USER GUIDE
SIC ASSEMBLER & SIMULATOR

DEVELOPED BY-

BISHAL ROY (301510501001)
SUFI AFROZA NASRIN (301510501002)
MD DANISH (001410501067)
SIDDHARTH PASWAN(001410501056)

CONTENTS

❖ INTRODUCTION

SIC is a hypothetical computer that has been carefully designed to include the hardware features most often found on real machines, while avoiding unusual or irrelevant complexities. This approach provides the reader with starting point from which to begin the design of system software for a new or unfamiliar computer.

So, we have developed SIC assembler and simulator in our system programming lab.

❖ REQUIREMENTS

- ✓ The code is written in C++. So, C++ compiler is needed.
- ✓ For handling the graphical user interface Qt application should be installed.

❖ INSTRUCTION IMPLEMENTED:

LDA, LDX, STA, STX

ADD, SUB, MUL

TIX, JEQ, JLT, JGT

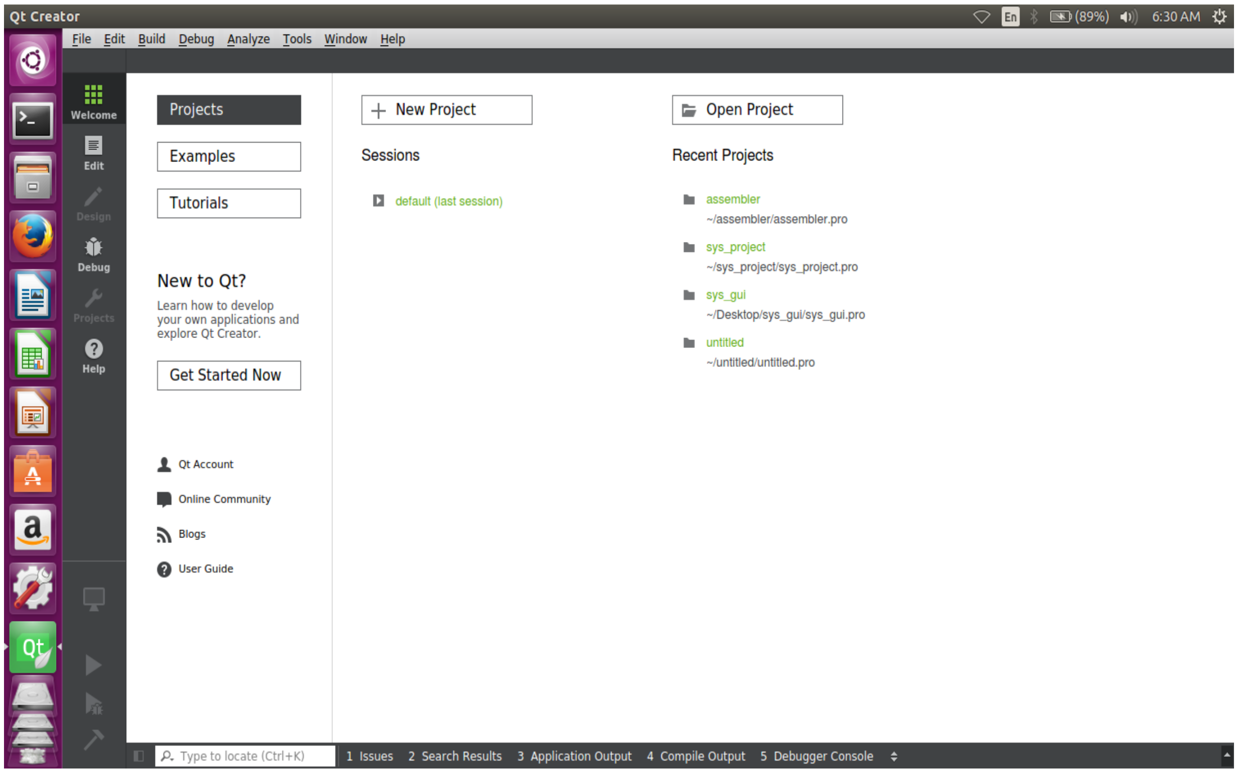
The above instructions of SIC are implemented. But we have written some extra functions for string manipulation and memory handling. These instructions are-

LDS, CMPS, READ, READMEM, WRITE, WRITEMEM, STAM, STXM

❖ HOW TO OPEN AND RUN PROGRAM

1. Write any SIC program and save it as .txt file inside your project.

2. Click on Qt creator. Then screen will appear given below:

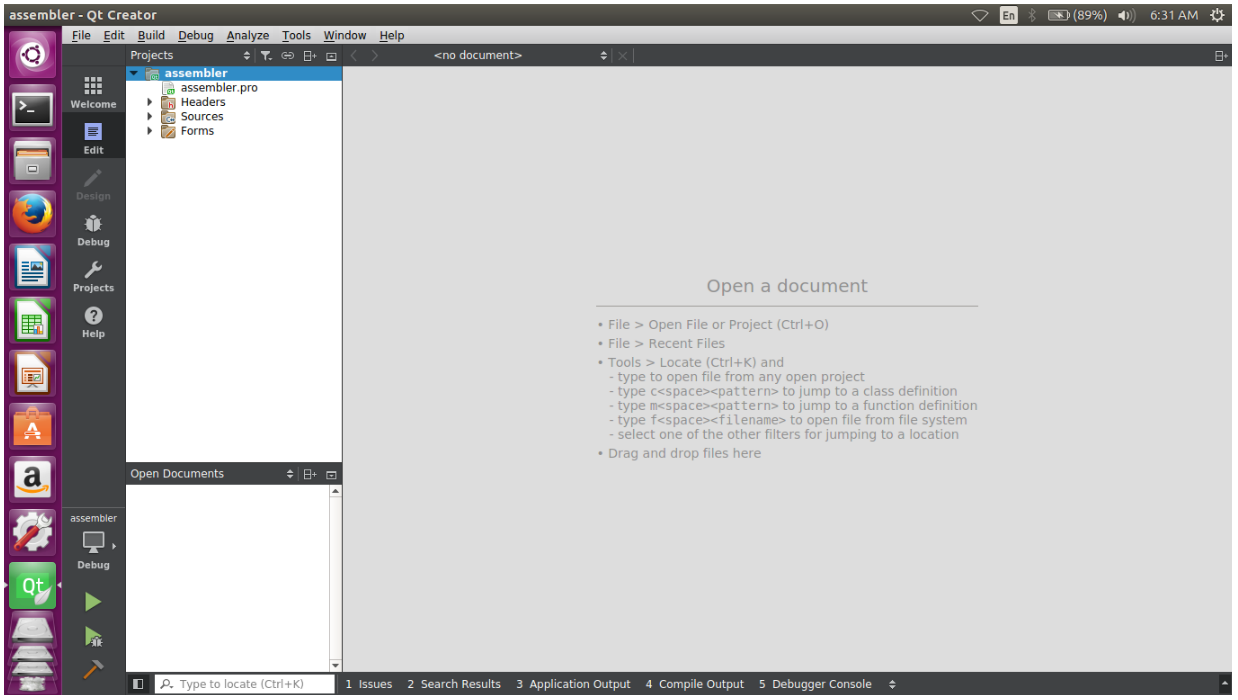


default (last session)

assembler
~/assembler/assembler.pro
sys_project
~/sys_project/sys_project.pro
sys_gui
~/Desktop/sys_gui/sys_gui.pro
untitled
~/untitled/untitled.pro

New to Qt?
Learn how to develop your own applications and explore Qt Creator.

3. Select the project and open that. Then the screen will appear given below:

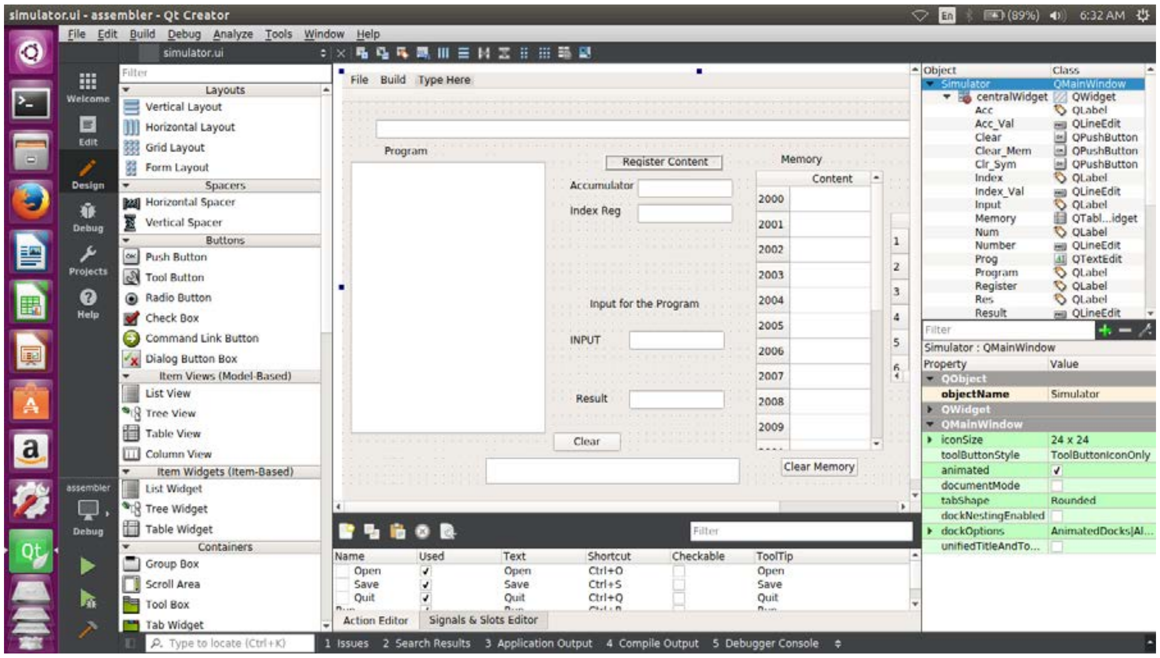


```
assembler
    assembler.pro
    Headers
    Sources
    Forms
```

Open a document

- File > Open File or Project (Ctrl+O)
- File > Recent Files
- Tools > Locate (Ctrl+K) and
  - type to open file from any open project
  - type c<space><pattern> to jump to a class definition
  - type m<space><pattern> to jump to a function definition
  - type f<space><filename> to open file from file system
  - select one of the other filters for jumping to a location
- Drag and drop files here

4. Then click on forms option and the screen given below will appear:



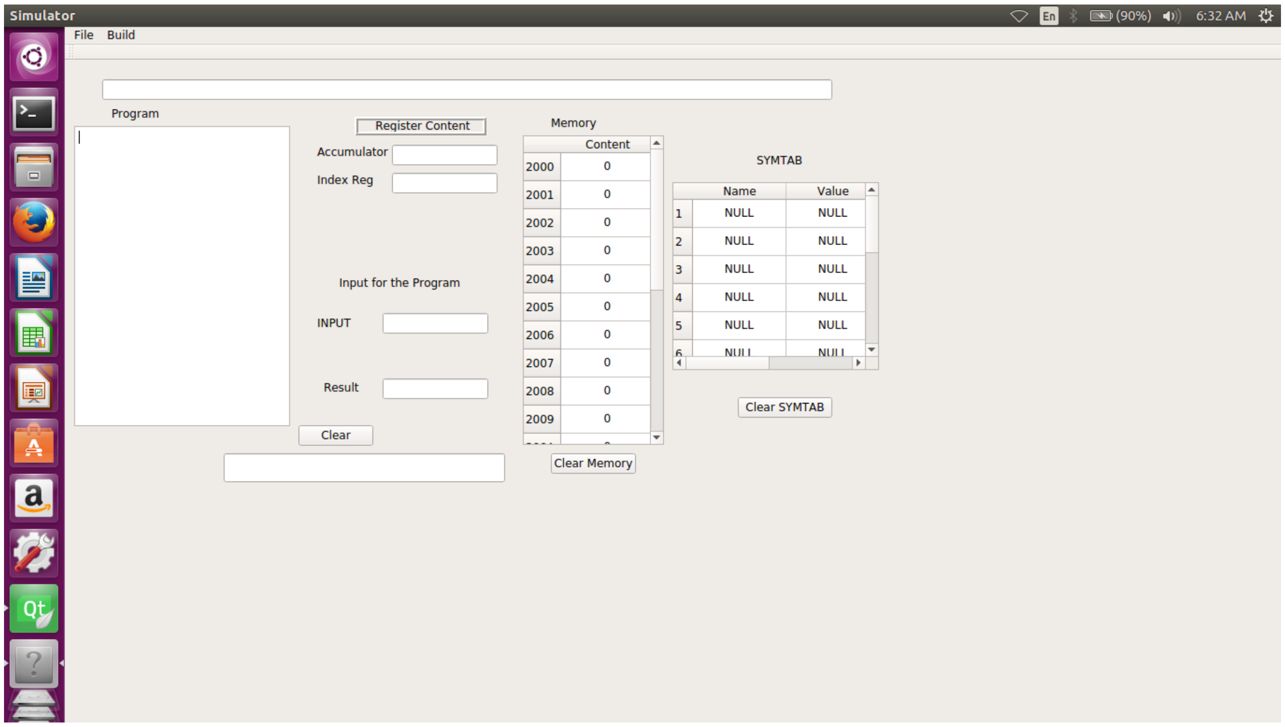
Object	Class
Simulator	QMainWindow
centralWidget	QWidget
Acc	QLabel
Acc_Val	QLineEdit
Clear	QPushButton
Clear_Mem	QPushButton
Clr_Sym	QPushButton
Index	QLabel
Index_Val	QLineEdit
Input	QLabel
Memory	QTabl...idget
Num	QLabel
Number	QLineEdit
Prog	QTextEdit
Program	QLabel
Register	QLabel
Res	QLabel
Result	QLineEdit

Simulator : QMainWindow

Property	Value
QObject	
objectName	Simulator
QWidget	
QMainWindow	
iconSize	24 x 24
toolButtonStyle	ToolButtonIconOnly
animated	✓
documentMode	
tabShape	Rounded
dockNestingEnabled	
dockOptions	AnimatedDocks|Al...
unifiedTitleAndTo...	

Name	Used	Text	Shortcut	Checkable	ToolTip
Open	✓	Open	Ctrl+O		Open
Save	✓	Save	Ctrl+S		Save
Quit	✓	Quit	Ctrl+Q		Quit

5. Now click on run. Then the graphical user interface will appear. User can run their program easily in GUI. The interface is given below:



	Content
2000	0
2001	0
2002	0
2003	0
2004	0
2005	0
2006	0
2007	0
2008	0
2009	0

	Name	Value
1	NULL	NULL
2	NULL	NULL
3	NULL	NULL
4	NULL	NULL
5	NULL	NULL
6	NULL	NULL

6. Go to file now and browse your SIC source code and then go to build option to run your program.

- HOW IT ACTUALLY WORKS?
  Basically, we have implemented a two-pass assembler.

Structure variables:

Program	➢ String Label ➢ String Opcode ➢ String Operand	These variables are to hold label opcode and operand of a code line instruction
Data	➢ string varname ➢ bool reserve ➢ bool word ➢ bool hex ➢ char value[100]	These are to hold the variable name prototype and value of a Data line
Register	➢ char value	This are to hold values of registers A , X , S and B respectively in array reg[4]
Memory	➢ int number	

List of other variables used:

VARIABLE TYPE	NAME	USES
Const string	datastart(".DATA")	It indicates the start of data part of the program
Const string	codestart(".CODE")	It indicates the start of code part of the program
int	Length	It indicates the length of the code of the program
int	Data length	It indicates the length of the data in the dat[ ] array
int	Program counter	It keeps the value of the program counter
Int	Memory counter	It stores the value of memory counter
Bool	Error_flag	It is set according to the error
Int	Error_line	It stores the line number where the error is occurred
Bool	OPCODE_ERROR	It is set according to the error in the opcode
int	COMPARE_FLAG	It is set according to the value of the Accumulator and a string

In pass 1, we are taking the source code as file. Then, all instructions are read line by line. We are parsing all lines to get the label, opcode, and operand part separately.

Functions Implemented:

- PASS(char* filename)
  - saveLine (line , cur_code)
  - getDataLines (opcode)
    - saveDataLine (line)
- int getDataPosition (string &operand)
- void printDataLines ( )
- void displayMemory ( )
- void checkError ( )
- void executeCode ( )

We are creating a symbol table which will be used in pass2.

The symbol table includes the name and value for each label in the source program.

In pass2, all data values are generated defined by BYTE, WORD and program executes.

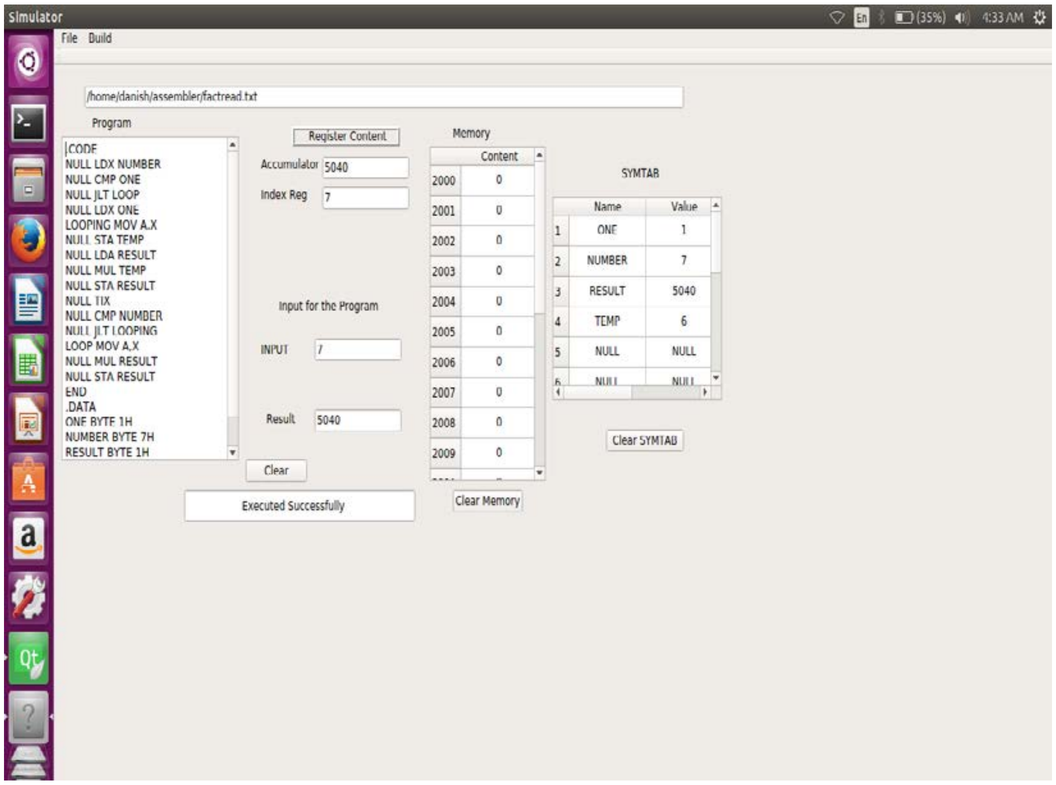
- GUI DESCRIPTION

The Graphical User Interface (GUI) is a created in Qt creator and the user interface consists of all the entities required for the execution of SIC programs in the Assembler Simulator.

- It consists of a large text box portion which will contain the program. User can load program in the box by using action button "**OPEN**" in the "File" Menu in the menu bar. Also user can write any program in the specified box and save it any required location using "**SAVE**" button in the "File" Menu.
- At the top there is a title box which contains the path of the program that is currently in execution.
- There are two text boxes Acc_Val and Index_Val for showing the values of the Accumulator and Index register respectively.
- A tableWidget **Memory** is used to store the contents in the memory and that value can be used for different operation.
- A tableWidget **SYMTAB** is used to show the contents of the Symbol Table that are updated after the execution of the program.
- There are two text boxes for **Input** and **Result** that shows the input on which program has worked and the result that the program has generated.
- There are three PushButtons **Clear**, **Clear Memory** and **Clear SYMTAB** for clearing the contents of program, registers and the input ad result boxes; for clearing **Memory** and for clearing the **SYMTAB** respectively.
- At the top, in the MenuBar UI has a **BUILD** menu which contains the actionButton **RUN** that is used to build and run the program loaded in the program textBox. The Run button does all the functionality of executing the program.

❖ SAMPLE PROGRAMS

1. TO FIND FACTORIAL OF A NUMBER



/home/danish/assembler/factread.txt

Program

```
.CODE
NULL LDX NUMBER
NULL CMP ONE
NULL JLT LOOP
NULL LDX ONE
LOOPING MOV A,X
NULL STA TEMP
NULL LDA RESULT
NULL MUL TEMP
NULL STA RESULT
NULL TIX
NULL CMP NUMBER
NULL JLT LOOPING
LOOP MOV A,X
NULL MUL RESULT
NULL STA RESULT
END
.DATA
ONE BYTE 1H
NUMBER BYTE 7H
RESULT BYTE 1H
```

Accumulator 5040
Index Reg 7

Input for the Program

INPUT 7

Result 5040

Executed Successfully

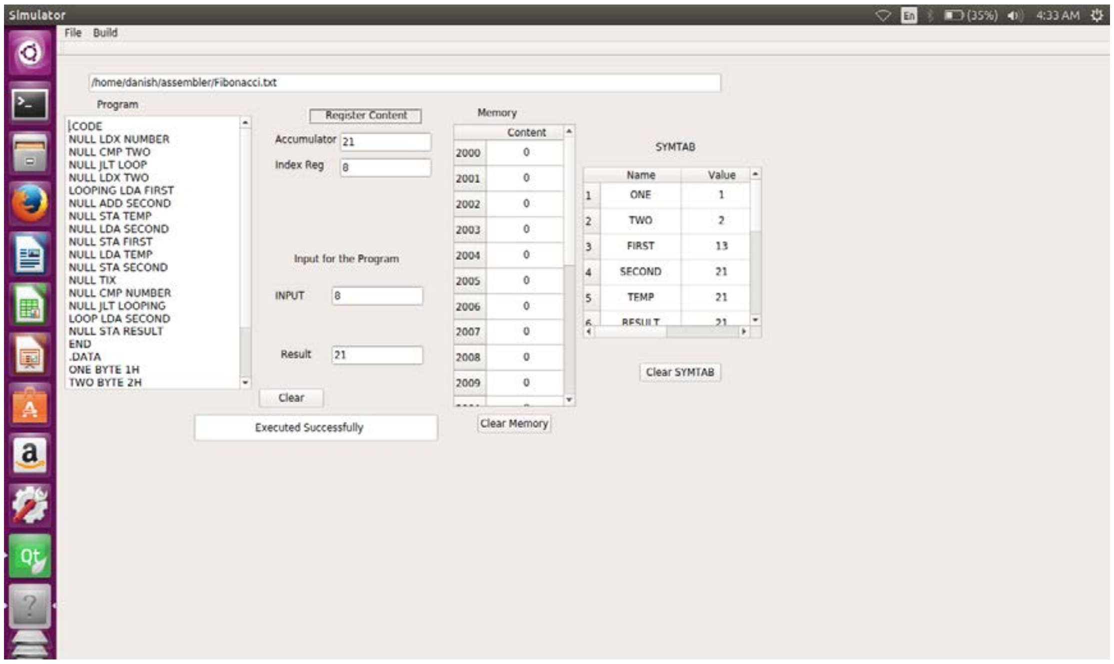
Memory

	Content
2000	0
2001	0
2002	0
2003	0
2004	0
2005	0
2006	0
2007	0
2008	0
2009	0

SYMTAB

	Name	Value
1	ONE	1
2	NUMBER	7
3	RESULT	5040
4	TEMP	6
5	NULL	NULL
6	NULL	NULL

2. TO CREATE FIBONACCI SERIES



/home/danish/assembler/Fibonacci.txt

Program

```
.CODE
NULL LDX NUMBER
NULL CMP TWO
NULL JLT LOOP
NULL LDX TWO
LOOPING LDA FIRST
NULL ADD SECOND
NULL STA TEMP
NULL LDA SECOND
NULL STA FIRST
NULL LDA TEMP
NULL STA SECOND
NULL TIX
NULL CMP NUMBER
NULL JLT LOOPING
LOOP LDA SECOND
NULL STA RESULT
END
.DATA
ONE BYTE 1H
TWO BYTE 2H
```

Accumulator 21
Index Reg 8

Input for the Program

INPUT 8

Result 21

Executed Successfully

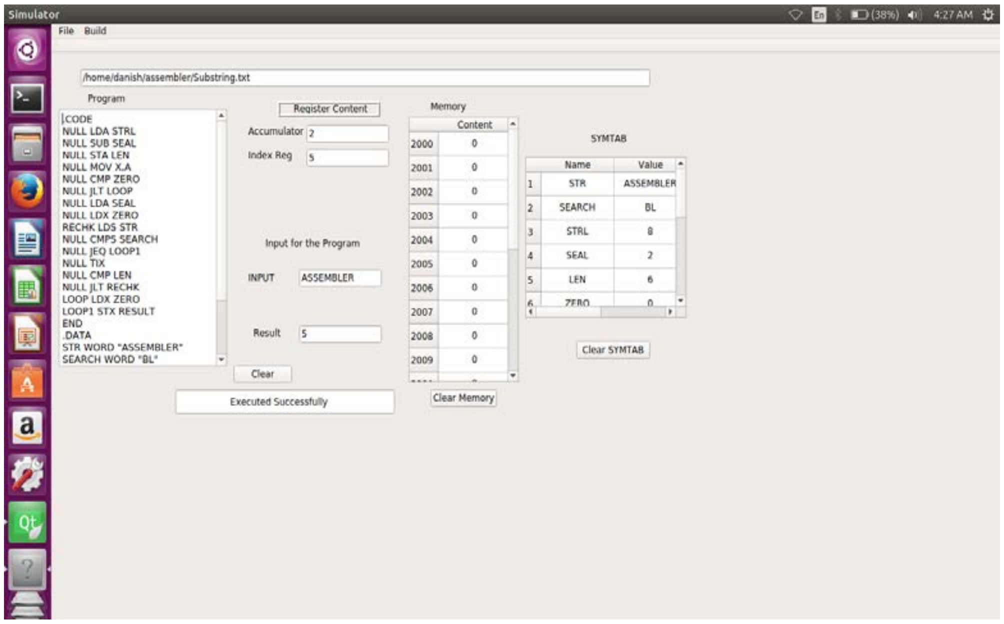
Memory

	Content
2000	0
2001	0
2002	0
2003	0
2004	0
2005	0
2006	0
2007	0
2008	0
2009	0

SYMTAB

	Name	Value
1	ONE	1
2	TWO	2
3	FIRST	13
4	SECOND	21
5	TEMP	21
6	RESULT	21

3. TO FIND SUBSTRING USING LINEAR SEARCH



/home/danish/assembler/Substring.txt

Program

```
.CODE
NULL LDA STRL
NULL SUB SEAL
NULL STA LEN
NULL MOV X,A
NULL CMP ZERO
NULL JLT LOOP
NULL LDA SEAL
NULL LDX ZERO
RECHK LDS STR
NULL CMPS SEARCH
NULL JEQ LOOP1
NULL TIX
NULL CMP LEN
NULL JLT RECHK
LOOP LDX ZERO
LOOP1 STX RESULT
END
.DATA
STR WORD "ASSEMBLER"
SEARCH WORD "BL"
```

Accumulator: 2
Index Reg: 5

Input for the Program
INPUT: ASSEMBLER
Result: 5

Executed Successfully

Memory

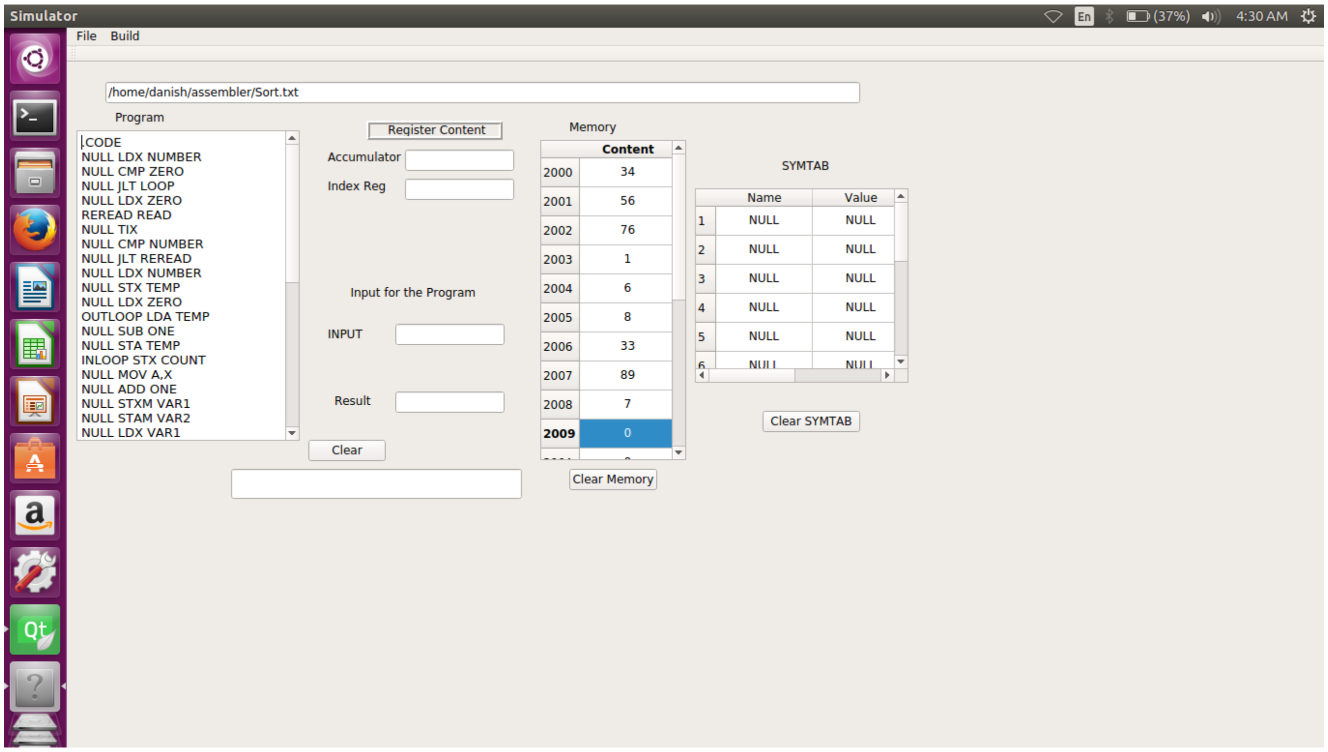
	Content
2000	0
2001	0
2002	0
2003	0
2004	0
2005	0
2006	0
2007	0
2008	0
2009	0

SYMTAB

	Name	Value
1	STR	ASSEMBLER
2	SEARCH	BL
3	STRL	8
4	SEAL	2
5	LEN	6
6	ZERO	0

4. TO PERFORM SORTING

A>BEFORE SORTING



/home/danish/assembler/Sort.txt

Program

```
.CODE
NULL LDX NUMBER
NULL CMP ZERO
NULL JLT LOOP
NULL LDX ZERO
REREAD READ
NULL TIX
NULL CMP NUMBER
NULL JLT REREAD
NULL LDX NUMBER
NULL STX TEMP
NULL LDX ZERO
OUTLOOP LDA TEMP
NULL SUB ONE
NULL STA TEMP
INLOOP STX COUNT
NULL MOV A,X
NULL ADD ONE
NULL STXM VAR1
NULL STAM VAR2
NULL LDX VAR1
```

Accumulator:
Index Reg:

Input for the Program
INPUT:
Result:

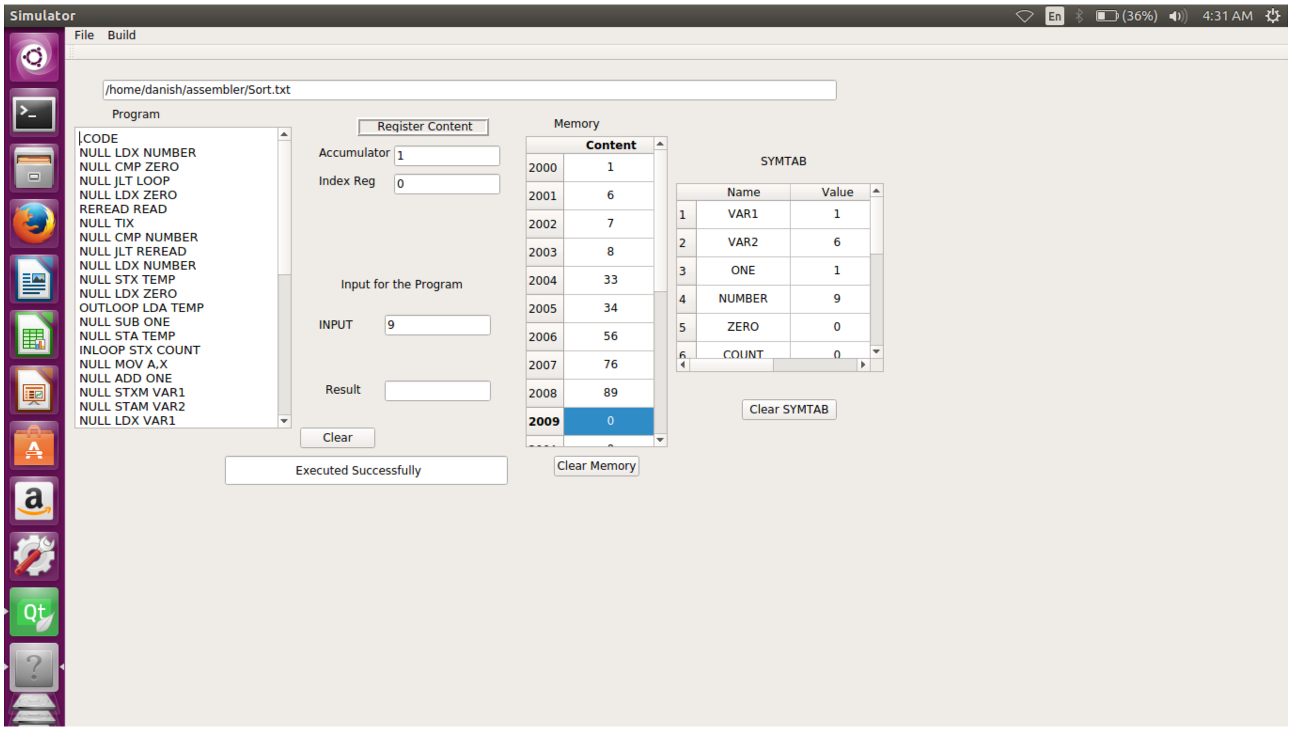
Memory

	Content
2000	34
2001	56
2002	76
2003	1
2004	6
2005	8
2006	33
2007	89
2008	7
2009	0

SYMTAB

	Name	Value
1	NULL	NULL
2	NULL	NULL
3	NULL	NULL
4	NULL	NULL
5	NULL	NULL
6	NULL	NULL

B>AFTER SORTING



/home/danish/assembler/Sort.txt

Program

```
CODE
NULL LDX NUMBER
NULL CMP ZERO
NULL JLT LOOP
NULL LDX ZERO
REREAD READ
NULL TIX
NULL CMP NUMBER
NULL JLT REREAD
NULL LDX NUMBER
NULL STX TEMP
NULL LDX ZERO
OUTLOOP LDA TEMP
NULL SUB ONE
NULL STA TEMP
INLOOP STX COUNT
NULL MOV A,X
NULL ADD ONE
NULL STXM VAR1
NULL STAM VAR2
NULL LDX VAR1
```

Accumulator 1

Index Reg 0

Input for the Program

INPUT 9

Result

Executed Successfully

Memory

	Content
2000	1
2001	6
2002	7
2003	8
2004	33
2005	34
2006	56
2007	76
2008	89
2009	0

SYMTAB

	Name	Value
1	VAR1	1
2	VAR2	6
3	ONE	1
4	NUMBER	9
5	ZERO	0
6	COUNT	0