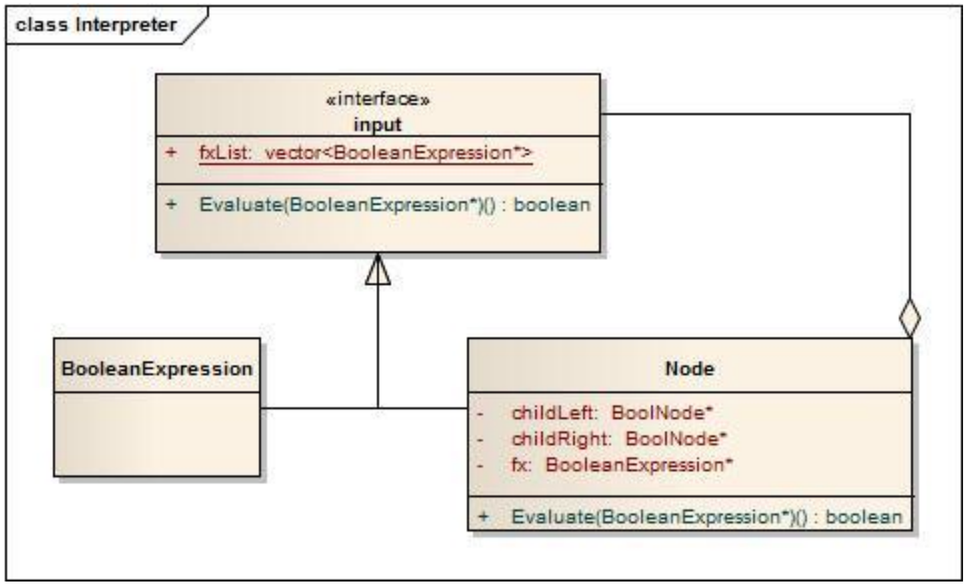

BooleanExpression
-_operand1: BooleanExpression*
-_operand2: BooleanExpression*
+BooleanExpression()
+~BooleanExpression()
+Evaluate(Context&): bool
+getType(): char
+getVar(): char
+getOperand1(): BooleanExpression*
+getOperand2(): BooleanExpression*
+setOperands(BooleanExpression*,BooleanExpression*=NULL): void
client
NotExpression
+Evaluate(Context&): bool
OrExpression
+Evaluate(Context&): bool
AndExpression
+Evaluate(Context&): bool

Phase 1 – Complex Boolean Expressions

Here we enabled the construction of more complex Boolean expressions by making use of more than one gate and also gives you the ability to make use of variables (**W**,**X**,**Y** and **Z**). These variables can also be set to desired values.

Interpreter

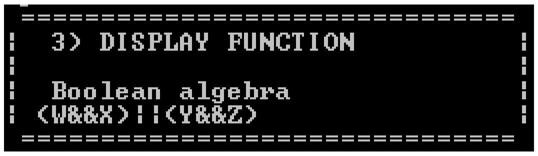
Intent: "Compose objects into tree structures to represent whole-part hierarchies. Composite lets clients treat individual objects and composition of objects uniformly."[0]



Phase 2 – Different views

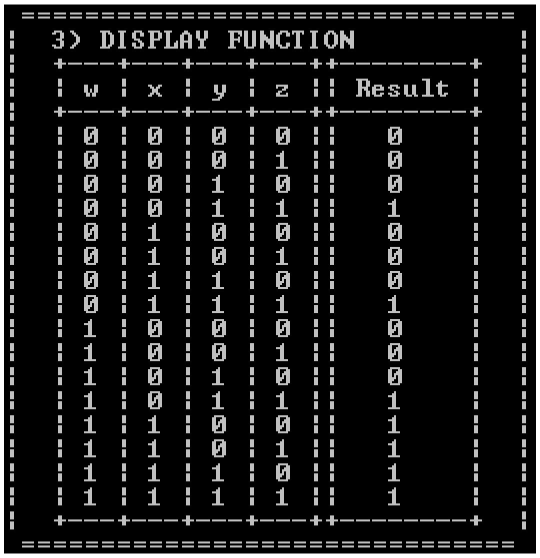
We now enabled the program to display the Boolean expression in two different ways.

-Boolean algebra



```
 ==============================
¦  3> DISPLAY FUNCTION         ¦
¦                              ¦
¦  Boolean algebra             ¦
¦ (W&&X)¦¦(Y&&Z)               ¦
 ==============================
```

-Truth Table



```
 ==============================
¦  3> DISPLAY FUNCTION         ¦
¦   +---+---+---+---++--------+ ¦
¦   ¦ w ¦ x ¦ y ¦ z ¦¦ Result ¦ ¦
¦   +---+---+---+---++--------+ ¦
¦   ¦ 0 ¦ 0 ¦ 0 ¦ 0 ¦¦   0    ¦ ¦
¦   ¦ 0 ¦ 0 ¦ 0 ¦ 1 ¦¦   0    ¦ ¦
¦   ¦ 0 ¦ 0 ¦ 1 ¦ 0 ¦¦   0    ¦ ¦
¦   ¦ 0 ¦ 0 ¦ 1 ¦ 1 ¦¦   1    ¦ ¦
¦   ¦ 0 ¦ 1 ¦ 0 ¦ 0 ¦¦   0    ¦ ¦
¦   ¦ 0 ¦ 1 ¦ 0 ¦ 1 ¦¦   0    ¦ ¦
¦   ¦ 0 ¦ 1 ¦ 1 ¦ 0 ¦¦   0    ¦ ¦
¦   ¦ 0 ¦ 1 ¦ 1 ¦ 1 ¦¦   1    ¦ ¦
¦   ¦ 1 ¦ 0 ¦ 0 ¦ 0 ¦¦   0    ¦ ¦
¦   ¦ 1 ¦ 0 ¦ 0 ¦ 1 ¦¦   0    ¦ ¦
¦   ¦ 1 ¦ 0 ¦ 1 ¦ 0 ¦¦   0    ¦ ¦
¦   ¦ 1 ¦ 0 ¦ 1 ¦ 1 ¦¦   1    ¦ ¦
¦   ¦ 1 ¦ 1 ¦ 0 ¦ 0 ¦¦   1    ¦ ¦
¦   ¦ 1 ¦ 1 ¦ 0 ¦ 1 ¦¦   1    ¦ ¦
¦   ¦ 1 ¦ 1 ¦ 1 ¦ 0 ¦¦   1    ¦ ¦
¦   ¦ 1 ¦ 1 ¦ 1 ¦ 1 ¦¦   1    ¦ ¦
¦   +---+---+---+---++--------+ ¦
 ==============================
```

Strategy

Intent: "Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it."[0]

Bridge

Intent: "Decouple an abstraction from its implementation so that the two can vary independently."[0]

Observer

Intent: "Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically."[0]

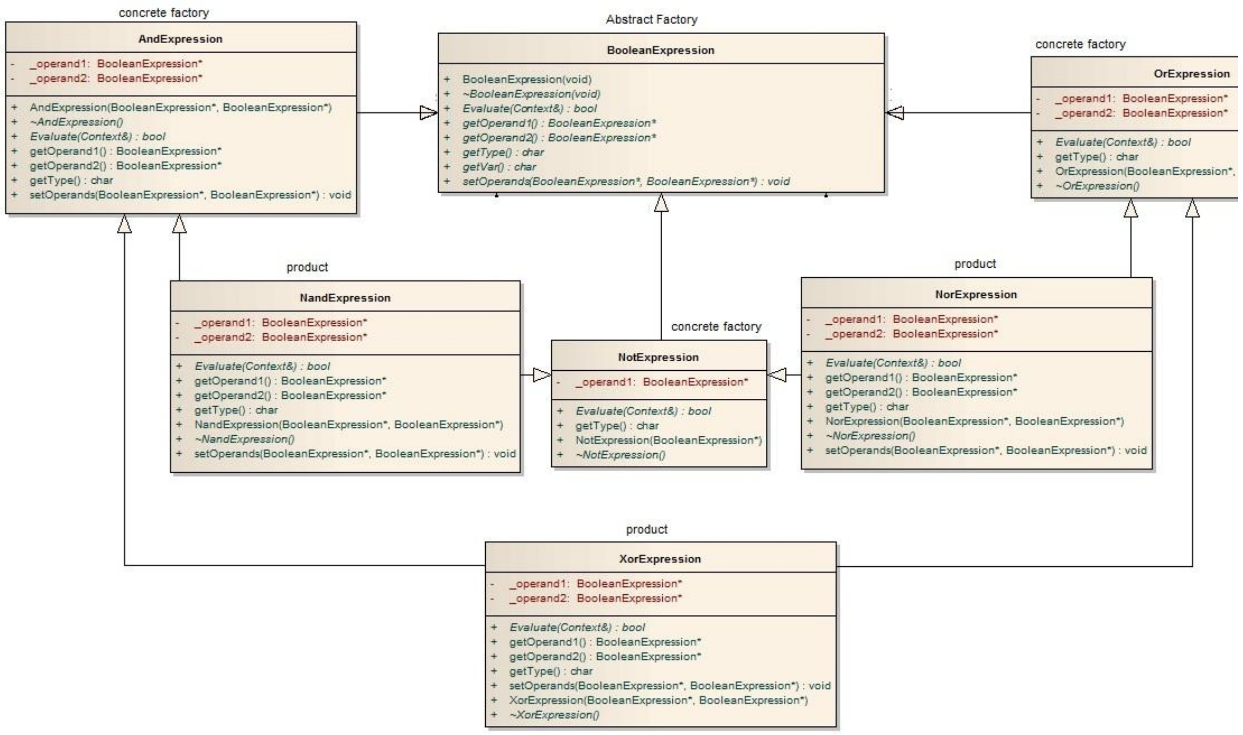
Phase 3 – Predefined Boolean Operators

We used the functions in phase 0 to construct more predefined Boolean gates.

XOR	**UU**
NAND	**$$**
NOR	**^^**
EQUIVALENCE	**==**

Abstract Factory Design Pattern

Intent: "Provide an interface for creating families of related or dependent objects without specifying their concrete classes."[0]



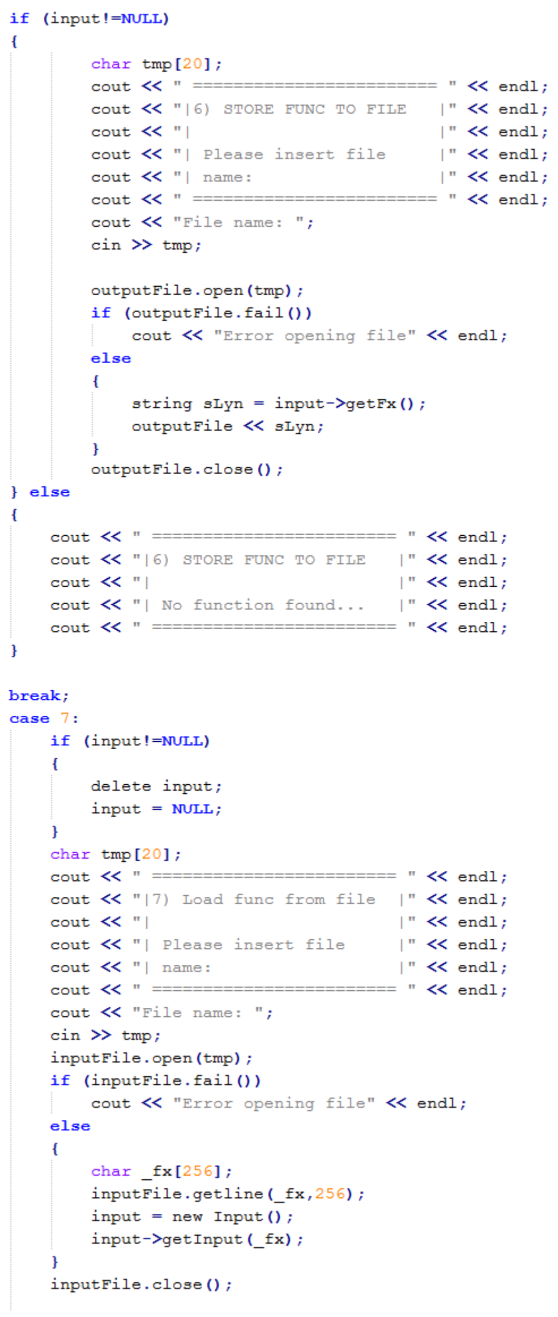
concrete factory
AndExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ AndExpression(BooleanExpression*, BooleanExpression*)
+ ~AndExpression()
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ setOperands(BooleanExpression*, BooleanExpression*) : void
Abstract Factory
BooleanExpression
+ BooleanExpression(void)
+ ~BooleanExpression(void)
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ getVar() : char
+ setOperands(BooleanExpression*, BooleanExpression*) : void
concrete factory
OrExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ Evaluate(Context&) : bool
+ getType() : char
+ OrExpression(BooleanExpression*,
+ ~OrExpression()
product
NandExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ NandExpression(BooleanExpression*, BooleanExpression*)
+ ~NandExpression()
+ setOperands(BooleanExpression*, BooleanExpression*) : void
concrete factory
NotExpression
- _operand1: BooleanExpression*
+ Evaluate(Context&) : bool
+ getType() : char
+ NotExpression(BooleanExpression*)
+ ~NotExpression()
product
NorExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ NorExpression(BooleanExpression*, BooleanExpression*)
+ ~NorExpression()
+ setOperands(BooleanExpression*, BooleanExpression*) : void
product
XorExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ setOperands(BooleanExpression*, BooleanExpression*) : void
+ XorExpression(BooleanExpression*, BooleanExpression*)
+ ~XorExpression()

Prototype

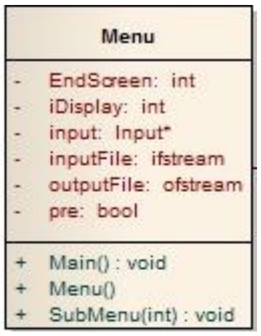
Intent: "Specify kinds of objects to create using a prototypical instance, and create new objects by copying this prototype."[0]

Phase 4 – File Input/Output

We now allowed the user to store and load files to and from text files. The function in Boolean algebraic infix notation gets stored to the text file the user specifies. When the user loads a function, the user specified file is read and the program is manipulated as to take the input as if it was done manually.

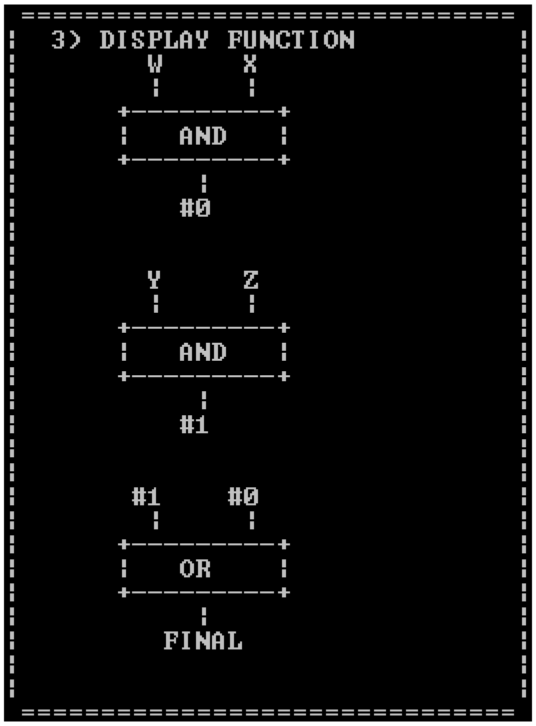


```
if (input!=NULL)
{
        char tmp[20];
        cout << " ======================== " << endl;
        cout << "|6) STORE FUNC TO FILE    |" << endl;
        cout << "|                         |" << endl;
        cout << "| Please insert file      |" << endl;
        cout << "| name:                   |" << endl;
        cout << " ======================== " << endl;
        cout << "File name: ";
        cin >> tmp;

        outputFile.open(tmp);
        if (outputFile.fail())
            cout << "Error opening file" << endl;
        else
        {
            string sLyn = input->getFx();
            outputFile << sLyn;
        }
        outputFile.close();
} else
{
    cout << " ======================== " << endl;
    cout << "|6) STORE FUNC TO FILE    |" << endl;
    cout << "|                         |" << endl;
    cout << "| No function found...    |" << endl;
    cout << " ======================== " << endl;
}

break;
case 7:
    if (input!=NULL)
    {
        delete input;
        input = NULL;
    }
    char tmp[20];
    cout << " ======================== " << endl;
    cout << "|7) Load func from file   |" << endl;
    cout << "|                         |" << endl;
    cout << "| Please insert file      |" << endl;
    cout << "| name:                   |" << endl;
    cout << " ======================== " << endl;
    cout << "File name: ";
    cin >> tmp;
    inputFile.open(tmp);
    if (inputFile.fail())
        cout << "Error opening file" << endl;
    else
    {
        char _fx[256];
        inputFile.getline(_fx,256);
        input = new Input();
        input->getInput(_fx);
    }
    inputFile.close();
```



Phase 5 – Component Circuits and elementary simulation

The system can now display any function in circuit view.

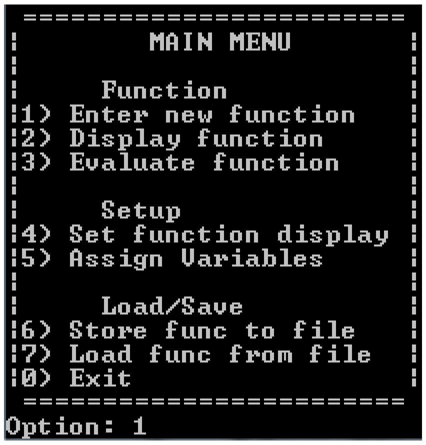


```
 ===============================
¦  3) DISPLAY FUNCTION          ¦
¦        W     X                ¦
¦        ¦     ¦                ¦
¦      +----------+             ¦
¦      ¦   AND    ¦             ¦
¦      +----------+             ¦
¦           ¦                   ¦
¦          #0                   ¦
¦                               ¦
¦                               ¦
¦        Y     Z                ¦
¦        ¦     ¦                ¦
¦      +----------+             ¦
¦      ¦   AND    ¦             ¦
¦      +----------+             ¦
¦           ¦                   ¦
¦          #1                   ¦
¦                               ¦
¦                               ¦
¦       #1    #0                ¦
¦        ¦     ¦                ¦
¦      +----------+             ¦
¦      ¦   OR     ¦             ¦
¦      +----------+             ¦
¦           ¦                   ¦
¦         FINAL                 ¦
¦                               ¦
¦                               ¦
 ===============================
```

Phase 6 – Only partially as it is only a console navigation system

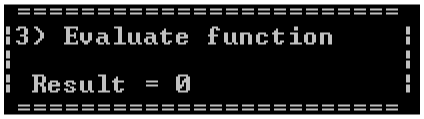
Print screens of menus:

Main menu:



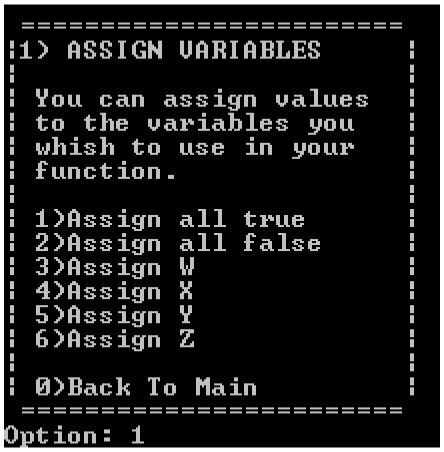
```
 ========================
|        MAIN MENU       |
|                        |
|      Function          |
|1) Enter new function   |
|2) Display function     |
|3) Evaluate function    |
|                        |
|      Setup             |
|4) Set function display |
|5) Assign Variables     |
|                        |
|      Load/Save         |
|6) Store func to file   |
|7) Load func from file  |
|0) Exit                 |
 ========================
Option: 1
```

Evaluate:



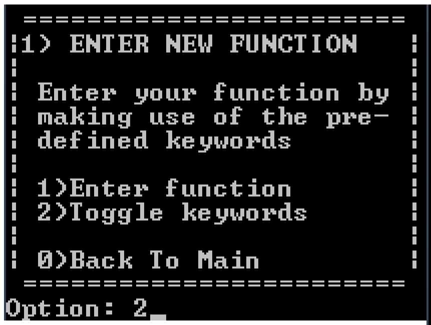
```
 ========================
|3) Evaluate function    |
|                        |
| Result = 0             |
 ========================
```

Assign Variables:

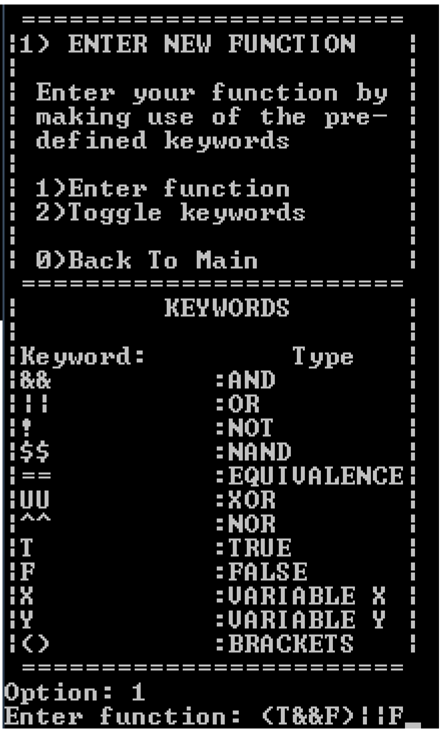


```
 ========================
|1) ASSIGN VARIABLES     |
|                        |
| You can assign values  |
| to the variables you   |
| whish to use in your   |
| function.              |
|                        |
| 1)Assign all true      |
| 2)Assign all false     |
| 3)Assign W             |
| 4)Assign X             |
| 5)Assign Y             |
| 6)Assign Z             |
|                        |
| 0)Back To Main         |
 ========================
Option: 1
```

Enter new function:

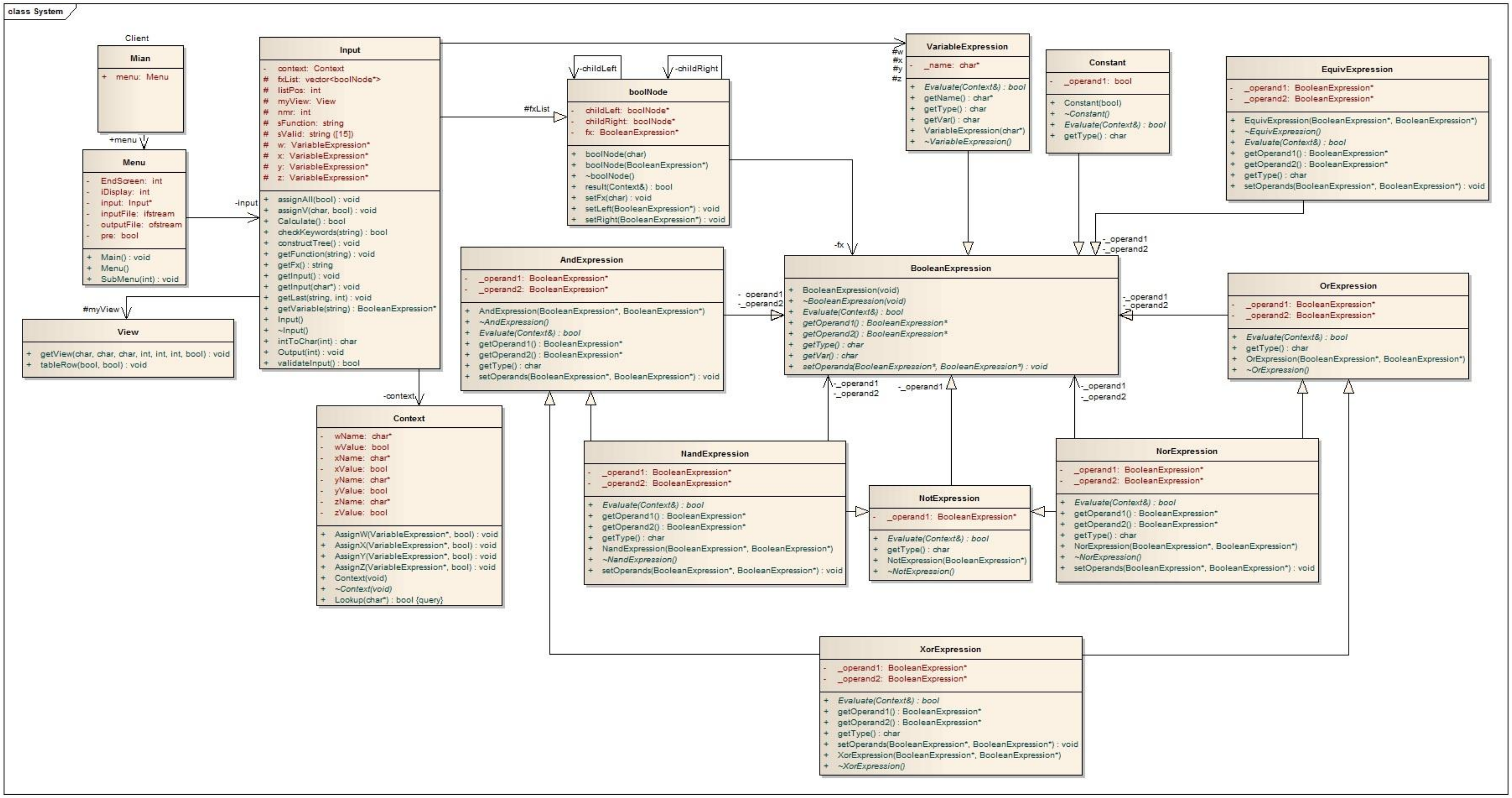


```
 ========================
|1) ENTER NEW FUNCTION   |
|                        |
| Enter your function by |
| making use of the pre- |
| defined keywords       |
|                        |
| 1)Enter function       |
| 2)Toggle keywords      |
|                        |
| 0)Back To Main         |
 ========================
Option: 2_
```



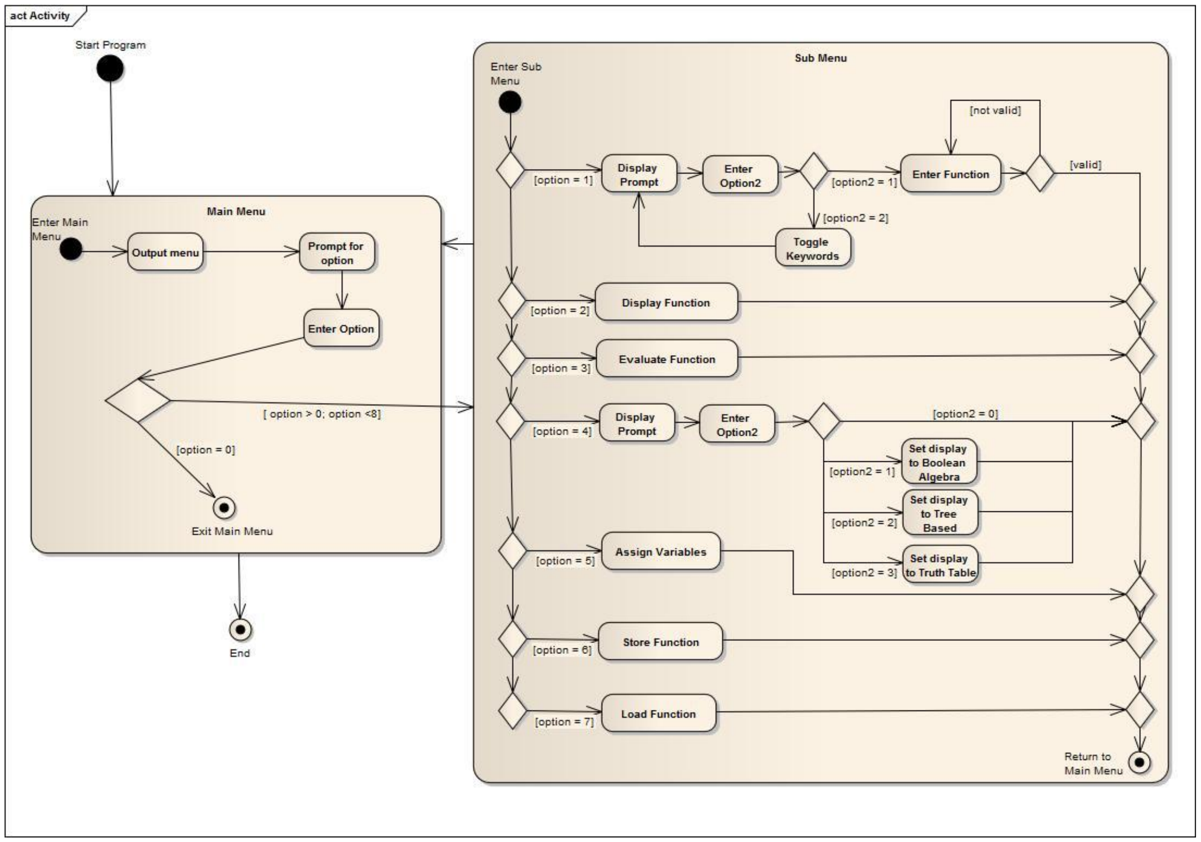
```
 ========================
|1) ENTER NEW FUNCTION   |
|                        |
| Enter your function by |
| making use of the pre- |
| defined keywords       |
|                        |
| 1)Enter function       |
| 2)Toggle keywords      |
|                        |
| 0)Back To Main         |
 ========================
|         KEYWORDS       |
|                        |
|Keyword:        Type    |
|&&          :AND        |
|||          :OR         |
|!           :NOT        |
|$$          :NAND       |
|==          :EQUIVALENCE|
|UU          :XOR        |
|^^          :NOR        |
|T           :TRUE       |
|F           :FALSE      |
|X           :VARIABLE X |
|Y           :VARIABLE Y |
|<>          :BRACKETS   |
 ========================
Option: 1
Enter function: (T&&F)||F_
```

Full project class diagram

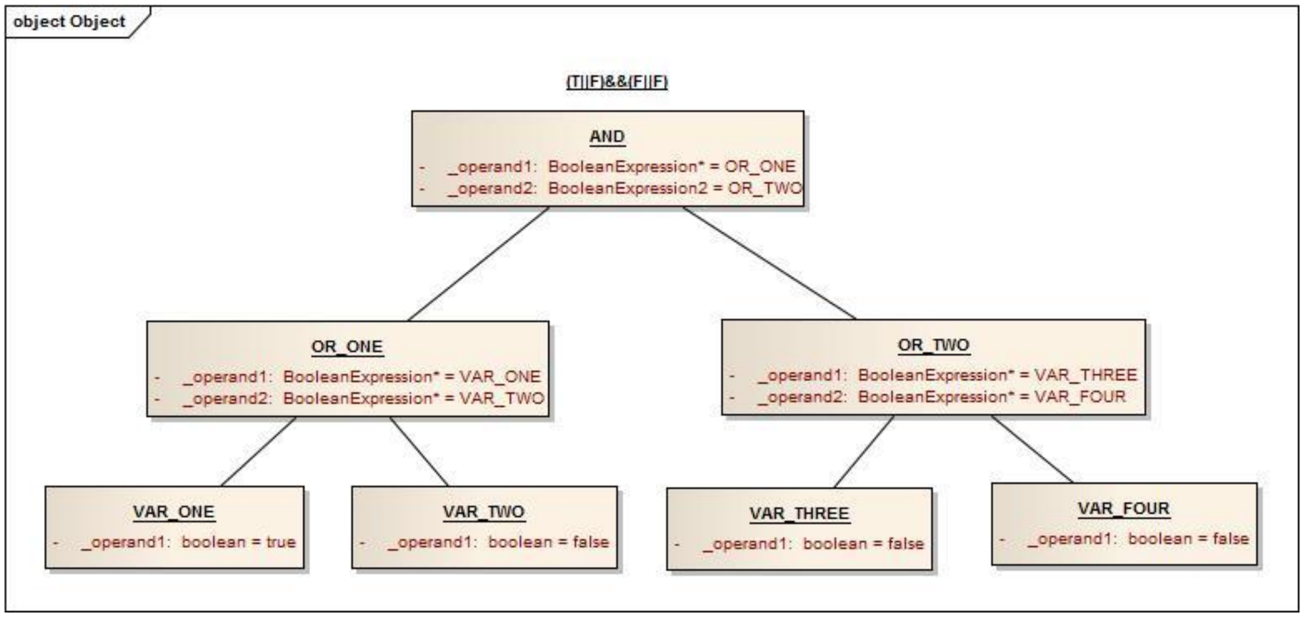

class System
Client
Mian
+ menu: Menu
+menu
Menu
- EndScreen: int
- iDisplay: int
- input: Input*
- inputFile: ifstream
- outputFile: ofstream
- pre: bool
+ Main() : void
+ Menu()
+ SubMenu(int) : void
-input
#myView
View
+ getView(char, char, char, int, int, int, bool) : void
+ tableRow(bool, bool) : void
Input
- context: Context
# fxList: vector<boolNode*>
# listPos: int
# myView: View
# nmr: int
# sFunction: string
# sValid: string ([15])
# w: VariableExpression*
# x: VariableExpression*
# y: VariableExpression*
# z: VariableExpression*
+ assignAll(bool) : void
+ assignV(char, bool) : void
+ Calculate() : bool
+ checkKeywords(string) : bool
+ constructTree() : void
+ getFunction(string) : void
+ getFx() : string
+ getInput() : void
+ getInput(char*) : void
+ getLast(string, int) : void
+ getVariable(string) : BooleanExpression*
+ Input()
+ ~Input()
+ intToChar(int) : char
+ Output(int) : void
+ validateInput() : bool
-context
Context
- wName: char*
- wValue: bool
- xName: char*
- xValue: bool
- yName: char*
- yValue: bool
- zName: char*
- zValue: bool
+ AssignW(VariableExpression*, bool) : void
+ AssignX(VariableExpression*, bool) : void
+ AssignY(VariableExpression*, bool) : void
+ AssignZ(VariableExpression*, bool) : void
+ Context(void)
+ ~Context(void)
+ Lookup(char*) : bool {query}
#fxList
-childLeft
-childRight
boolNode
- childLeft: boolNode*
- childRight: boolNode*
- fx: BooleanExpression*
+ boolNode(char)
+ boolNode(BooleanExpression*)
+ ~boolNode()
+ result(Context&) : bool
+ setFx(char) : void
+ setLeft(BooleanExpression*) : void
+ setRight(BooleanExpression*) : void
-fx
#w
#x
#y
#z
VariableExpression
- _name: char*
+ Evaluate(Context&) : bool
+ getName() : char*
+ getType() : char
+ getVar() : char
+ VariableExpression(char*)
+ ~VariableExpression()
Constant
- _operand1: bool
+ Constant(bool)
+ ~Constant()
+ Evaluate(Context&) : bool
+ getType() : char
EquivExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ EquivExpression(BooleanExpression*, BooleanExpression*)
+ ~EquivExpression()
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ setOperands(BooleanExpression*, BooleanExpression*) : void
-_operand1
-_operand2
AndExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ AndExpression(BooleanExpression*, BooleanExpression*)
+ ~AndExpression()
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ setOperands(BooleanExpression*, BooleanExpression*) : void
- operand1
-_operand2
BooleanExpression
+ BooleanExpression(void)
+ ~BooleanExpression(void)
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ getVar() : char
+ setOperands(BooleanExpression*, BooleanExpression*) : void
-_operand1
-_operand2
OrExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ Evaluate(Context&) : bool
+ getType() : char
+ OrExpression(BooleanExpression*, BooleanExpression*)
+ ~OrExpression()
-_operand1
-_operand2
-_operand1
-_operand1
-_operand2
NandExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ NandExpression(BooleanExpression*, BooleanExpression*)
+ ~NandExpression()
+ setOperands(BooleanExpression*, BooleanExpression*) : void
NotExpression
- _operand1: BooleanExpression*
+ Evaluate(Context&) : bool
+ getType() : char
+ NotExpression(BooleanExpression*)
+ ~NotExpression()
NorExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ NorExpression(BooleanExpression*, BooleanExpression*)
+ ~NorExpression()
+ setOperands(BooleanExpression*, BooleanExpression*) : void
XorExpression
- _operand1: BooleanExpression*
- _operand2: BooleanExpression*
+ Evaluate(Context&) : bool
+ getOperand1() : BooleanExpression*
+ getOperand2() : BooleanExpression*
+ getType() : char
+ setOperands(BooleanExpression*, BooleanExpression*) : void
+ XorExpression(BooleanExpression*, BooleanExpression*)
+ ~XorExpression()

Extra UML diagrams:

Activity diagram of menu



Object diagram of a function:"(T||F)&&(F||F)"


object Object
(T||F)&&(F||F)
AND
- _operand1: BooleanExpression* = OR_ONE
- _operand2: BooleanExpression2 = OR_TWO
OR_ONE
- _operand1: BooleanExpression* = VAR_ONE
- _operand2: BooleanExpression* = VAR_TWO
OR_TWO
- _operand1: BooleanExpression* = VAR_THREE
- _operand2: BooleanExpression* = VAR_FOUR
VAR_ONE
- _operand1: boolean = true
VAR_TWO
- _operand1: boolean = false
VAR_THREE
- _operand1: boolean = false
VAR_FOUR
- _operand1: boolean = false

Header files:

Menu.h

View.h

Input.h

boolNode.h

Interpreter.h

Context.h

Constant.h

BooleanExpression.h

Variable.h

OR.h

AND.h

NOT.h

NAND.h

EQUIV.h

XOR.h

NOR.h

References:

- [0] Gamma, G., Helm, R., Johnson. R and Vlissides J. (1995). Design Patterns - Elements of Reusable Object-Oriented Software. Addison Wesley.
- Bennet S., Skelton J., Lunn K. (2001). Schaum's Outlines UML Second Edition. McGraw-Hill International.
- SourceMaking, URL: http://www.sourcemaking.com