www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 1, Jan 2014 Page No. 3704-3729

Graphical Interpreter

Prof. A.Nagesh, A.Chiranjeevi2,jayanth vangari3,Malyala.Madhu shyam4

1(Professor, CSE. Dept., Mahatma Gandhi Institute of Technology, Hyderabad, Andhra Pradesh, India)

2(4/4 B.Tech, CSE. Dept., Mahatma Gandhi Institute of Technology, Hyderabad, Andhra Pradesh, India)

3(4/4 B.Tech, CSE. Dept., Chaitanya Bharathi Institute of Technology, Hyderabad, Andhra Pradesh, India)

4(4/4 B.Tech, civil. Dept., Mahatma Gandhi Institute of Technology, Hyderabad, Andhra Pradesh, India

1. INTRODUCTION

1.1 PURPOSE

In a large number of educational institutions, programming languages are taught without giving the student an adequate appreciation of the way in which the languages are executed on the machine. Even though students learn to write code well, they lack the basic knowledge of computer architecture.

In order to become better programmers, every programmer must have an overall understanding of the machine's internals and how programming instructions are executed in the machine to produce the desired results. Experimentation / interaction leads to a much fuller understanding and retention of any subject. In this project, an application **"Graphical Interpreter: Buddhuram Dumbo In Action"** is to be developed to allow students to feed commands and observe how they get executed inside the machine.

The purpose of this document is to record the various activities that were undertaken during the course of the implementation of this project.

1.2 THE SCOPE OF THE GRAPHICAL INTERPRETER

Ø The student can feed a single declaration and observe a memory location getting labeled. Ø The student can feed an initialization statement to see the memory location being 'filled'. Ø The student can feed simple instructions and observe the effect on memory/registers.
Ø The student can store his own set of instruction and replay them.
Ø The student can load illustrative sample sets and see the effect.

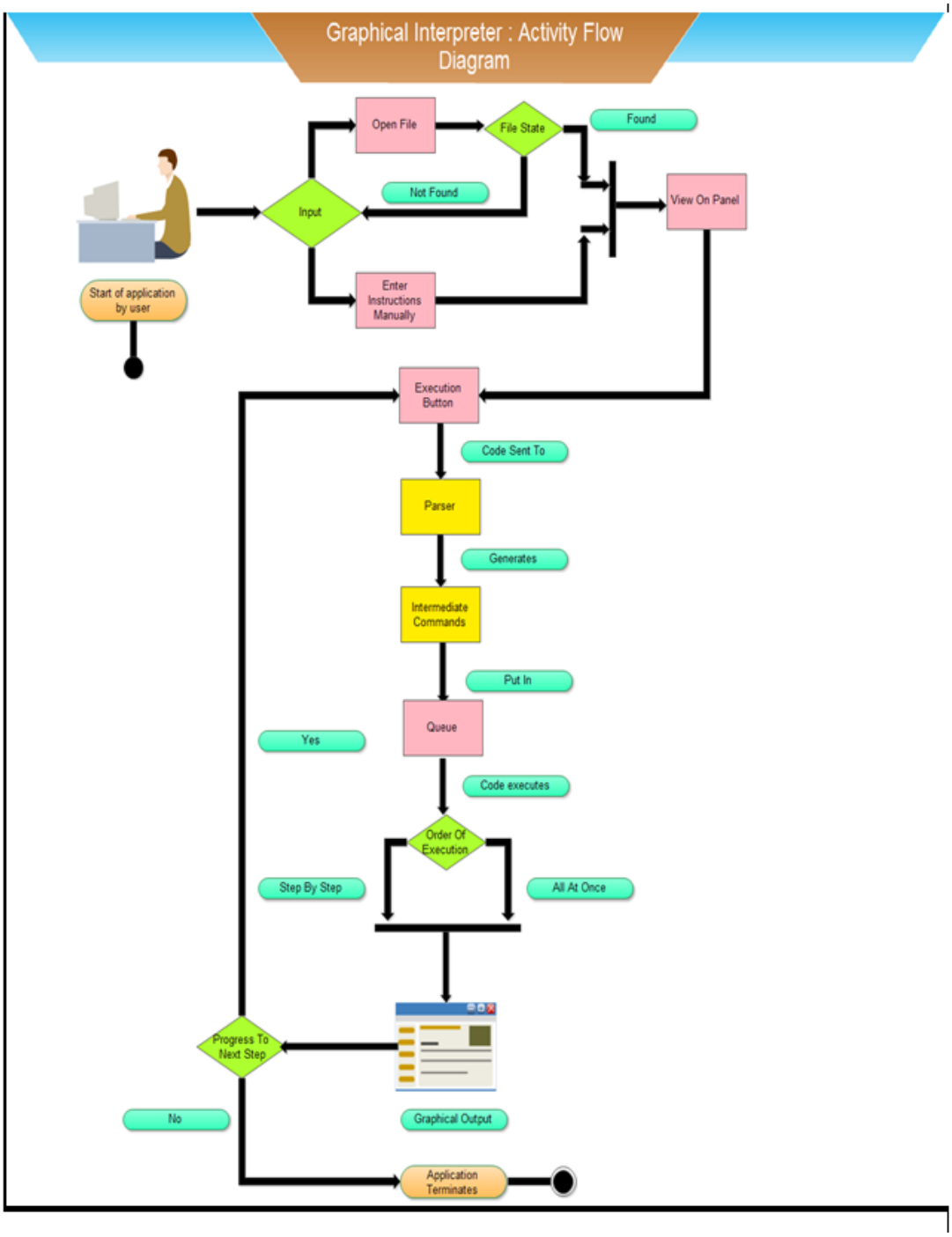
.

1.3 OBJECTIVES

The Graphical Interpreter is intended as an educational tool that helps students better understand basic computer architecture. The application is intended to initially execute C commands. The architecture of the system is intended to be neutral in order for the application to be extensible. The application is intended to be a stand-alone application that will function without the use of a server. It is also intended as a cross platform tool that can be run over a variety of platforms.

2. APPLICATION MODULES

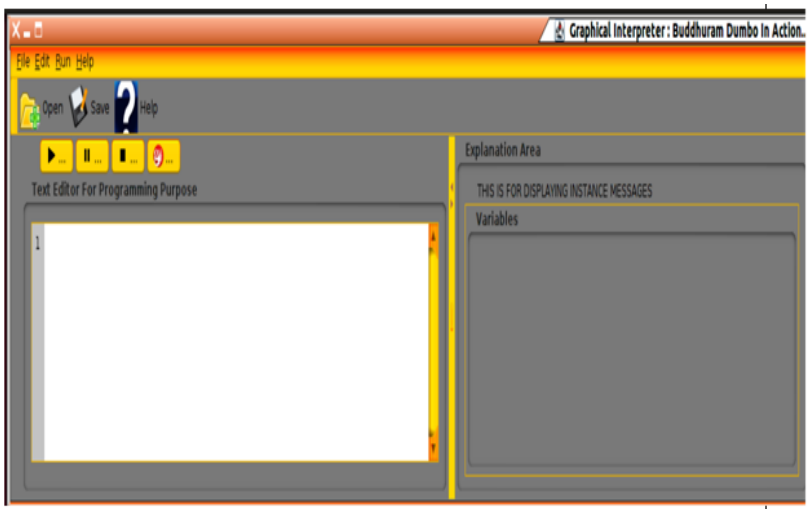
ACTIVITY DIAGRAM:



The application **"Graphical Interpreter: Buddhuram Dumbo in Action"** executes in following manner:

1. The user may input C code in two ways:
    a. Selects a file which has C code stored in it.
    b. Write instructions manually using the text editor.
2. The C code from the file or the instructions that the user has entered can be viewed in the view panel.
3. The user clicks the execute button.
4. The C code is then sent to the Parser which tokenizes the code and calls the appropriate graphical interface functions.
5. The graphical interface functions generate intermediate codes.
6. Intermediate codes are pushed in the queue.
7. As the user clicks the play button, intermediate code is executed and graphics animations are played.
8. The user has two options; either to execute intermediate code step-by-step or all at once.
9. The execution stops when there are no more commands left in the queue. A message stating the same is displayed to the user.

2.1 FRAMEWORK MODULE



The Framework of **Graphical Interpreter** is made in java language by importing java libraries:

- javax.swing.*

§ java.awt.*

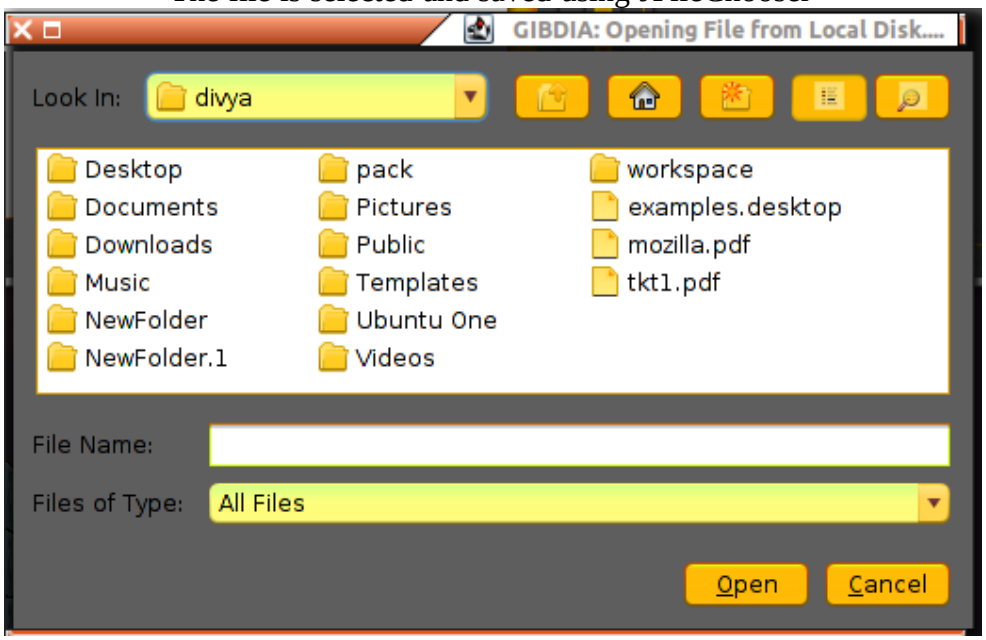
The graphical interpreter ‘executes’ a given code segment statement by statement. There are two options for specifying input. The student can store his own code segments and review their execution. Finally the student can write his own statement and observe the machine execute the code.

This GUI (Graphical User Interface) application has following functionalities:

§ Open File Operation:

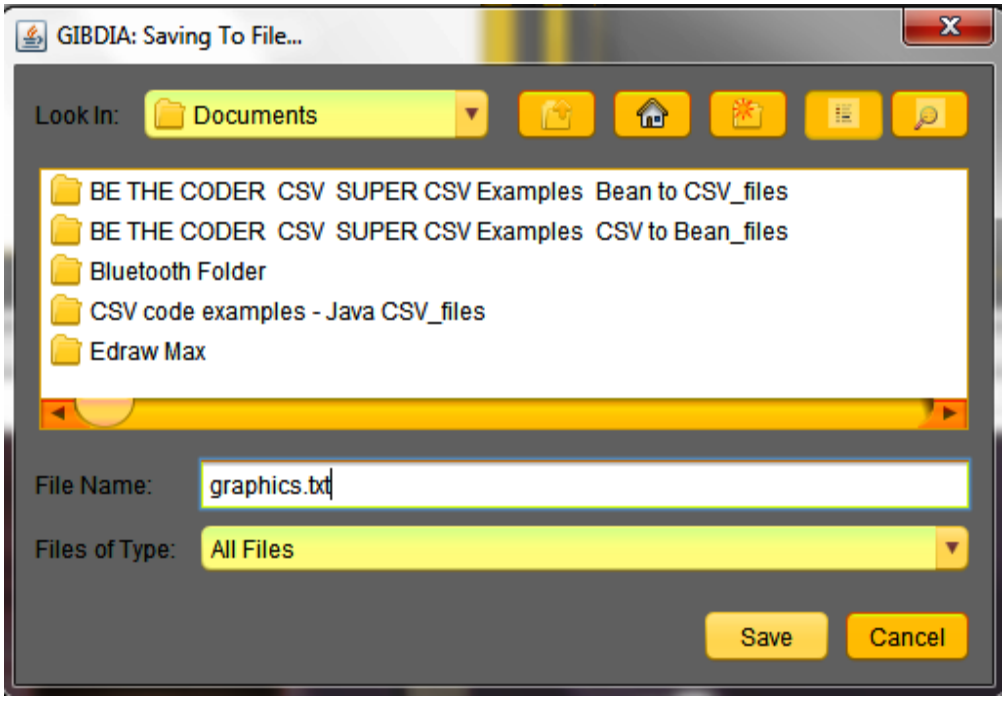
Used to select a file which user wants to use as input and displays the contents of it in Text Editor Panel for execution.

The file is selected and saved using JFileChooser



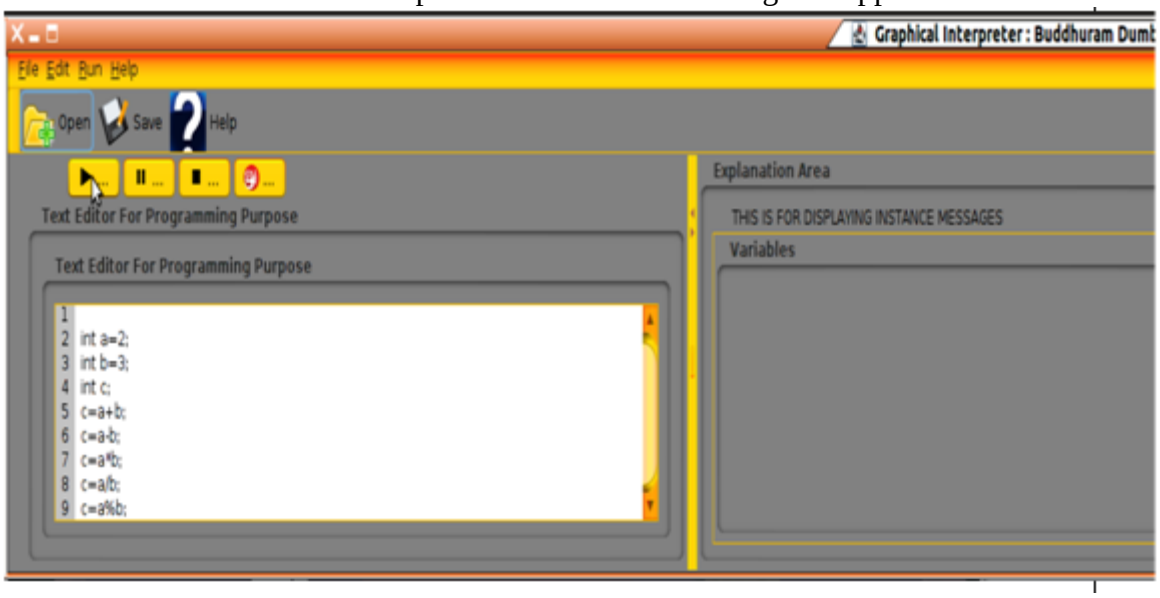
§ Save File Operation:

Used to save the C instructions entered by user manually.



§ Help Menu:

Provides Online help and user manual for using this application.



§ Animation Control

- Play : Plays the animation.
- Pause : Pauses the animation.
- Stop: Stops the animation.
- Execute: Executes the code segments step-by-step.

§ Explanation and Error Displays

- Gives information about the current instruction being executed and its memory location labeling.

- Displays errors if the current instruction has errors like syntax errors, division by zero etc.

§ Variables Panel

- Display Values of each variable

JAVA CLASSES IN FRAMEWORK MODULE

There are four classes in GUI:

1). **FrameWorkGIBDIA** : This is the main class of graphical user interface. User can open the file using file browser and contents of file are displayed on the screen. This class shows the buttons for controlling the animation. It also provides the flexibility of typing the instructions manually through keyboard and execution of them.

2). **EditorPanel**: EditorPanel class is responsible for text area where the source code is shown on the user interface. Text area has undoable listener. It supports all cut, copy, and paste operations

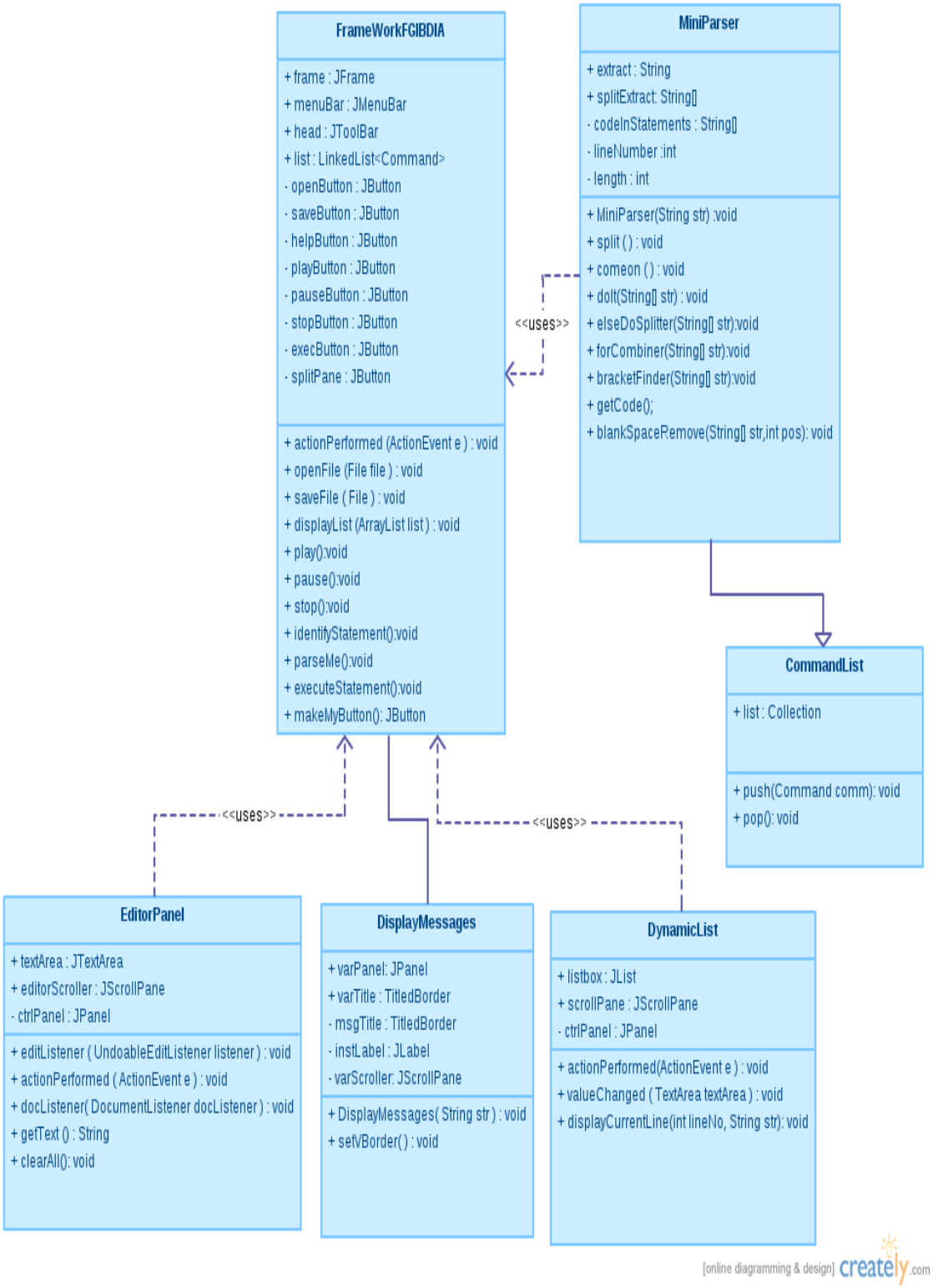
3). **DynamicList**: When a user clicks execute button DynamicList class convert the text area into ArrayList that gets the code ready for execution.

4). **DisplayMessages**: Displays all the error messages that occurred while executing the instant instruction or Error Display area on user interface.

Error may be graphics module error, parser errors like syntax, semantic, etc.... It also shows the explanation for the instruction that is being executed.

5). **MiniParser**: The Miniparser orders the given text before sending it to the parser. It separates the input text into C language statements. It also puts brackets and other key words into separate statements.

6). **CommandList**: CommandList stores the graphical instructructions to be executed. The push method is used by the parser to insert commands. The pop command is used by the framework to remove the first instruction in the queue. **ACTIVITY CLASS DIAGRAM : FRAMEWORK MODULE**


FrameWorkFGIBDIA
+ frame : JFrame
+ menuBar : JMenuBar
+ head : JToolBar
+ list : LinkedList<Command>
- openButton : JButton
- saveButton : JButton
- helpButton : JButton
- playButton : JButton
- pauseButton : JButton
- stopButton : JButton
- execButton : JButton
- splitPane : JButton
+ actionPerformed (ActionEvent e) : void
+ openFile (File file) : void
+ saveFile (File) : void
+ displayList (ArrayList list) : void
+ play():void
+ pause():void
+ stop():void
+ identifyStatement():void
+ parseMe():void
+ executeStatement():void
+ makeMyButton(): JButton
MiniParser
+ extract : String
+ splitExtract: String[]
- codeInStatements : String[]
- lineNumber :int
- length : int
+ MiniParser(String str) :void
+ split ( ) : void
+ comeon ( ) : void
+ doIt(String[] str) : void
+ elseDoSplitter(String[] str):void
+ forCombiner(String[] str):void
+ bracketFinder(String[] str):void
+ getCode();
+ blankSpaceRemove(String[] str,int pos): void
<<uses>>
CommandList
+ list : Collection
+ push(Command comm): void
+ pop(): void
<<uses>>
<<uses>>
EditorPanel
+ textArea : JTextArea
+ editorScroller : JScrollPane
- ctrlPanel : JPanel
+ editListener (UndoableEditListener listener) : void
+ actionPerformed (ActionEvent e) : void
+ docListener(DocumentListener docListener) : void
+ getText () : String
+ clearAll(): void
DisplayMessages
+ varPanel: JPanel
+ varTitle : TitledBorder
- msgTitle : TitledBorder
- instLabel : JLabel
- varScroller: JScrollPane
+ DisplayMessages(String str) : void
+ setVBorder( ) : void
DynamicList
+ listbox : JList
+ scrollPane : JScrollPane
- ctrlPanel : JPanel
+ actionPerformed(ActionEvent e) : void
+ valueChanged (TextArea textArea) : void
+ displayCurrentLine(int lineNo, String str): void
[online diagramming & design] creately.com

2.2 Parser Module

Parser is the second module of the project called by the framework. It takes a raw C statement and converts it into intermediate commands that are understood by the graphical module.

While doing so, the parser should keep in mind that the order of intermediate commands generated is in accordance with the precedence and associativity defined in the C language.

DUMMY PARSER

To simulate the behaviour of a parser we have written a dummy parser, where intermediate commands are manually written. The constructor of the dummy parser class takes file name as the input. It has the following functions:-

(1) boolean parse (String Cstatement, int lineno);

The framework calls this parse function with the C statement and the line number as argument. The parse function has two switch statements. First switch is for the filename and the second switch is for line number. Inside this, all the intermediate commands for that particular line is coded.
The function returns true or false based on whether that C statement is properly compiled without any errors or not.

2.3 Graphics Module

The Graphics part of **Graphical Interpreter** is made in LIBGDX , Blender and java libraries.

Libgdx is a Java game development framework that provides a unified API that works across all supported platforms.

The framework provides an environment for rapid prototyping and fast iterations. Instead of deploying to Android/iOS/JavaScript after each code change, you can run and debug your game on the desktop, natively. Desktop JVM features like code hot swapping reduce your iteration times considerably.

FEATURES:

The exceptional features of the Libgdx API motivated us to use it. The major features of the API are

Libgdx provides a single API to obtain crossplatform feature. It provides the capability to run the application on various platforms .They are

a. Windows
b. Linux
c. Max OS X
d. Android (+1.5)
e. iOS
f. Java Applet (requires JVM to be installed)
g. Javascript/WebGL

Libgdx provides strong API for the graphics operations. It supports both 2d and 3D graphics. The functions of the API are built on the concepts of object oriented programming. The major advantage is its handling of the 3d objects which enables us to do many graphics operations with already predefined methods.

Libgdx also allows us to use various utilities like Texture packer, bitmap font generator, GDX setup UI and many more. This allows us to work with the API for dynamic production of instances and their rendering.

Libgdx also has libraries for mathematics, physics, file input and output handling and many other utitlities.

BASIC STRUCTURE:

The basic structure of the Libgdx program consists of five major methods. They are as follows:

Create ()-This is used to create the graphic objects and all the declarations are handled by this method.

Render ()-This is the major method which is used to obtain the graphics onto the screen. The render is the infinite recursive method which is continuously plots the graphics defined in the program.

Dispose ()-This method is used to dispose the assets continuously on the screen to obtain the effect of graphics

Resume ()-This is used to resume the paused graphics. Pause ()-

This is used to pause the graphic operations.

Resize ()-This is used to obtain the resize property of the graphic window.

Example code

```
package com.test.myfirsttriangle;
import com.badlogic.gdx.ApplicationListener;

public class MyFirstTriangle implements ApplicationListener {
	@Override
	public void create() {
		// TODO Auto-generated method stub

	}

	@Override
	public void dispose() {
		// TODO Auto-generated method stub

	}

	@Override
	public void pause() {
		// TODO Auto-generated method stub

	} @Override
public void render() {// TODO Auto-generated method stub

	} @Override
	public void resize(int width, int height) {
		// TODO Auto-generated method stub

	} @Override
	public void resume(){	}
```

Blender is a free and open-source 3D computer graphics software product used for creating animated films, visual effects, art, 3D printed models, interactive 3D applications and video games. Blender is a dominant open-source product with a range of features comparable to mid- to high-range commercial, proprietary software.

The default blender screen is fixed with the camera, lamp and the basic primitive mesh i.e the cube.

FEATURES:

Blender has a relatively small installation size, of about 70 megabytes for builds and 115 megabytes for official releases. Official versions of the software are released for Linux, Mac OS X, Microsoft Windows, and FreeBSD in both 32 and 64 bits. This software contains features that are characteristic to the high end 3D software.

Some of the features that make blender one of the most popular among all the Animation software are:

Support of a huge variety of geometric primitives like Bezier curves, meshes etc. Internal render engine with scanline ray tracing, indirect lighting, and ambient occlusion that can export in a wide variety of formats. A pathtracer render engine called Cycles.

Keyframed animation tools including inverse kinematics, armature (skeletal), hook, curve and lattice-based deformations, shape keys (morphing), non-linear animation, constraints, and vertex weighting.

Modifiers to apply non-destructive effects.

Python scripting for tool creation and prototyping, game logic, importing and/or exporting from other formats, task automation and custom tools.

The Blender Game Engine

Procedural and node-based textures, as well as texture painting, projective painting, vertex painting, weight painting and dynamic painting.

Camera and object tracking. Excellent user Interface

BASIC BLENDER FEATURE CREATION:

Blender is flexible and easily adaptable software for animations. Every function in blender has a shortcut making it easy for usage. It is provided with contextual menus for more logical and streamlined operations. There are also imparting certain features like colors, themes, widgets etc.

Blender has two major modes of working. They are:

1) Object Mode: In this mode we can create our 3D objects by inserting and modifying the already provided set of objects. The operations on the complete objects can be performed in the object mode.

2) Edit Mode: In this mode we can edit or modify the selected object or any part of the selected object. The edit mode is used for performing various operations on the attributes of the object like size, location, alignment etc.

The major advantage of blender is the 3d model that is made can be exported into different formats. To embed the model made in blender into our gaming platform Libgdx we export it into the Wave front format (.obj) and import it as a model into our application on the Libgdx platform.

THE GRAPHICS WINDOW:

The graphics window is the graphical output that the user can see. It contains the graphical objects depicting the hardware used for executing the basic "C" instructions. The graphics window mainly contains the blender models imported into Libgdx code of the application.

The main components in the window are:

1) The Room: The room depicts the hardware of the workspace where the memory and registers operations take place

2) The Boxes: The boxes which are on to the left side of the room depict the memory locations in the computer. Each box depicts a single memory location .The variable and value of the variable is stored

in the memory in this case the box. The memory locations are infinite in number to enable large variable declarations.

3) The Windows: The windows which are to the right of the room depict the registers of the computer. The values of the variables which are involved in the operations are taken from the memory i.e the box and put into the registers "R1" and "R2" for evaluation. The result is put in a special window named accumulator "ACC". The result is then taken back and stored into the memory boxes.

4) The Display: The display board is hung from the ceiling of the roof. This board indicates the present statement that is being executed. This enables the dumbo to pick up the values and perform operations on them.

5)The Dumbo: The dumbo is the important part of the execution as it depicts the control flow of instructions and the procedure of how these instructions are performed. The dumbo picks the values from display, puts it in memory or registers and manipulates the position and value according to the evaluation of the instructions given.

6) The Label: The values appearing on the box and the register are dynamically generated which depict the variable name and its value. The label is generated according to the execution of the given instructions.

These objects included in the graphics are given animation to give a sense of execution through the graphical interface commands.

When the user types the instruction " int a=2;", the instruction is displayed on the screen and the following scenario takes place.

The current instruction is displayed on display screen.

A free memory box is selected and it is labeled as "a"

The dumbo opens the memory box “ a ” and puts “ 2 ” into it

Similar operation takes place when user inputs “int b=3;” and “int c”;.

When the user enters “c=a+b;” the following scenario takes place:

- The dumbo picks the value of variable “a” from memory box “a” and puts it into register R1.
- The dumbo picks the value of variable “b” from memory box “b” and puts it into register R2.
- Appropriate mathematical operation is performed and result is stored in accumulator. Here, addition is being performed.


4. c=a+b;
R1
R2
Acc


test
2
R1
R2
Acc


MEMO


test
2
3
R1
R2
Acc

- Memory cache is used to store values temporary

JAVA CLASSES IN GRAPHICS MODULE Package graphics:

Dumbo Class:
Main graphics class.

Public void create() : default Libgdx method for initialization of graphical components.

Public void loading() : loads assets in program

Public void render() : it's a game loop(called by Libgdx as many times as possible) and contains dynamic handling of graphical objects.

Public void show_text(int type, int value, String text) : displays the operand and value of variables as text in box, registers, accumulator .

Public void box_close(int boxno) : close the box with given box number. Public void box_open(int boxno) : opens the box with given box number. Public void labelBox(String label, int boxNo) : labels the given box with label. Public void look_to_default() : sets the camera at default position.

Public void look_to_memory_wall(int boxNo) : sets the camera to look at given box.

Public void look_to_register_wall() : sets the camera to look at register wall. Public void movedumbo(int init_pos_type,int final_pos_type,int final_pos) : moves the dumbo from initial to final position.

Private void movedumbo() : called by render method to move the dumbo(for internal use).

Private void rotate_dumbo() : to rotate the dumbo while moving(for internal use).

Public void dispose() : releases all the resources while exiting. Public Boolean needsGL20() : returns true if openGL 2.0 is needed.

Public void pause() : called when pause button is pressed and saves the program state.

Public void resume() : resumes from saved program state.

Public void displaycmd(String string) : displays the given string on screen.

DesktopGraphic Class: Main class which instantiates dumbo graphics class and Framework class.

main(): main class called by jvm.

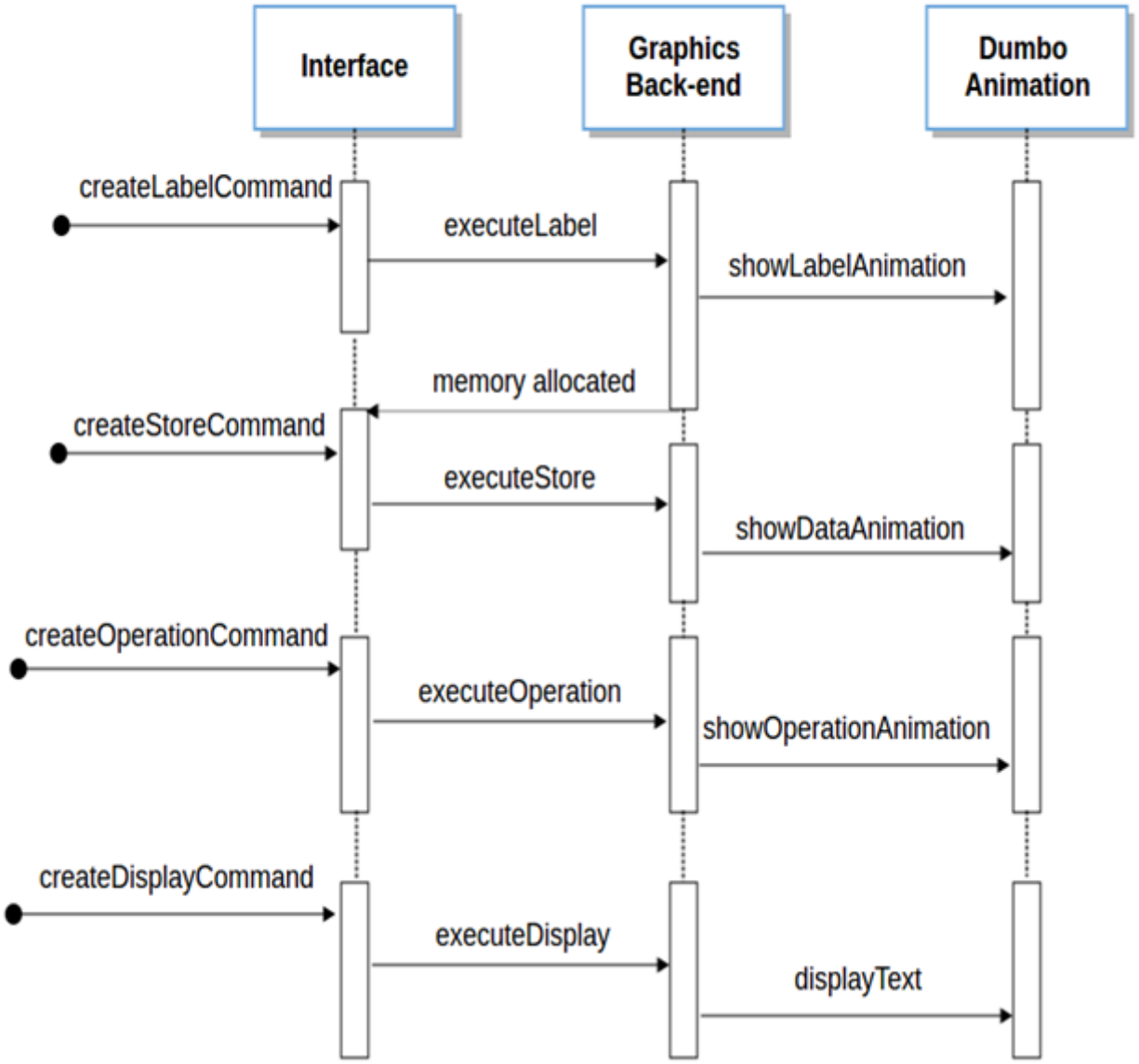
DisplayMessages Class: Displays messages in the framework.

DisplayLine(): display the current highlighted line.

FrameWorkGIBDIA Class: Main class of framework. And controls all execution of commands.

startFramework():start loading all panel and its listeners.
showGUI(): shows all defined GUI components.

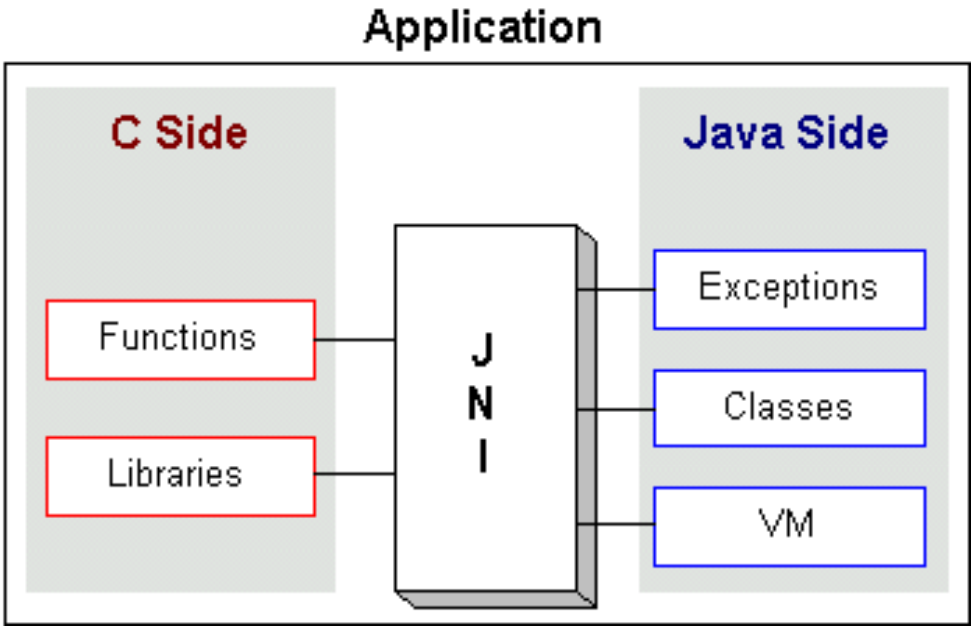
SEQUENCEDIAGRAMFORGRAPHICSMODULE



3 INTERFACING MODULES:

3.1 INTERFACING BETWEEN FRAME WORK MODULE AND PARSER MODULE:

3.1.1 JAVA NATIVE INTERFACE



The Java Native Interface (JNI) is a programming framework that enables Java code running in a Java Virtual Machine (JVM) to call, and to be called by, native applications (programs specific to a hardware and operating system platform) and libraries written in other languages such as C, C++ and assembly.

Although there may be several reasons that one would want to use native code, two common justifications are *code reuse* and *performance*.

Code Reusability. Sometimes, the benefits of using Java at a level do not outweigh the overwhelming task of re-implementing currently tested and debugged C/ C++ code. Given large amounts of proven C/C++ code, it may be more cost-effective to incorporate its functionality into a Java program, rather than to port (re-implement/test/debug) the C/C++ functionality in Java.

Performance. Given that Java is interpreted (more or less), one would assume that compiled native code would perform much better. It may be useful to "hand-code" native methods where speed is critical.

3.2INTERFACING BETWEEN PARSER MODULE AND GRAPHICS MODULE

1. The C code is sent to the parser which tokenizes the code and calls the appropriate graphical interface functions.
2. The graphical interface functions generate intermediate codes.
3. Intermediate codes are pushed in the queue
4. As the user clicks the play button intermediate code executes and graphics animations is played.

JAVA CLASSES IN PARSER-GRAPHICS INTERFACING MODULE (GRAPHICAL VIRTUAL MACHINE)

1. **Accumulator Class** : Stores the results of the arithmetic and logical operations.

   getBoolean(): return boolean of the result.

   getValue(): returns value of the accumulator.

2. **Box Class** : A Memory Block, which has specified address and value in it.

   getValue(int n): returns the value in the box.

   createBox(Command comm): allocates a memory block.

   storeOperation(Operand op1,Operand op2): stores the values of operands in backend and shows visual of this operation.

3. **GraphicalInterpreter Class** : Hard coded file.

   Cache Class: Stores intermediate results of expression evaluation. Built on stack data structure.

   getValue(int n): returns the value at that index.

4. **Command Class**: We have four types of commands label, store , display, operation. It has methods to create and execute commands.

   setCommandtype setters and getters for commandtype, qualifiers, operators, type, and identifier name.

   executecommand() throws IOException: executes the command and throws exception if it is not valid.

5. **IndexQualifier Class**: Tells whether index is expression, literal or identifier. It is used for array values assignments.
6. **InterfaceClass Class**: The class which implements graphicalinterface.

   createLabelCommand(String identifier, int type,Qualifier qualifier): creates and returns the label command with the given arguments.

   createStoreCommand(Operand lhs, Operand rhs): crates and returns the store command with the given arguments.

   createOperationCommand(int operator, Operand operand1,Operand operand2): crates and returns the operation command with the given arguments.

   createDisplayCommand(String codeLine,int lineno): creates a and returns a display command with the given arguments.

7. **Memory Class**: Memory allocation tracker.

getBoxInstance(int box_num): returns the box instance for the given box number.

doRegisterBox(String identifier): assigns the box number.

getLabel(int boxno): returns identifier name of the specified memory location.

getQualifier(String identifier): returns the qualifier for the given idenfier.

getType(String identifier): returns the data type of the identifier. getValue(String

identifier): returns value of the identifier

8. **Operand Class**: We will have information of operand source. getters

and setters for value type and box number.

9. **Qualifier Class**: Will have additional information about identifier, like whether it's an array, or pointer or array of pointers variable.

getDimensions(): returns array dimensions.

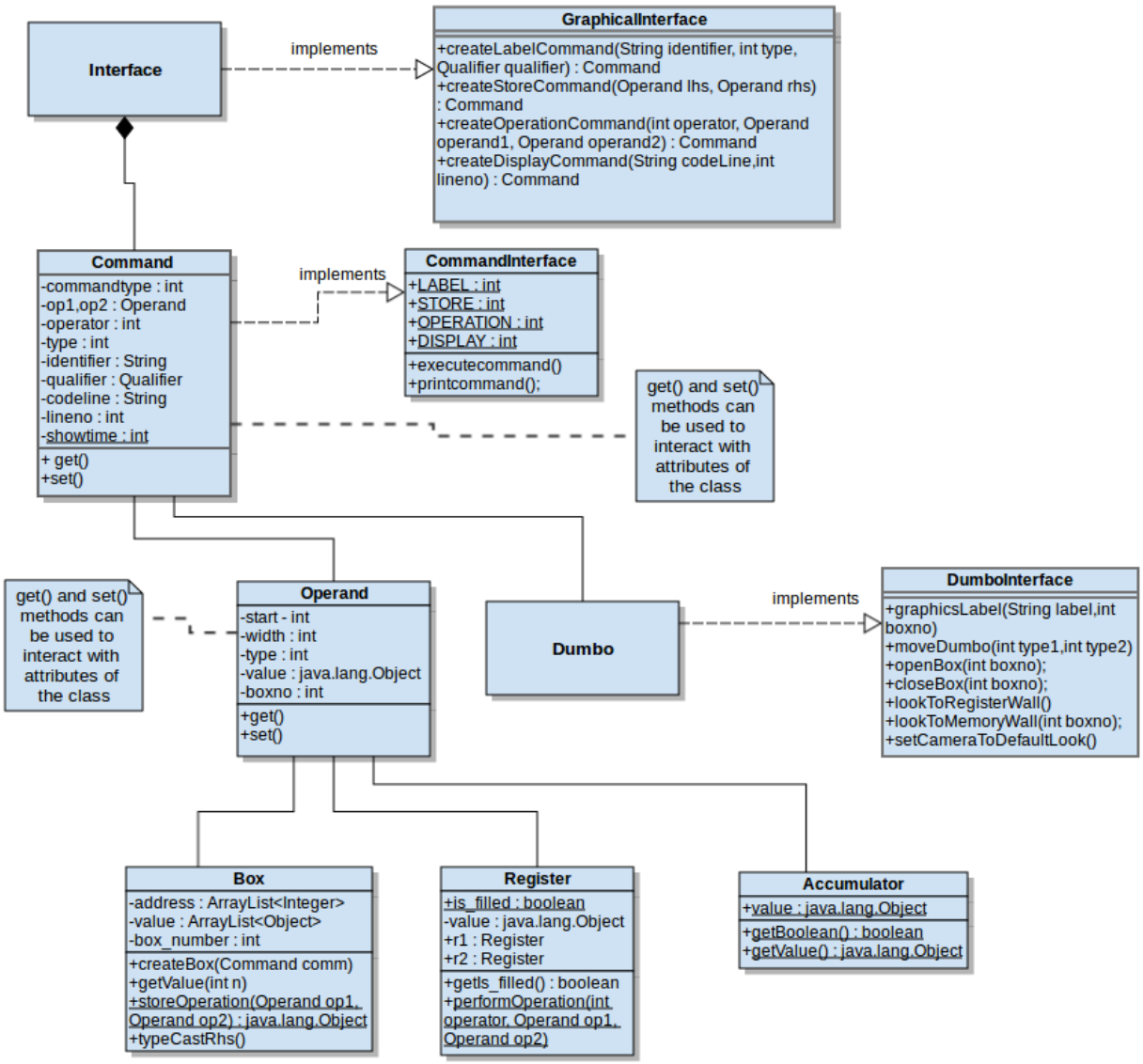
getQualifier_type(): returns qualifier type.

10. **Register Class**: One block memory location mimics register in hardware.

performOperation(int operator,Operand op1,Operand op2): performs the operation on the given arguments.

CLASS DIAGRAM : GRAPHICAL INTERFACE

Class Diagram



FUTURE SCOPE

The Graphical Interpreter can visualize the basic "C "instructions. This is the first version of the application which mainly focuses on the basics of programming language. The application is built with a flexible architecture that can be extended to implement any further developments. These developments lead to build a complete application which deals with the complex concepts of the language.

It is built to work with "C" language but can be used for various other programming languages by just changing the parser module of the existing Graphical interpreter application.

The application forms a major platform for the inquisitive students to learn the programming language from the fundamentals by browsing the already existing files or the by entering the code directly. This paves an easy way for students to develop interest in various programming languages and work towards it.

The graphical interpreter is also a teaching aid for teachers, which helps them in imparting the conceptual knowledge of the language and also develop interest among the students who lack understanding skills. It becomes a mode of analyzing the student's interest and his way of problem solving in the language.

CHALLENGES

- Generating the intermediate code
- Re-sizable panels
- Dynamic text generation
- Implementing the whole project in android platform
- Integrating all the modules

CONCLUSION

The project mainly works for enriching the knowledge among students who enter into the huge world of computers and become an integral part of that world.

The project is a little trail to uplift the process of understanding the language internal execution with the help of graphics. The graphics help us to give the proper order of execution in gaining complete overview of the underlying control of execution of the language. The application also involves modules to enhance the application. Thus, the application makes an important place in the basic understanding of programming language which enables it a good tool both for teachers and students.

BIBLIOGRAPHY

Ø Google search engine

Ø http://libgdx.badlogicgames.com

Ø steigert | Android Development: libgdx

Ø code.google.com/p/libgdx/

Ø https://github.com/libgdx/libgdx

Ø blog.xoppa.com/basic-3d-using-libgdx

Ø Blender Basics - Introduction for Beginners | Blender Cookie

Ø www.blender.org