

Parser Converter for FEM and CAD

REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENT FOR
Six Month Industrial Training

at

Testing and Consultancy Cell, GNDEC
(from July, 2014 to December, 2014)



Submitted By

Harjot Kaur

D_4 I.T

116060

1144695

Department of Information Technology
GURU NANAK DEV ENGINEERING COLLEGE
LUDHIANA-141006

To whom it may concern

I here by certify that the project Parser i.e. Converter for FEM and CAD by Harjot Kaur, University Roll No. 1144695 of Guru Nanak Dev Engineering College Ludhiana, has undergone six Monthss training from July, 2014 to December, 2014 at our organisation to fulfill the requirements for the award of six months traning of B.Tech (D4 I.T.). she worked on PARSER (CONVERTER For FEM and CAD) project during the training under the supervision of Dr. H.S. Rai (Dean, Testing and consultancy Cell, Guru Nanak Dev Engineering College). During her tenure with us we found her sincere and hard working. We wish her great success in the future.

Signature of the Student

Signature of the Supervisor

(Seal of Organisation)

Acknowledgement

I, student of Guru Nanak Dev Engineering College, Ludhiana, have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

The author is highly grateful to Dr. M.S. Saini Director, Guru Nanak Dev Engineering College, Ludhiana for providing him with the opportunity to carry out his Six Months Training at Testing and Consultancy Cell, Guru Nanak Dev Engineering College, Ludhiana.

The author would like to whole heartedly thank Dr. H.S. Rai Dean, Testing and Consultancy Cell, Guru Nanak Dev Engineering College, Ludhiana who is a vast sea of knowledge and without whose constant and never ending support and motivation, it would never have been possible to complete the project and other assignments so efficiently and effectively.

Abstract

Programming is the way through which we talk to computers and computer replies with an output. This output can be in textual form or graphical form. Humans are attached to the visual content more than he is attached to textual content. This is the reason that desktops have moved from Command Line Interface to Graphical User Interface. There's a lot of programming and brain involved behind getting these graphics on screen.

I, with curiosity of knowing the process and hence learning the same. There are many programming languages out there, which can be used for the process. I choose one of the most trusted and used language C++. C++ is a Object Oriented Programming language and working on it enhanced my general knowledge about programming and programming language. As writers tend to read a lot, singers tend to listen to music a lot a good programmer reads code a lot. This basic necessity of every programmer is fulfilled by treasure of open source softwares which provide the source for us to read, understand and learn from the source. Open source softwares are like a luxurious gift for anyone who wants to learn programming.

Converter for CAD and FEM This Project was started with the goal to make a converter to convert the files of two different platform dependent softwares i.e Windows and Unix, which was needed by the civil engineers for the "Finite Element Analysis" and also can be embedded in CAD softwares. Also, the parser for DXF file format has been developed which converts the text file into the Dxf file format, which is very frequently used in various major CAD ssoftwares like AutoCAD, SolidWorks, BRL-CAD, FreeCAD, OpenScad, etc. For the better understanding of the working and flow of parser by developing dummy parsers for some dummy file formats like ecad and xcad, and gd and gne.

The motive of the project is to reduce the time consumed and to enable fast and easy customization and even faster modifications when required as instead of typing and scal- ing, only values in the script need to be changed.

Contents

1	Introduction To Organisation	1
1.0.1	Testing and Consutancy Cell	2
1.0.2	Introduction to Project	3
1.0.3	Software requirements	5
1.0.4	Project Category	6
1.0.5	Objectives	6
1.0.6	Problem Formulation and Reorganization of Need	6
1.0.7	Existing System	7
1.0.8	Proposed System	7
1.0.9	Unique Features of the System	7
2	Requirement Analysis and Specification	8
2.0.10	Feasibility study	8
2.0.11	Software requirement specification document	8
2.0.12	Function Requirement	8
2.0.13	Non Functional Requirements	8
2.0.14	Software and Hardware Requirement	8
2.0.15	Software requirements	8
2.0.16	Hardware Requirements	9
2.0.17	Validation	9
2.0.18	Expected hurdles	9
2.0.19	Regular Expression	9
2.0.20	L ^A T _E X	9
2.0.21	Github	10
2.0.22	BNF Grammer	10
2.0.23	File format specifications	11
2.0.24	SDLC model to be used	12
3	System Design	13
3.0.25	Design Approach (Function oriented or Object oriented)	13
3.0.26	SA/SD methodology	14
3.0.27	Detail Design	15
3.0.28	System Design using design tools	16
3.0.29	User Interface Design	16
3.0.30	Methodology	17
4	Implementation ,Testing and Maintenance	19
4.0.31	Why we use Flex and Bison tools for parsing?	19
4.0.32	lex vs. flex, yacc vs. bison	19
4.0.33	flex: The Fast Lexical Analyzer	19
4.0.34	Bison	20
4.0.35	Main concepts of Bison	20

4.0.36	Languages and Context-Free Grammars	20
4.0.37	DXF File Format	21
4.0.38	General File Structure	22
4.0.39	Libraries Used in DXF Converter	22
4.0.40	dxflib	23
4.0.41	Group Codes	23
4.0.42	DXF CLASSES Section	23
4.0.43	DXF TABLES Section	23
4.0.44	Introduction to L ^A T _E X	24
4.0.45	Installing L ^A T _E X on System	24
4.0.46	Regular Expression	25
4.0.47	Github	26
4.0.48	shell scripting	27
4.0.49	Lua- Programming Language	28
4.0.50	GIMP	29
4.0.51	Features:	29
4.1	Coding standards of Language used	30
4.2	Project Scheduling	32
4.2.1	PERT Chart	32
4.3	Testing Techniques and Test Plans	33
5	Results and Discussions	35
5.0.1	User Interface Representation	35
5.0.2	Brief Description of Various Modules of the system	35
5.0.3	Snapshots of system with brief detail of each	35
5.0.4	DXF Converter	36
5.0.5	GD-GNE Converter	36
5.0.6	XCAD-ECAD Converter	39
5.0.7	Felt-Staad-Pro Converter	40
6	Conclusion and Future Scope	43
6.0.8	Conclusion:	43
6.0.9	Future Scope:	43

List of Figures

1.1	Guru Nanak Dev Engineering College	1
1.2	Testing and Consultancy Cell	2
2.1	meta classes	10
2.2	BNF Grammer in my project	11
3.1	Data Flow Diagram	17
3.2	Console User Interface	18
4.1	Richard Stallman	21
4.2	DXF working	23
4.3	L ^A T _E X Logo	24
4.4	Donald Knuth, Inventor Of T _E X typesetting system	24
4.5	meta classes	26
4.6	Github logo	26
4.7	GIMP logo	29
4.8	GIMP FEATURES	29
4.9	Pert Chart	33
5.1	DXF	36
5.2	DXF in C++	36
5.3	DXF Co-ordinates	37
5.4	Input File	37
5.5	Write DXF	37
5.6	List Of Files	38
5.7	Generate Library Files	38
5.8	Script run	38
5.9	Execute script	38
5.10	Output File	39
5.11	Two way converter	39
5.12	Output files generated by parser	39
5.13	List of files that we write	40
5.14	Output files generated by parser	40
5.15	Output file	40
5.16	Execute File	41
5.17	File Format	41
5.18	Output file	41
5.19	Terminal output	42
5.20	Snapshot option file	42
5.21	Snapshot output file	42

Chapter 1

Introduction To Organisation



Figure 1.1: Guru Nanak Dev Engineering College

I had my Six Months Industrial Training at TCC-Testing And Consultancy Cell, GNDEC Ludhiana. Guru Nanak Dev Engineering College was established by the Nankana Sahib Education Trust Ludhiana. The Nankana Sahib Education Trust i.e NSET was founded in memory of the most sacred temple of Sri Nankana Sahib, birth place of Sri Guru Nanak Dev Ji. With the mission of Removal of Economic Backwardness through Technology Shiromani Gurudwara Parbandhak Committee i.e SGPC started a Poly technical was started in 1953 and Guru Nanak Dev Engineering College was established in 1956.

NSET resolved to uplift Rural areas by admitting 70% of students from these rural areas ever year. This commitment was made to nation on 8th April, 1956, the day foundation stone of the college building was laid by Dr. Rajendra Prasad Ji, the First President of India. The College is now ISO 9001:2000 certified.

Guru Nanak Dev Engineering College campus is spread over 88 acres of prime land about 5 Km s from Bus Stand and 8 Km s from Ludhiana Railway Station on Ludhiana-Malerkotla Road. The college campus is well planned with beautifully laid out tree plantation, pathways, flowerbeds besides the well maintained sprawling lawns all around. It has beautiful building for College, Hostels, Swimming Pool, Sports and Gymnasium Hall Complex, Gurudwara Sahib, Bank, Dispensary, Post Office etc. There are two hostels for boys and one for girls with total accommodation of about 550 students. The main goal of this institute is:

- To build and promote teams of experts in the upcoming specialisations.

- To promote quality research and undertake research projects keeping in view their relevance to needs and requirements of technology in local industry.
- To achieve total financial independence.
- To start online transfer of knowledge in appropriate technology by means of establishing multipurpose resource centres.

1.0.1 Testing and Consultancy Cell

My Six Weeks Institutional Training was done by me at TCC i.e Testing And Consultancy Cell, GNDEC Ludhiana under the guidance of Dr. H.S.Rai Dean Testing and Consultancy Cell. Testing and Consultancy Cell was established in the year 1979 with a basic aim to produce quality service for technical problems at reasonable and affordable rates as a service to society in general and Engineering fraternity in particular.

Consultancy Services are being rendered by various Departments of the College to the in-



Figure 1.2: Testing and Consultancy Cell

dustry, State Government Departments and Entrepreneurs and are extended in the form of expert advice in design, testing of materials & equipment, technical surveys, technical audit, calibration of instruments, preparation of technical feasibility reports etc. This consultancy cell of the college has given a new dimension to the development programmers of the College. Consultancy projects of over Rs. one crore are completed by the Consultancy cell during financial year 2009-10.

Ours is a pioneer institute providing Consultancy Services in the States of Punjab, Haryana, Himachal, J&K and Rajasthan. Various Major Clients of the Consultancy Cell are as under:

- Larson & Turbo.
- Multi National Companies like AFCON & PAULINGS.
- Power Grid Corporation of India.

- National Building Construction Co.
- Punjab State Electricity Board.
- Punjab Mandi Board.
- Punjab Police Housing Corporation.
- National Fertilizers Ltd.

1.0.2 Introduction to Project

Converter for CAD and FEM This Project was started with the goal to make a converter to convert the files of two different platform dependent softwares i.e Windows and Unix, which was needed by the civil engineers for the Finite Element Analysis and also can be embedded in CAD softwares. Also, the parser for DXF file format has been developed which converts the text file into the Dxf file format, which is very frequently used in various major CAD softwares like AutoCAD, SolidWorks, BRL-CAD, FreeCAD, OpenScad, etc. For the better understanding of the working and flow of parser by developing dummy parsers for some dummy file formats like ecad and xcad, and gd and gne.

Various tools used to develop the project are:

- GCC Compiler
- Flex/lex
- Bison/yacc
- DXF file format
- C++
- C language

GCC Compiler:

A compiler is a software program which converts the code written in any programming language i.e high-level language into a low-level language code. A compiler is a translator whose source language is a high-level language and whose object language is close to the machine language of an actual computer. The typical compiler consists of several phases each of which passes its output to the next phase

Phases of Compiler:

- Lexical analysis: The lexical phase (scanner) groups characters into lexical units or tokens. The input to the lexical phase is a character stream. The output is a stream of tokens. Regular expressions are used to define the tokens recognized by a scanner (or lexical analyzer). The scanner is implemented as a finite state machine. Lex and Flex are tools for generating scanners: programs which recognize lexical patterns in text. Flex is a faster version of Lex.

- **Syntax analysis:** The parser groups tokens into syntactical units. The output of the parser is a parse tree representation of the program. Context-free grammars are used to define the program structure recognized by a parser. The parser is implemented as a push-down automata.
- **Semantic analysis:** The semantic analysis phase analyzes the parse tree for context-sensitive information often called the static semantics. The output of the semantic analysis phase is an annotated parse tree. Attribute grammars are used to describe the static semantics of a program.
- **Intermediate Code Generation:** This phase is often combined with the parser. During the parse, information concerning variables and other objects is stored in a symbol table. The information is utilized to perform the context-sensitive checking.
- **Code optimization:** The optimizer applies semantics preserving transformations to the annotated parse tree to simplify the structure of the tree and to facilitate the generation of more efficient code.
- **Code generation:** The code generator transforms the simplified annotated parse tree into object code using rules which denote the semantics of the source language. The code generator may be integrated with the parser.

Parser:

Parsing or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, according to the rules of a formal grammar. A parser is a program which determines if its input is syntactically valid and determines its structure. Parsers may be hand written or may be automatically generated by a parser generator from descriptions of valid syntactical structures. The descriptions are in the form of a context-free grammar. Yacc is a program which given a context-free grammar, constructs a C program which will parse input according to the grammar rules.

Parser invented by Donald Knuth in 1965. Parser breaks data into smaller elements for easy translation into another language. A parser takes input in the form of a sequence of tokens and usually builds a data structure in the form of a parse tree or an abstract syntax tree.

Flex:

The fast lexical analyser. It is the modern replacement for the classic Lex, which was developed by the Bell Laboratories in the 1970s. Flex was originally written by Jef Poskanzer; Vern Paxson and Van Jacobson have considerably improved it. Flex is a tool for generating scanners. A scanner, sometimes called a tokenizer, is a program which recognizes lexical patterns in text. The flex program reads user-specified input files, for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code, called rules. Flex generates a C source file named, "lex.yy.c", which defines the function yylex(). The file "lex.yy.c" can be compiled and linked to produce an executable. When the executable is run, it analyzes its input for occurrences of text matching the regular expressions for each rule. Whenever it finds a match, it executes the corresponding C code. Flex and

Bison files have three sections:

- The first is sort of "control" information
- The second is the actual token/grammar definitions
- The last is C/C++ code to be copied verbatim to the output

Bison:

Yacc and Bison are tools for generating parsers: programs which recognize the structure grammatical structure of programs. Bison is a faster version of Yacc. Bison was originally written by Robert Corbett in 1988. The sections on Yacc/Bison are a condensation and extension of the document BISON the Yacc-compatible Parser Generator by Charles Donnelly and Richard Stallman. Like Flex, Bison file is also divided into three sections divided by `%%`. Bison reads a specification of the user-specified grammar, warns about any parsing ambiguities, and generates a parser (either in C, C++, or Java) which reads sequences of tokens and decides whether the sequence conforms to the syntax specified by the grammar. A input file for Bison is of the form:

- C and parser declarations
- `%%`
- Grammar rules and actions
- `%%`
- C subroutines

DXF File Format

DXF stands for Drawing Exchange Format. Files that contain the .dxf file extension contain CAD vector image files. The DXF file format is similar to the DWG file format, but DXF files are ASCII based and are therefore more compatible with other computer applications. The DXF file format was developed as an exchange format for the CAD files that are created by computer aided drafting software applications. The file format was initially introduced in December of 1982 as a part of AutoCAD 1.0. The file format was meant to provide an exact representation of the data in the standard AutoCAD file format.

1.0.3 Software requirements

- Operating System: Linux/Windows
- Programming Language: C, C++

1.0.4 Project Category

Converter for CAD and FEM This Project was started with the goal to make a converter to convert the files of two different platform dependent softwares i.e Windows and Unix, which was needed by the civil engineers for the Finite Element Analysis and also can be embedded in CAD softwares. Also, the parser for DXF file format has been developed which converts the text file into the Dxf file format, which is very frequently used in various major CAD softwares like AutoCAD, SolidWorks, BRL-CAD, FreeCAD, OpenScad, etc. For the better understanding of the working and flow of parser by developing dummy parsers for some dummy file formats like ecad and xcad, and gd and gne.

What is Internet based Application?

A Internet based application is any software that runs in a web browser. It is created in a browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and relies on a web browser to render the application. Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client, sometimes called a thin client. The ability to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility.

What is Industry automation?

Industrial automation is the use of control systems, such as computers or robots, and information technologies for handling different processes and machineries in an industry to replace a human being. It is the second step beyond mechanization in the scope of industrialization.

1.0.5 Objectives

At present, no such software exists which convert the file format of STAAD-PRO and FELT softwares. Both of these are FEM softwares which are used by the Civil Engineers for the purpose of Finite Element Analysis. As both of these softwares run on different platforms, the users feel difficulty to analyze the output files of both the softwares together. This encouraged us to develop such a software which will be very useful to the users. With objective to develop the software which can easily convert the two file formats into each other and can gather the information of both the softwares, which is of great use for the civil engineers.

1.0.6 Problem Formulation and Reorganization of Need

The STAAD-PRO and FELT are both the FEM softwares which are based on different platforms. So users have to switch between the Operating Systems to analyze the files created by both the softwares. As there is no such software which can read both the files and convert into each others. Both the softwares are of great use of Civil Engineers. So a new software has been developed to ease the Finite Element Analysis of the Civil Engineers.

1.0.7 Existing System

The STAAD-PRO and FELT are both the FEM softwares which are based on different platforms. So users have to switch between the Operating Systems to analyze the files created by both the softwares. As there is no such software which can read both the files and convert into each others. Both the softwares are of great use of Civil Engineers. So a new software has been developed to ease the Finite Element Analysis of the Civil Engineers.

1.0.8 Proposed System

The STAAD-PRO and FELT are both the FEM softwares which are based on different platforms. So users have to switch between the Operating Systems to analyze the files created by both the softwares. As there is no such software which can read both the files and convert into each others. Both the softwares are of great use of Civil Engineers. So a new software has been developed to ease the Finite Element Analysis of the Civil Engineers.

1.0.9 Unique Features of the System

The STAAD-PRO and FELT are both the FEM softwares which are based on different platforms. So users have to switch between the Operating Systems to analyze the files created by both the softwares. As there is no such software which can read both the files and convert into each others. Both the softwares are of great use of Civil Engineers. So a new software has been developed to ease the Finite Element Analysis of the Civil Engineers.

Chapter 2

Requirement Analysis and Specification

2.0.10 Feasibility study

- **Technical Feasibility** As this whole project is based on C++ programming language and Flex and Bison parsing tools, technical feasibility of this project revolves around the technical boundaries and limitations of the C++ and these tools i.e. Flex and Bison. But as Flex and Bison is much powerful parsing tool and C++ is powerful programming language, so these languages are perfect to design the software under this project.
- **Economic Feasibility** Almost all the softwares used in this project are Open source and the software released under this project are Open source too and are released under GNU GPLv3 (General Public Licence). So this project is fully economic feasible.
- **Operational Feasibility** This project is also operational feasible as it automates the work of converting one file format into another which not only saves time. It also converts one operating system files into another. In our Project Flex and Bison tools convert Felt software files i.e. Linux based into Staad-Pro software i.e. window based. It makes conversion easily and save time.

2.0.11 Software requirement specification document

2.0.12 Function Requirement

A functional requirement defines the functions of a software system. In this software there are tools Flex and Bison are used to develop the software. Flex and Bison tools to develop software.

2.0.13 Non Functional Requirements

The fast lexical analyser. It is the modern replacement for the classic Lex, which was developed by the Bell Laboratories in the 1970s. Flex was originally written by Jef Poskanzer; Vern Paxson and Van Jacobson have considerably improved it. Flex is a tool for generating scanners. A scanner, sometimes called a tokenizer, is a program which recognizes lexical patterns in text. The flex program reads user-specified input files, for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code, called rules. Flex generates a C source file named, "lex.yy.c", which defines the function yylex().

The file "lex.yy.c" can be compiled and linked to produce an executable. When the executable is run, it analyzes its input for occurrences of text matching the regular expressions for each rule. Whenever it finds a match, it executes the corresponding C code. Flex and Bison files have three sections:

- The first is sort of "control" information
- The second is the actual token/grammar definitions
- The last is C/C++ code to be copied verbatim to the output

2.0.14 Software and Hardware Requirement

2.0.15 Software requirements

- Operating System: Linux/Windows
- Tools: Flex, Bison, DXF file format
- Programming Language: C, C++

2.0.16 Hardware Requirements

Hardware requirement of this project is any Desktop or Laptop machine for local use or a Server with minimum available configuration to make Project globally available. Hardware specifications of the machine used depends upon the hardware requirements of the Operating System installed on it. As such there are no special hardware requirements of this project.

2.0.17 Validation

For every software and technology, validations are must. In our parser, This software is command line based. So, there is no any database used. But we use files and validations on these files are:

- When we execute files and generate output. If we give wrong input and output file names. It shows error message which is required validation for our software.
- Flex tool is the scanner tool which divides input into small parts called tokens. In Flex, regular expressions are used as input. These tokens or regex are valid for output file. If regex are not correct, it shows message error.
- Bison or yacc is a tool used for parsing, In this tool BNF grammar is used. Grammar is the main part of the parser. If we didn't write correct grammar. There was a problem in conversion of files.
- In DXF converter, we use two different libraries. The use of libraries is for specific task. If libraries are not included then task does not take place.

2.0.18 Expected hurdles

2.0.19 Regular Expression

A regular expression processor translates a regular expression into a nondeterministic finite automaton (NFA), which is then made deterministic and run on the target text string to recognize substrings that match the regular expression. Regular expressions are so useful in computing that the various systems to specify regular expressions have evolved to provide both a basic and extended standard for the grammar and syntax; modern regular expressions heavily augment the standard. Regular expression processors are found in several search engines, search and replace dialogs of several word processors and text editors, and in the command lines of text processing utilities, such as sed and AWK.

POSIX	Non-standard	Perl/Tcl	Vim	ASCII	Description
[[:alnum:]]				[A-Za-z0-9]	Alphanumeric characters
	[[:word:]]	\w	\w	[A-Za-z0-9_]	Alphanumeric characters plus "_"
		\W	\W	[^A-Za-z0-9_]	Non-word characters
[[:alpha:]]			\a	[A-Za-z]	Alphabetic characters
[[:blank:]]		\s	[\t]		Space and tab
		\b	\< \>	(?<=\W) (?=\w) (?<=\w) (?=\W)	Word boundaries
[[:cntrl:]]				[\x00-\x1F\x7F]	Control characters
[[:digit:]]		\d	\d	[0-9]	Digits
		\D	\D	[^0-9]	Non-digits
[[:graph:]]				[\x21-\x7E]	Visible characters
[[:lower:]]			\l	[a-z]	Lowercase letters
[[:print:]]			\p	[\x20-\x7E]	Visible characters and the space character
[[:punct:]]				[!\"#\$%&'()*+,-./:;<=>?@^_`{ }~]	Punctuation characters
[[:space:]]		\s	\s	[\t\r\n\v\f]	Whitespace characters
		\S		[^ \t\r\n\v\f]	Non-whitespace characters
[[:upper:]]			\u	[A-Z]	Uppercase letters
[[:xdigit:]]			\x	[A-Fa-f0-9]	Hexadecimal digits

Figure 2.1: meta classes

2.0.20 L^AT_EX

L^AT_EX is most widely used by mathematicians, scientists, engineers, philosophers, linguists, economists and other scholars in academia. As a primary or intermediate format, e.g., translating DocBook and other XML-based formats to PDF, L^AT_EX is used because of the high quality of typesetting achievable by T_EX. The typesetting system offers programmable desktop publishing features and extensive facilities for automating most aspects of typesetting and desktop publishing, including numbering and cross-referencing, tables and figures, page layout and bibliographies.

L^AT_EX is intended to provide a high-level language that accesses the power of T_EX. L^AT_EX essentially comprises a collection of T_EX macros and a program to process L^AT_EX documents. Because the T_EX formatting commands are very low-level, it is usually much simpler for end-users to use L^AT_EX.

2.0.21 Github

The Git feature that really makes it stand apart from nearly every other Source Code Management (SCM) out there is its branching model. Git allows and encourages you to have

multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.

2.0.22 BNF Grammer

In computer science, BNF (Backus Normal Form or BackusNaur Form) is one of the two main notation techniques for context-free grammars, often used to describe the syntax of languages used in computing, such as computer programming languages, document formats, instruction sets and communication protocols; the other main technique for writing context-free grammars is the van Wijngaarden form. They are applied wherever exact descriptions of languages are needed: for instance, in official language specifications, in manuals, and in textbooks on programming language theory.

In order to write a parser, we need some way to describe the rules the parser uses to turn a sequence of tokens into a parse tree. The most common kind of language that computer parsers handle is a context-free grammar (CFG). The standard form to write down a CFG is Backus-Naur Form (BNF), created around 1960 to describe Algol 60 and named after two members of the Algol 60 committee. In flt-std Parser we have used BNF grammer.

Each line is a rule that says how to create a branch of the parse tree. In BNF, ::= can be read is a or becomes, and — is or, another way to create a branch of the same kind. The name on the left side of a rule is a symbol or term. By convention, all tokens are considered to be symbols, but there are also symbols that are not tokens.

Useful BNF is invariably quite recursive, with rules that refer to themselves directly or indirectly. These simple rules can match an arbitrarily complex sequence of additions and multiplications by applying them recursively.

```
%%
/**
 * Grammar rules
 */
converter:
    title jc nl end
    ;
title:
    DESCRIPTION { w.write_title(); }
    ;
jc:
    JC coord { mode = 'b'; }
    ;
nl:
    MI coord { }
    ;
coord:
    digit SEMICOLON
    | digit SEMICOLON coord
    ;
digit:
    digit DIGIT {
        if (mode == 'n') {
            w.store_values('n', $2, n_i);
            n_i++;
        } else {
            w.store_values('b', $2, b_i);
            b_i++;
        }
    }
    | DIGIT { }
    ;
end:
    FINISH { w.write_data(); w.write_end_file(); }
    ;
%%
```

Figure 2.2: BNF Grammer in my project

2.0.23 File format specifications

A Drawing Interchange File is simply an ASCII text file with a file type of .dxf and specially formatted text. The overall organization of a DXF file is as follows:

1. **HEADER section** General information about the drawing is found in this section of the DXF file. Each parameter has a variable name and an associated value.
2. **TABLES section** This section contains definitions of named items.
 - Linetype table (LTYPE)
 - Layer table (LAYER)
 - Text Style table (STYLE)
 - View table (VIEW)
 - User Coordinate System table (UCS)
 - Viewport configuration table (VPOR)
 - Dimension Style table (DIMSTYLE)
 - Application Identification table (APPID)
 - **BLOCKS section** This section contains Block Definition entities describing the entities that make up each Block in the drawing.
 - **ENTITIES section** This section contains the drawing entities, including any Block References.
 - **END OF FILE**

2.0.24 SDLC model to be used

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

- **Identification**

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase. This also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral the product is deployed in the identified market.

- **Design**

Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

- **Construct or Build**

Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback. Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to customer for feedback.

- **Evaluation and Risk Analysis**

Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

Chapter 3

System Design

3.0.25 Design Approach (Function oriented or Object oriented)

A software design document (SDD) is a written description of a software product, that a software designer writes in order to give a software development team overall guidance to the architecture of the software project. An SDD usually accompanies an architecture diagram with pointers to detailed feature specifications of smaller pieces of the design. Practically, a design document is required to coordinate a large team under a single vision. A design document needs to be a stable reference, outlining all parts of the software and how they will work. The document is commanded to give a fairly complete description, while maintaining a high-level view of the software.

There are two kinds of design documents called HLDD (high-level design document) and LLDD (low-level design document). The SDD contains the following documents:

- The data design describes structures that reside within the software. Attributes and relationships between data objects dictate the choice of data structures.
- The architecture design uses information flowing characteristics, and maps them into the program structure. The transformation mapping method is applied to exhibit distinct boundaries between incoming and outgoing data. The data flow diagrams allocate control input, processing and output along three separate modules.
- The interface design describes internal and external program interfaces, as well as the design of human interface. Internal and external interface designs are based on the information obtained from the analysis model.
- The procedural design describes structured programming concepts using graphical, tabular and textual notations. These design mediums enable the designer to represent procedural detail, that facilitates translation to code. This blueprint for implementation forms the basis for all subsequent software engineering worked.

System Design : Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering.

1. **External design** -External design consists of conceiving, planning out and specifying the externally observable characteristics of the software product. These characteristics include user displays or user interface forms and the report formats, external data sources and the functional characteristics, performance requirements etc. External design begins during the analysis phase and continues into the design phase.
2. **Logical design** - The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modeling, which involves a simplistic (and sometimes graphical) representation of an actual system. In the context of systems design, modeling can undertake the following forms, including:
 Data flow diagrams Entity Relationship Diagrams
3. **Physical design** - The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is input into a system, how it is verified/authenticated, how it is processed, and how it is displayed as output.

It follows a object oriented design approach. object-oriented design techniques are very popular, currently in use in many software development organizations.

Object-oriented analysis and design is a popular technical approach to analyzing, designing an application, system, or business by applying the object-oriented paradigm and visual modeling throughout the development life cycles to foster better stakeholder communication and product quality.

3.0.26 SA/SD methodology

It has essential features of several important function-oriented design methodologies. If you need to use any specific design methodology later on, you can do so easily with small additional effort.

Structured Analysis and Design Technique (SADT) is a diagrammatic notation designed specifically to help people describe and understand systems. It offers building blocks to represent entities and activities, and a variety of arrows to relate boxes. These boxes and arrows have an associated informal semantics. SADT can be used as a functional analysis tool of a given process, using successive levels of details. The SADT method not only allows one to define user needs for IT developments, which is often used in the industrial Information Systems, but also to explain and present an activity's manufacturing processes and procedures.

Object-oriented design is the process of planning a system of interacting objects for the purpose of solving a software problem. It is one approach to software design. An object contains encapsulated data and procedures grouped together to represent an entity. The 'object interface', how the object can be interacted with, is also defined.

An object-oriented program is described by the interaction of these objects. Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis. What follows is a description of the class-based subset of object-oriented design, which does not include object prototype-based approaches where objects are not typically obtained by instantiating classes but by cloning other (prototype) objects. Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as state and dynamic models of the system under design.

3.0.27 Detail Design

We basically describe the functionality of the system internally. The internal design describes how data is flowing from database to the user and how they both are internally connected. For this reason we can show the design of the system in detailed manner by many ways:

Flowchart: A flowchart is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation can give a step-by-step solution to a given problem. Process operations are represented in these boxes, and arrows connecting them represent flow of control. Data flows are not typically represented in a flowchart, in contrast with data flow diagrams; rather, they are implied by the sequencing of operations. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

Design standards must be set at the start of the DD phase by project management to co-ordinate the collective efforts of the team. This is especially necessary when development team members are working in parallel. The developers must first complete the top-down decomposition of the software started in the AD phase (DD02) and then outline the processing to be carried out by each component. Developers must continue the structured approach and not introduce unnecessary complexity. They must build defences against likely problems.

Developers should verify detailed designs in design reviews, level by level. Review of the design by walkthrough or inspection before coding is a more efficient way of eliminating design errors than testing. The developer should start the production of the user documentation early in the DD phase. This is especially important when the HCI component is significantly large writing the SUM forces the developer to keep the users view continuously in mind.

Reuse of the software: Software reuse questions can arise at all stages of design. In the DD phase decisions may have been taken to reuse software for all or some major components, such as:

- Application generators;
- Database management systems;
- Human-computer interaction utilities;
- Mathematical utilities;
- Graphical utilities

The high-level design will normally have been identified during the AD phase. Detailed designs may have to be prototyped in the DD phase to find out which designs best meet the requirements.

The feasibility of a novel design idea should be checked by prototyping. This ensures that an idea not only works, but also that it works well enough to meet non-functional requirements for quality and performance.

3.0.28 System Design using design tools

A Data Flow Diagram (DFD) is a graphical representation of the flow of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of process or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

3.0.29 User Interface Design

This project is based on Command-Line based. In this approach not User Interface Design needs. Console Mode or command-Line based design is Text mode programs are program which output is only made out of text. In Windows terminology, they are called CUI (Console User Interface) executables, by opposition to GUI (Graphical User Interface) executables. Win32 API provide a complete set of APIs to handle this situation, which goes from basic features like text printing, up to high level functionalities (like full screen editing, color support, cursor motion, mouse support), going through features like line editing or raw/cooked input stream support.

Command-line interfaces to computer operating systems are less widely used by casual computer users, who favor graphical user interfaces. Command-line interfaces are often

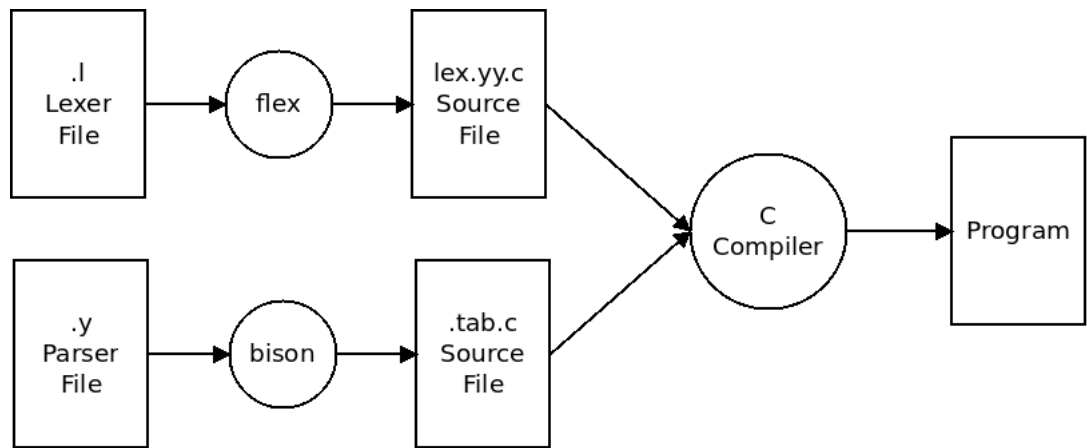


Figure 3.1: Data Flow Diagram

preferred by more advanced computer users, as they often provide a more concise and powerful means to control a program or operating system.

3.0.30 Methodology

Step 1: Researching the needs of the students and administration and finding out the facilities required to develop project.

Step 2: Documenting the needs and then preparing the layout for the project, and deciding the various modules to be included in the software.

Step 3: DFDs will be prepared showing various interactions between users and the system.

Step 4: Selecting the technology for developing the project and installing the required tools for developing the project. We will install Net beans-IDE and My SQL Server.

Step 5: Developing the front end- All the forms required to get users information, informative forms etc. will be developed in html.

Step 6: Developing the back end- the tables necessary to store the data relating to customers and tables containing the data of the organization will be developed in My SQL. Tables can be updated from time to time. Efforts will be made to ensure no redundancy.

Step 7: Connectivity between front end and back end- will be used to connect front end and back end, this will result in complete running project where user can interact with system and can enter details which will saved at back end. The details already entered can be updated.

Step 8: Testing the system by running it and by entering data and retrieving it.

Chapter 4

Implementation ,Testing and Maintenance

4.0.31 Why we use Flex and Bison tools for parsing?

Flex and Bison are aging unix utilities that help you write very fast parsers for almost arbitrary file formats. Formally, they implement Look-Ahead-Left-Right (as opposed to "recursive descent") parsing of non-ambiguous context-free (as opposed to "natural language") grammars.

Why should you learn the Flex/Bison pattern syntax when you could just write your own parser? Well, several reasons. First, Flex and Bison will generate a parser that is virually guaranteed to be faster than anything you could write manually in a reasonable amount of time. Second, updating and fixing Flex and Bison source files is a lot easier than updating and fixing custom parser code. Third, Flex and Bison have mechanisms for error handling and recovery, which is something you definitely don't want to try to bolt onto a custom parser. Finally, Flex and Bison have been around for a long time, so they far freer from bugs than newer code.

4.0.32 lex vs. flex, yacc vs. bison

In addition to hearing about "flex and bison", you will also hear about "lex and yacc". "lex and yacc" are the original tools; "flex and bison" are their almost completely compatible newer versions. Only very old code uses lex and yacc; most of the world has moved on to Flex and Bison.

All four of the above are C-based tools; they're written in C, but more important their output is C code. However, my project was in C++ – so this is also a tutorial on how to use C++ with Flex and Bison!

4.0.33 flex: The Fast Lexical Analyzer

Flex is a tool for generating scanners. A scanner, sometimes called a tokenizer, is a program which recognizes lexical patterns in text. The flex program reads user-specified input files, or its standard input if no file names are given, for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code,

called rules. Flex generates a C source file named, "lex.yy.c", which defines the function `yylex()`. The file "lex.yy.c" can be compiled and linked to produce an executable. When the executable is run, it analyzes its input for occurrences of text matching the regular expressions for each rule. Whenever it finds a match, it executes the corresponding C code. Flex and Bison files have three sections:

- The first is sort of "control" information
- The second is the actual token/grammar definitions
- The last is C/C++ code to be copied verbatim to the output

Flex file can be compiled by running this:

1. *flexnameOfFlexFile*
2. *g + lex.yy.c - lfl - ooutfile*

4.0.34 Bison

Bison is a general-purpose parser generator that converts an annotated context-free grammar into a deterministic LR or generalized LR (GLR) parser employing LALR(1) parser tables. As an experimental feature, Bison can also generate IELR(1) or canonical LR(1) parser tables. Once you are proficient with Bison, you can use it to develop a wide range of language parsers, from those used in simple desk calculators to complex programming languages.

Bison is upward compatible with Yacc: all properly-written Yacc grammars ought to work with Bison with no change. Anyone familiar with Yacc should be able to use Bison with little trouble. You need to be fluent in C or C++ programming in order to use Bison or to understand this manual. Java is also supported as an experimental feature.

Bison was written originally by Robert Corbett. Richard Stallman made it Yacc-compatible. Wilfred Hansen of Carnegie Mellon University added multi-character string literals and other features.

4.0.35 Main concepts of Bison

4.0.36 Languages and Context-Free Grammars

In order for Bison to parse a language, it must be described by a context-free grammar. This means that you specify one or more syntactic groupings and give rules for constructing them from their parts. For example, in the C language, one kind of grouping

is called an expression. One rule for making an expression might be, An expression can be made of a minus sign and another expression. Another would be, An expression can be an integer. As you can see, rules are often recursive, but there must be at least one rule which leads out of the recursion.

The most common formal system for presenting such rules for humans to read is



Figure 4.1: Richard Stallman

Backus- Naur Form or BNF, which was developed in order to specify the language Algol 60. Any grammar expressed in BNF is a context-free grammar. The input to Bison is essentially machine-readable BNF.

In the formal grammatical rules for a language, each kind of syntactic unit or grouping is named by a symbol. Those which are built by grouping smaller constructs according to grammatical rules are called nonterminal symbols; those which can't be subdivided are called terminal symbols or token types. We call a piece of input corresponding to a single terminal symbol a token, and a piece corresponding to a single nonterminal symbol a grouping.

4.0.37 DXF File Format

The task of reading through the DXF library and rebuilding it, requires a great effort. At this stage I can't think of re-writing the library as I'm new to it. Today, I have gone through the basics of DXF library. DXF is a file format used for importing and exporting 2-d drawings.

DXF stands for Drawing Exchange Format. Files that contain the .dxf file extension contain CAD vector image files. The DXF file format is similar to the DWG file format, but DXF files are ASCII based and are therefore more compatible with other computer applications.

The DXF file format was developed as an exchange format for the CAD files that are created by computer aided drafting software applications. The file format was initially introduced in December of 1982 as a part of AutoCAD 1.0. The file format was meant to provide an exact representation of the data in the standard AutoCAD file format.

4.0.38 General File Structure

A Drawing Interchange File is simply an ASCII text file with a file type of .dxf and specially formatted text. The overall organization of a DXF file is as follows:

1. HEADER section General information about the drawing is found in this section of the DXF file. Each parameter has a variable name and an associated value.
2. TABLES section This section contains definitions of named items.
 - Linetype table (LTYPE)
 - Layer table (LAYER)
 - Text Style table (STYLE)
 - View table (VIEW)
 - User Coordinate System table (UCS)
 - Viewport configuration table (VPOR)
 - Dimension Style table (DIMSTYLE)
 - Application Identification table (APPID)
 - BLOCKS section This section contains Block Definition entities describing the entities that make up each Block in the drawing.
 - ENTITIES section This section contains the drawing entities, including any Block References.
 - END OF FILE

4.0.39 Libraries Used in DXF Converter

- libdxfrw 0.5.7
- dxflib

4.0.40 dxflib

dxflib is an open source C++ library mainly for parsing DXF files. QCAD, CAM Expert dxflib to import DXF files. dxflib can also write DXF files, but you need to have good knowledge of the DXF format to produce valid output. dxflib parses DXF files and calls functions in your class. In those functions you can for example add the entities to an entity container.

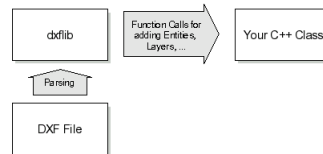


Figure 4.2: DXF working

4.0.41 Group Codes

The group code is an integer which implicitly defines the value type and acts as a key for the value. In ASCII DXF, group codes and values are written in a single line each. Each group code listed in the DXF reference topics is presented by a numeric group code value and a description. All group codes can apply to DXF files, applications (AutoLISP or ObjectARX), or both. When the description of a code is different for applications and DXF files (or applies to only one or the other), the description is preceded by the following indicators:

- APP. Application-specific description.
- DXF. DXF file-specific description.

4.0.42 DXF CLASSES Section

The CLASSES section in DXF files holds the information for application-defined classes whose instances appear in the BLOCKS, ENTITIES, and OBJECTS sections of the database. It is assumed that a class definition is permanently fixed in the class hierarchy.

4.0.43 DXF TABLES Section

The group codes described in this chapter are found in DXF TM files and used by applications. The TABLES section contains several tables, each of which can contain a variable number of entries. These codes are also used by AutoLISP and ObjectARX applications in entity definition lists.



Figure 4.3: L^AT_EX Logo

4.0.44 Introduction to L^AT_EX

L^AT_EX, I had never heard about this term before doing this project, but when I came to know about it's features, it is just excellent. L^AT_EX(pronounced /letk/, /letx/, /ltx/, or /ltk/) is a document markup language and document preparation system for the T_EX typesetting program. Within the typesetting system, its name is styled as L^AT_EX.



Figure 4.4: Donald Knuth, Inventor Of T_EX typesetting system

Within the typesetting system, its name is styled as L^AT_EX. The term L^AT_EX refers only to the language in which documents are written, not to the editor used to write those documents. In order to create a document in L^AT_EX, a .tex file must be created using some form of text editor. While most text editors can be used to create a L^AT_EX document, a number of editors have been created specifically for working with L^AT_EX.

L^AT_EX is most widely used by mathematicians, scientists, engineers, philosophers, linguists, economists and other scholars in academia. As a primary or intermediate format, e.g., translating DocBook and other XML-based formats to PDF, L^AT_EX is used because of the high quality of typesetting achievable by T_EX. The typesetting system offers programmable desktop publishing features and extensive facilities for automating most aspects of typesetting and desktop publishing, including numbering and cross-referencing, tables and figures, page layout and bibliographies.

L^AT_EX is intended to provide a high-level language that accesses the power of T_EX. L^AT_EX essentially comprises a collection of T_EX macros and a program to process L^AT_EXdocuments. Because the T_EX formatting commands are very low-level, it is usually much simpler for end-users to use L^AT_EX.

4.0.45 Installing L^AT_EX on System

Installation of L^AT_EX on personal system is quite easy. As i have used L^AT_EX on Ubuntu 13.04 so i am discussing the installation steps for Ubuntu 13.04 here:

- Go to terminal and type

sudo apt-get install texlive-full

- Your Latex will be installed on your system and you can check for manual page by typing.

man latex

in terminal which gives manual for latex command.

- To do very next step now one should stick this to mind that the document which one is going to produce is written in any type of editor whether it may be your most common usable editor Gedit or you can use vim by installing first vim into your system using command.

sudo apt-get install vim

- After you have written your document it is to be embedded with some set of commands that Latex uses so as to give a structure to your document. Note that whenever you wish your document to be looked into some other style just change these set of commands.

- When you have done all these things save your piece of code with .tex format say test.tex. Go to terminal and type

latex path of the file test.tex Or pdflatex path of the file test.tex

eg: pdflatex test.tex

for producing pdf file simultaneously.

After compiling it type command

evince filename.pdf

eg: evince test.pdf

To see output pdf file.

4.0.46 Regular Expression

A regular expression processor translates a regular expression into a nondeterministic finite automaton (NFA), which is then made deterministic and run on the target text string to recognize substrings that match the regular expression. Regular expressions are so useful in computing that the various systems to specify regular expressions have evolved to provide both a basic and extended standard for the grammar and syntax; modern regular expressions heavily augment the standard. Regular expression processors are found in several search engines, search and replace dialogs of several word processors and text editors, and in the command lines of text processing utilities, such as sed and AWK.

POSIX	Non-standard	Perl/Tcl	Vim	ASCII	Description
[:alnum:]				[A-Za-z0-9]	Alphanumeric characters
[:word:]		\w	\w	[A-Za-z0-9_]	Alphanumeric characters plus "_"
		\W	\W	[^A-Za-z0-9_]	Non-word characters
[:alpha:]			\a	[A-Za-z]	Alphabetic characters
[:blank:]			\s	[\t]	Space and tab
		\b	\< \>	(?<=\W) (?=\w) (?<=\w) (?=\W)	Word boundaries
[:cntrl:]				[\x00-\x1F\x7F]	Control characters
[:digit:]		\d	\d	[0-9]	Digits
		\D	\D	[^0-9]	Non-digits
[:graph:]				[\x21-\x7E]	Visible characters
[:lower:]			\l	[a-z]	Lowercase letters
[:print:]			\p	[\x20-\x7E]	Visible characters and the space character
[:punct:]				[! " # \$ % & ' () * + , - . : ; < = > ? @ [\ ^ _ ` { } ~ -]	Punctuation characters
[:space:]		\s	_s	[\t\r\n\v\f]	Whitespace characters
		\S		[^ \t\r\n\v\f]	Non-whitespace characters
[:upper:]			\u	[A-Z]	Uppercase letters
[:xdigit:]			\x	[A-Fa-f0-9]	Hexadecimal digits

Figure 4.5: meta classes

4.0.47 Github

The Git feature that really makes it stand apart from nearly every other Source Code Management (SCM) out there is its branching model. Git allows and encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds. This means that you can do things like:



Figure 4.6: Github logo

- Frictionless Context Switching. Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, and merge it in.
- Role-Based Codelines. Have a branch that always contains only what goes to production, another that you merge work into for testing, and several smaller ones for day to day work.
- Feature Based Workflow. Create new branches for each new feature you're working on so you can seamlessly switch back and forth between them, then delete each branch when that feature gets merged into your main line.

- Disposable Experimentation. Create a branch to experiment in, realize its not going to work, and just delete it - abandoning the work with nobody else ever seeing it (even if you've pushed other branches in the meantime). Notably, when you push to a remote repository, you do not have to push all of your branches. There are ways to accomplish some of this with other systems, but the work involved is much more difficult and error-prone. Git makes this process incredibly easy and it changes the way most developers work when they learn it.

Firstly, You create a repository on github. To add new files to the repository or add changed files to staged area:

- *git init*
- *git status*
- *git add < files >*
- *git commit*
- *git commit - a*
- *git push origin*
- *git push origin master*
- *git fetch origin*

If you want to clone any repository from github. Follow these commands:

- `cd nameofdirectory`
- `git init`
- `git clone forkedURL`

4.0.48 shell scripting

- What is Bash?

Bash is a Unix shell: a command-line interface for interacting with the operating system. It is widely available, being the default shell on many GNU/Linux distributions and on Mac OS X; and ports exist for many other systems. It was created in the late 1980s by a programmer named Brian Fox, working for the Free Software Foundation. It was intended as a free-software alternative to the Bourne shell (in fact, its name is an acronym for Bourne-again shell), and it incorporates all features of that shell, as well as new features such as integer arithmetic and in-process regular expressions.

- What is Shell?

The shell is the program which actually processes commands and returns output. Most shells also manage foreground and background processes, command history and command line editing. These features (and many more) are standard in bash, the most common shell in modern linux systems.

- What is shell scripting?

In addition to the interactive mode, where the user types one command at a time, with immediate execution and feedback, Bash (like many other shells) also has the ability to run an entire script of commands, known as a Bash shell script (or Bash script or shell script or just script). A script might contain just a very simple list of commands or even just a single command or it might contain functions, loops, conditional constructs, and all the other hallmarks of imperative programming. In fact, a Bash shell script is a computer program written in the Bash programming language. Shell scripting is the art of creating and maintaining such scripts. Shell scripts can be called from the interactive command-line described above; or, they can be called from other parts of the system.

One script might be set to run when the system boots up; another might be set to run every weekday at 2:30 AM; another might run whenever a user logs into the system. Shell scripts are commonly used for many system administration tasks, such as performing disk backups, evaluating system logs, and so on. They are also commonly used as installation scripts for complex programs.

They are particularly suited to all of these because they allow complexity without requiring it: if a script just needs to run two external programs, then it can be a two-line script, and if it needs all the power and decision-making ability of a Turing-complete imperative programming language, then it can have that as well.

4.0.49 Lua- Programming Language

Lua is a powerful, fast, lightweight, embeddable scripting language. Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

Lua is free software distributed in source code. It may be used for any purpose, including commercial purposes, at absolutely no cost. All versions are available for download. The current version is Lua 5.2 and its current release is Lua 5.2.3. Lua has many features:

- LUA is a proven, Robust language.
- LUA is fast.
- LUA is portable.
- LUA is embeddable.
- LUA is powerful, but simple.
- LUA is small.
- LUA is free.

4.0.50 GIMP



Figure 4.7: GIMP logo

GIMP is an acronym for GNU Image Manipulation Program. It is a freely distributed program for such tasks as photo retouching, image composition and image authoring. It has many capabilities. It can be used as a simple paint program, an expert quality photo retouching program, an online batch processing system, a mass production image renderer, an image format converter, etc. GIMP is expandable and extensible. It is designed to be augmented with plug-ins and extensions.

4.0.51 Features:

GIMP has tools used for image retouching and editing, free-form drawing, resizing, cropping, photo-montages, converting between different image formats, and more specialized tasks. Animated images such as GIF and MPEG files can be created using an animation plugin.

The developers and maintainers of GIMP strive to create a high-end free software graphics application for the editing and creation of original images, photos, icons, graphical elements of web pages, and art for user interface elements.

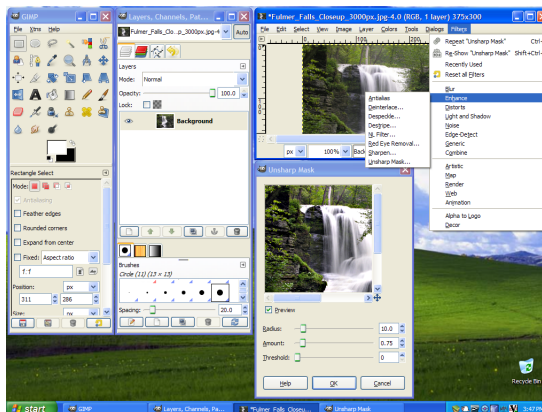


Figure 4.8: GIMP FEATURES

colour There are several ways of selecting colors including palettes, color choosers and using an eyedropper tool to select a color on the canvas. The built-in color choosers

include RGB/HSV selector or scales, water-color selector, CMYK selector and a color-wheel selector. Colors can also be selected using hexadecimal color codes as used in HTML color selection.

Selections and paths There are tools for creation of selections include a rectangular and circular selection tool, free select tool, and fuzzy select tool (also known as magic wand). More advanced selection tools include the select by color tool for selecting contiguous regions of color and the scissors select tool which creates selections semi-automatically between areas of highly contrasting colors.

Image editing There are many tools that can be used for editing images in GIMP. The more common tools include a paint brush, pencil, airbrush, eraser and ink tools used to create new or blended pixels. Tools such as the bucket fill and blend tools are used to change large regions of space in an image and can be used to help blend images. A list of GIMP transform tools include the align tool, move, crop, rotate, scale, shear, perspective and flip tools.

Layers, layer masks and channels An image being edited in GIMP can consist of many layers in a stack. The user manual suggests that "A good way to visualize a GIMP image is as a stack of transparencies," where in GIMP terminology, each transparency is a layer. This channel measures opacity where a whole or part of an image can be completely visible, partially visible or invisible.

File formats

1. Import and export: GIMP has import and export support for image formats such as BMP, JPEG, PNG, GIF and TIFF, along with the file formats of several other applications such as Autodesk flic animations, Corel Paint Shop Pro images, and Adobe Photoshop documents. Other formats with read/write support include PostScript documents, X bitmap image and Zsoft PCX.
2. Import only: GIMP can import Adobe PDF documents and the raw image formats used by many digital cameras, but cannot save to these formats. An open source plug-in, UFRaw, adds full raw compatibility, and has been noted for being quicker than Adobe in updating for new camera models, several times.

4.1 Coding standards of Language used

1. **Meaningful names of variables, classes and functions:**
The name of any variable, class or function should clearly imply what it does. Assigning a random name to any of these should not be done.
e.g. **myFunctionName()** is not a meaningful name.
2. **Spacing:**
Proper indentation and spacing is to be followed.
 - (a) No space should precede the comma.
e.g

```
func(a, b, c) // correct
func(a ,b , c) // incorrect
func(a , b,c) // incorrect
```

- (b) While creating conditional blocks, say using **if** or **for** etc., a space should be there between the keyword and the opening parenthesis that follows.

e.g

```
if (condition) // correct
if(condition)// incorrect
```

- (c) Creating a class should look like

```
className : public inheritedClass{
    //class definition
};
```

Keeping a space before and after the colon is advised and curly brace must start after the class name.

3. camel-Casing:

The naming standard for names to be followed is camel-Casing.

e.g

```
functionName
className
variableName
```

4. Defining if-else blocks:

A line between **if** and **else** blocks: An empty line must be there between if and else blocks if each contains multiple statements. Otherwise, this empty line can be avoided.

e.g

```
if (condition) {
    /**
     * do something
     * do more
     */
} else {
    /**
     * do something
     * do more
     */
}
```

Or it can be like:

```
if (condition)
    // do something
else
    // do something
```

4.2 Project Scheduling

Scheduling the project tasks is an important project planning activity. Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.

In project management, a schedule is a listing of a project's milestones, activities, and deliverables, usually with intended start and finish dates. Those items are often estimated in terms of resource allocation, budget and duration, linked by dependencies and scheduled events. A schedule is commonly used in project planning and project portfolio management parts of project management. Elements on a schedule may be closely related to the work breakdown structure.

Before a project schedule can be created, the schedule maker should have a work breakdown structure (WBS), an effort estimate for each task, and a resource list with availability for each resource.

In order to schedule the project, activities, a software project needs to the following:

1. Identify all the tasks needed to complete the project
2. Break down large tasks into small activities
3. Determine the dependency among different activities
4. Establish the time duration for completing the various activities
5. Plan starting and ending date for completing the various activities

Various tools that are used for Project scheduling are Gantt Charts or PERT Charts.

4.2.1 Gantt Charts

A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity. This allows you to see at a glance:

1. What the various activities are
2. When each activity begins and ends
3. How long each activity is scheduled to last
4. Where activities overlap with other activities, and by how much
5. The start and end date of the whole project

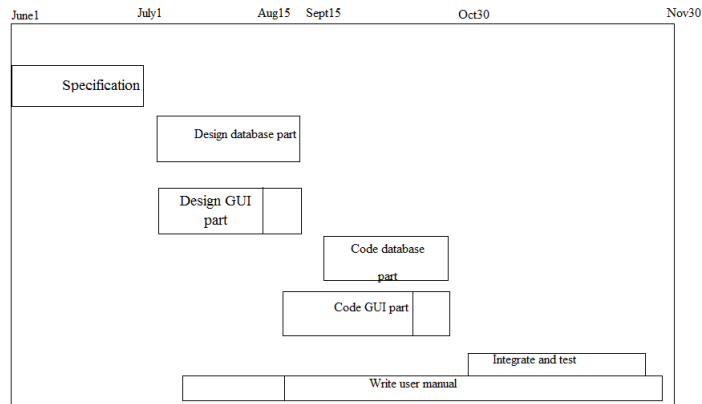


Figure 4.9: Gantt Chart

4.2.2 PERT Chart

The program (or project) evaluation and review technique, commonly abbreviated PERT, is a statistical tool, used in project management, which was designed to analyze and represent the tasks involved in completing a given project. PERT is a method to analyze the involved tasks in completing a given project, especially the time needed to complete each task, and to identify the minimum time needed to complete the total project.

PERT was developed primarily to simplify the planning and scheduling of large and complex projects. It was developed for the U.S. Navy Special Projects Office in 1957 to support the U.S. Navy's Polaris nuclear submarine project. It was able to incorporate uncertainty by making it possible to schedule a project while not knowing precisely the details and durations of all the activities. It is more of an event-oriented technique rather than start- and completion-oriented, and is used more in projects where time is the major factor rather than cost. It is applied to very large-scale, one-time, complex, non-routine infrastructure and Research and Development projects.

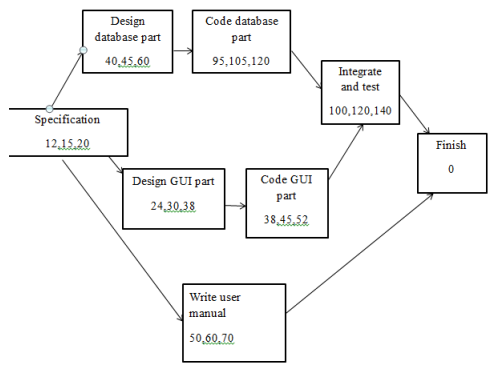


Figure 4.10: Pert Chart

4.3 Testing Techniques and Test Plans

System testing validates software once it has been incorporated into a larger system. Software is incorporated with other system elements and a series of system integration and validation tests are conducted. System testing is actually a series of different tests whose primary purpose is to fully exercise the computer based system. Once the system has been developed it has to be tested. In the present system we have to take care of valid property and assessment numbers i.e. there should not exist any duplicate number in each case. Care should be taken that the appropriate data is retrieved in response to the queries. The main thing, which should be considered, is the user interface. We have to see whether the user finds any difficulty in executing the system. Necessary messages and prompts should appear as needed to necessitate the users and make him alert if the invalid actions and help him to redo the required action. All the above cases any more are tested by providing the required data as input to the system so that any more tested by providing the required data as input to the system so that any uncovered errors can be traced out. For unit testing, we tested each module, as they are working well independently, went for integration testing. We integrated all the modules and after integration also they gave same results. For validation we checked for all different type of inputs and they gave correct results, and did the total process will satisfy the total validations that will be checked and find out the results.

1. The Data type Input Testing

- **String Test Case** : It is the test case in which there is an insertion of string records in a text box.
 - (a) A-Z alphabet acceptance testing: in a string test case, we can enter only alphabets (a-z) in a textbox.
 - (b) Number testing: cannot enter numerals in string test case.
 - (c) Blank Space testing: cannot enter blank spaces.
 - (d) Maximum length: cannot enter more than 50 characters.
- **Numerical Test Case** : There is an insertion of numbers ranging from 0-9 in the text box.
 - (a) 0-9 number acceptance testing: The numbers 0-9 can only enter in the text box.
 - (b) A-Z alphabet testing: cannot enter alphabets (a-z) in number test case.
 - (c) Blank Space testing: cannot put blank spaces in number text box.
 - (d) Length of integer testing: The numerals in a text box are restricted by a specific maximum length for a specific field. Maximum length for Phone (cell, home) = 10
- **Boolean Test case** : It is a testing of check boxes which are to be set either true or false.
- **String, Number and Symbol Test case** : It is a test case consists of entry of all the data types like string, number and symbols, etc.
- **Date input Test Case** : when the text box is to be entered with date, then this testing is used.

- (a) 5.1. Due (Before) date testing: When the date entered is to be the date less than the todays date.
 - (b) After Date testing: When the date entered is to be the date after the todays date.
 - (c) Current date Testing: When the date entered is to be the current date only.
2. **CRUD Operations Testing** : Its testing of create, read, update, delete operations.
- (a) **Save** : It is to test the successful saving operation of record in any part of the software.
 - (b) **Update** : It is to test the update/editing operation of records in any part of software.
 - (c) **Delete** : This test is to check the delete operation in the software.
 - (d) **Retrieval** : This test case is to check the successful retrieval of data and records on clicking of any link.

Chapter 5

Results and Discussions

5.0.1 User Interface Representation

As this software is to be used mostly by civil and mechanical engineers or by developers, so to make its look and feel professional, the console based user interface is designed for this software. Also shell scripting has been used for the execution of the software, which further increases its speed and is easier to use and maintain both by users and developers.

5.0.2 Brief Description of Various Modules of the system

Brief Description of Various Modules: The various modules which have been used are as follows:

Shell Scripting -

All the modules are combined using shell scripting. The main execution is also done through scripts.

std-flt -

It is a one way converter which converts the file format of STAAD-PRO software i.e std to the file format of FELT software i.e flt. It uses the Flex for lexical analysis which is the first phase of Converter. It reads the input file, i.e, .std file and break down into tokens through lexical analysis. Then the tokens are parsed using Bison parser generator tool and the parsed information is used for writing the output file, i.e .flt file, which is the last phase of converter. And finally the .std file is converted to .flt file.

flt-std -

This module completes the two way converter as it converts the file format of FELT i.e flt to the file format of STAAD-PRO, i.e std. It uses the same methodology for its conversion, i.e Lexical analysis of the input file and dividing into tokens, Parsing tokens returned after reading it and then finally Writing the output file. The Lexical analysis is done using Flex, Parsing by Bison and File is written using C++ Programming language.

5.0.3 Snapshots of system with brief detail of each

In Six Months Training, I have done Project based on Flex and Bison technology. Using this, I have done four file converter i.e. DXF Converter, GDGNE Converter, XCAD-ECAD Converter and the final project is Felt-Staad-Pro Converter.

5.0.4 DXF Converter

dxfl - Qmake is used to compile the program. Qmake is the very first command which is used to compile this program. The make file is created by using qmake. dxfl -

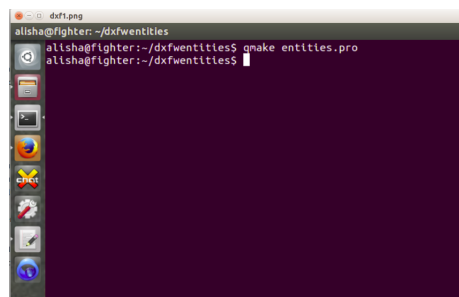


Figure 5.1: DXF

The next command is make. This make command compiles the program using g++ compiler, which is a GNU compiler and used to compile programs written in C++. After this command an executable file is created, ./entities which takes input a text file and creates a dxf file based on the instructions given in the input text file.

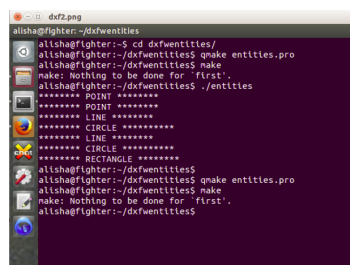


Figure 5.2: DXF in C++

dxfl3 - The last step is to run the executable file which creates the output file i.e the dxf file. Dxf file when opened using any CAD software produces a drawing based on the instructions given in the text file and the output which is displayed on terminal which acknowledges the user, the entities which are written in dxf file.

dxfl4 - This is the sample input file which was given to run the program.

dxfl5 - This is the output file which produces a drawing when opened in LibreCAD, a CAD software.

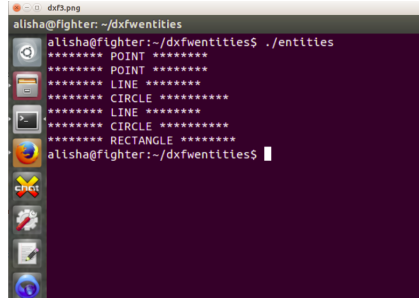


Figure 5.3: DXF Co-ordinates

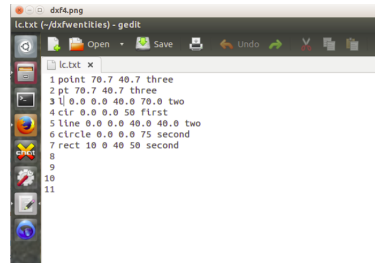


Figure 5.4: Input File

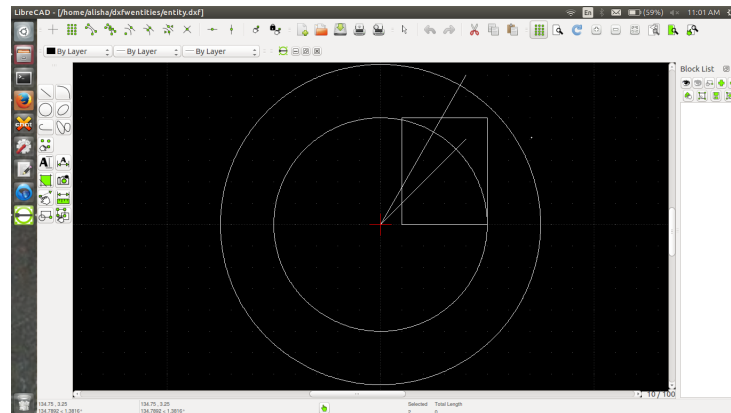


Figure 5.5: Write DXF

5.0.5 GD-GNE Converter

GNE-GD converter is two a way converter. In which we converts on file "conv.gne" to "conv.gd". snapshots of system with brief detail of each:

1. In below figure, the list of files which we write are given. "conv.gne" and "conv.gd" are the input files. "conv.l" is lexer file and conv.y is Bison file. Parser.cc and Parser.h files are include C++ code.
2. Now, two library files are created by Flex and Bison. "lex.yy.c" file is generated by Flex. "conv.tab.c" and "conv.tab.h" files are generated by Bison.
3. "a.sh" is script which runs the code. This file includes Flex and Bison commands.

```
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$ ls
a.sh conv.gd conv.l lex.yy.c parser.h
conv conv.gne conv.y parser.cc README.md
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$
```

Figure 5.6: List Of Files

```
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$ ls
a.sh conv.gd conv.l lex.yy.c parser.h
conv conv.gne conv.y parser.cc README.md
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$ flex conv.l
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$ bison -d conv.y
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$ ls
a.sh conv.gd conv.l conv.tab.h lex.yy.c parser.h
conv conv.gne conv.tab.c conv.y parser.cc README.md
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$
```

Figure 5.7: Generate Library Files

```
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$ bash a.sh
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$ ./conv
*****
Enter the file format you want to convert
1 - Convert GNE format to GD
2 - Convert GD format to GNE

```

Figure 5.8: Script run

4. In screenshot the executable script shows.

```
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$ bash a.sh
jot_kaur@jotkaur: ~/Documents/bison-flex/GD.GNEconverter$ ./conv
*****
Enter the file format you want to convert
1 - Convert GNE format to GD
2 - Convert GD format to GNE
1
Enter the name of file you want to output
data
Enter the name of file you want to input

```

Figure 5.9: Execute script

5. As shown in image, the output file and input files are generated.

```
jot_kaur@jotkaur:~/Documents/bison-flex/GD.GNEconverter$ bash a.sh
jot_kaur@jotkaur:~/Documents/bison-flex/GD.GNEconverter$ ./conv
*****
Enter the file format you want to convert
1 - Convert GNE format to GD
2 - Convert GD format to GNE
1
Enter the name of file you want to output
data
Enter the name of file you want to input
conv
*****
jot_kaur@jotkaur:~/Documents/bison-flex/GD.GNEconverter$
```

Figure 5.10: Output File

6. Two way converter is used in GD-GNE parser which shows in image.

```
jot_kaur@jotkaur:~/Documents/bison-flex/GD.GNEconverter
*****
Enter the file format you want to convert
1 - Convert GNE format to GD
2 - Convert GD format to GNE
1
Enter the name of file you want to output
data
Enter the name of file you want to input
conv
*****
jot_kaur@jotkaur:~/Documents/bison-flex/GD.GNEconverter$ vim data.gd
jot_kaur@jotkaur:~/Documents/bison-flex/GD.GNEconverter$ ./conv
*****
Enter the file format you want to convert
1 - Convert GNE format to GD
2 - Convert GD format to GNE
2
Enter the name of file you want to output
data
Enter the name of file you want to input
conv
*****
jot_kaur@jotkaur:~/Documents/bison-flex/GD.GNEconverter$ vim data.gne
```

Figure 5.11: Two way converter

7. As shown in image, data.gne and data.gd files are generated by GD-GNE Parser.

```
jot_kaur@jotkaur:~/Documents/bison-flex/GD.GNEconverter
I data.gne Row 1 Col 1 4:08 Ctrl-K H for help
BINDU (10.4, 7.8) pnt1
REKHA (4.6, 6.7) p1
(5.5, 2) p2
ln1

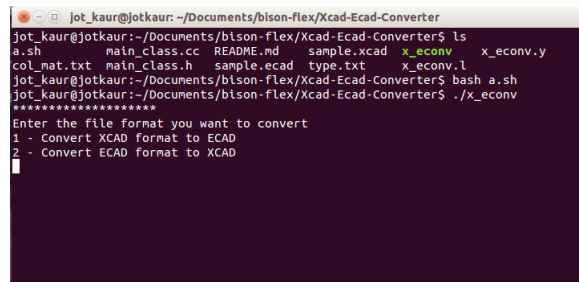
I data.gd Row 1 Col 1 4:08 Ctrl-K H for help
POINT
pnt1
x = 8.8
y = 8.5

LINE
ln1
p1
p2
p1
** Joe's Own Editor v3.7 ** (utf-8) ** Copyright © 2008 **
```

Figure 5.12: Output files generated by parser

5.0.6 XCAD-ECAD Converter

8. It asks the user about his requirements regarding the conversion of file formats i.e which file format will be his input file and which one will be his output file. On the basis of his input, the further working of the software is proceeded.

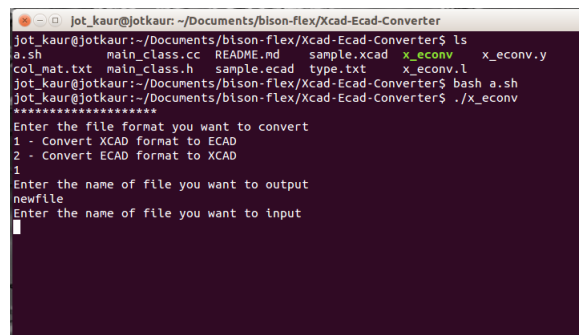


```
jot_kaur@jotkaur: ~/Documents/bison-flex/Xcad-Ecad-Converter
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$ ls
a.sh          main_class.cc  README.md     sample.xcad  x_econv      x_econv.y
col_mat.txt   main_class.h   sample.ecad   type.txt     x_econv.l
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$ bash a.sh
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$ ./x_econv
*****
Enter the file format you want to convert
1 - Convert XCAD format to ECAD
2 - Convert ECAD format to XCAD

```

Figure 5.13: List of files that we write

9. If the user chooses first option, this window is printed, which guides the user about how he is going to direct the system about his input file in the std format and the output file in the ftt format.

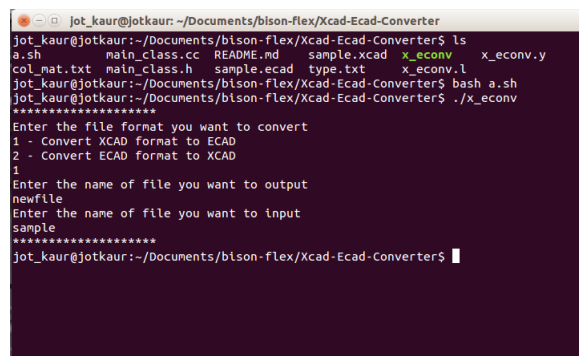


```
jot_kaur@jotkaur: ~/Documents/bison-flex/Xcad-Ecad-Converter
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$ ls
a.sh          main_class.cc  README.md     sample.xcad  x_econv      x_econv.y
col_mat.txt   main_class.h   sample.ecad   type.txt     x_econv.l
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$ bash a.sh
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$ ./x_econv
*****
Enter the file format you want to convert
1 - Convert XCAD format to ECAD
2 - Convert ECAD format to XCAD
1
Enter the name of file you want to output
newfile
Enter the name of file you want to input

```

Figure 5.14: Output files generated by parser

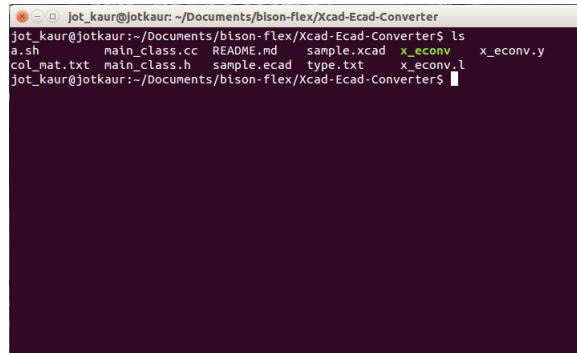
10. This acknowledges the user about the creation of the new output file i.e ftt file in the same directory where he is running his software.



```
jot_kaur@jotkaur: ~/Documents/bison-flex/Xcad-Ecad-Converter
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$ ls
a.sh          main_class.cc  README.md     sample.xcad  x_econv      x_econv.y
col_mat.txt   main_class.h   sample.ecad   type.txt     x_econv.l
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$ bash a.sh
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$ ./x_econv
*****
Enter the file format you want to convert
1 - Convert XCAD format to ECAD
2 - Convert ECAD format to XCAD
1
Enter the name of file you want to output
newfile
Enter the name of file you want to input
sample
*****
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$
```

Figure 5.15: Output file

11. It echos a message which acknowledges the user about the creation of the new std file with the name as prescribed by the user, in the same directory in which he is executing the software.



```
jot_kaur@jotkaur: ~/Documents/bison-flex/Xcad-Ecad-Converter
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$ ls
a.sh      main_class.cc  README.md      sample.xcad    x_econv       x_econv.y
col_mat.txt main_class.h    sample.ecad     type.txt       x_econv.l
jot_kaur@jotkaur:~/Documents/bison-flex/Xcad-Ecad-Converter$
```

Figure 5.16: Execute File

5.0.7 Felt-Staad-Pro Converter

snapshots of system with brief detail of each:

1. snapshot 1- It asks the user about his requirements regarding the conversion of file formats i.e which file format will be his input file and which one will be his output file. On the basis of his input, the further working of the software is proceeded.



```
jot_kaur@jotkaur: ~/Documents/Training/std-flt-converter
jot_kaur@jotkaur:~/Documents/Training$ cd std-flt-converter/
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$ ./compile.sh
1. Stad-flt
2. Flt-Stad
Please choose a word [1,2]?
```

Figure 5.17: File Format

2. snapshot 2- If the user chooses first option, this window is printed, which guides the user about how he is going to direct the system about his input file in the std format and the output file in the flt format.
3. snapshot 3- This acknowledges the user about the creation of the new output file i.e flt file in the same directory where he is running his software.
4. snapshot 4- If the user chooses second option, this message is printed, which lets the user use with the software in a more easier way. A message is displayed which

```
jot_kaur@jotkaur: ~/Documents/Training/std-flt-converter
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$ ./compile.sh
1. Stad-flt
2. Flt-Stad
Please choose a word [1,2]? 1
*****
*                               *
*   Run the following command to convert the file   *
*   ./convert-std-flt input_file_name.std output_file_name.flt   *
*                               *
*****
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$
```

Figure 5.18: Output file

```
jot_kaur@jotkaur: ~/Documents/Training/std-flt-converter
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$ ./compile.sh
1. Stad-flt
2. Flt-Stad
Please choose a word [1,2]? 1
*****
*                               *
*   Run the following command to convert the file   *
*   ./convert-std-flt input_file_name.std output_file_name.flt   *
*                               *
*****
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$ ./convert-std-flt beam.
std newfile.flt
*****
*   newfile.flt has been generated in the same directory   *
*                               *
*****
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$
```

Figure 5.19: Terminal output

guides the user about the commands he should run to convert his flt file to std file.

```
jot_kaur@jotkaur: ~/Documents/Training/std-flt-converter
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$ ./compile.sh
1. Stad-flt
2. Flt-Stad
Please choose a word [1,2]? 2
*****
*                               *
*   Run the following command to convert the file   *
*   ./convert-flt-std input_file_name.flt output_file_name.std   *
*                               *
*****
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$
```

Figure 5.20: Snapshot option file

5. snapshot 5- It echos a message which acknowledges the user about the creation of the new std file with the name as prescribed by the user, in the same directory in which he is executing the software.

```
jot_kaur@jotkaur: ~/Documents/Training/std-flt-converter
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$ ./compile.sh
1. Stad-flt
2. Flt-Stad
Please choose a word [1,2]? 2

*****
*          Run the following command to convert the file          *
*          ./convert-flt-std input_file_name.flt output_file_name.std *
*****
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$ ./convert-flt-std bean.
flt file.std
*****
*          file.std has been generated in the same directory          *
*****
jot_kaur@jotkaur:~/Documents/Training/std-flt-converter$
```

Figure 5.21: Snapshot output file

Chapter 6

Conclusion and Future Scope

6.0.8 Conclusion:

These are two way converters which converts two file formats into each other according to the user's requirement. The file format conversion is strictly done on the basis of file format specifications. Also the conversion goes through various phases among which reading and parsing the input file, and writing the output file are major. Also the converter is platform independent as it converts the files of softwares which are for different platforms that is STAAD-PRO, which is a Windows based Fem software and FELT, which is an open source software.

6.0.9 Future Scope:

A general purpose converter can be developed by integrating all these converters and including many other file formats which can be embedded in any CAD or FEM software or can be used independently.

Bibliography

- [1] *whatis.techtarget.com/fileformat/DXF – Drawing – Interchange – Format – AutoCAD*
- [2] *ribbonsoft.com/doc/dxf/lib/2.5/reference/dxf/lib-reference-manual.pdf*
- [3] *flex.sourceforge.net/*
- [4] *aquamentus.com/flexison.html*
- [5] *dinosaur.compilertools.net/*
- [6] *github.com/jotpandher/Feelt-staad-converter*
- [7] *harjotpandher93.wordpress.com/*