Compiler Construction Tools

Mrs. Bhumika S. Zalavadia[1]
[1]HOD Diploma Computer Department
[1]Atmiya Institute of Technology and Science for Diploma Studies-Rajkot, India

Abstract--- In this basic idea of compiler and its functionality is discussed. Compiler is the most important part of computer system. When user tries to run any program in any programming language, compiler comes into picture at first. The key role of compiler is to scan the entire program and convert it not machine language. Compiler construction is very vast branch of computer science that deals with the theoretical and practical aspects of designing the compiler using different programming language.

I. INTRODUCTION

Compiler is software that scans the entire program first and then translates it into machine code. Before converting the entire program into machine code, it checks the entire program for syntax error if any and reports the list of syntax errors. When all syntax errors are removed it converts the source code into machine code which is also known as object code.

There are two aspects of compilation:

(1) Generate code to implement meaning of a source program.
(2) Provide diagnostics for violation of program language Symantec in a source program.

II. COMPILATION PROCESS

Compilation process consists of following steps:

A. Lexical Analysis

The lexical analyzer reads the stream of characters which makes the source program and groups them into meaningful sequences called lexemes.

B. Syntax Analysis

The list of tokens produced by the lexical analysis phase forms the input and arranges them in the form of tree-structure called the syntax tree. This reflects the structure of the program. This phase is also called parsing.

C. Semantic analysis

This phase uses the syntax tree and the information in the symbol table to check the source program for consistency with the language definition.

D. Intermediate Code Generation

This is the process of translating a source program into target code. Compiler may generate one or more intermediate codes.

E. Code Optimization

This is a machine-independent phase which attempts to improve the intermediate code for generating better target code.

F. Code Generator

This takes the intermediate representation of the source program as input and maps it to the target language.

III. COMPILER CONSTRUCTION TOOLS

Compiler construction tools can use modern software development environments containing tools such as language editors, debuggers, version managers, profilers, test harnesses, and so on. Along with these general software development tools, other more specialized tools have been created to help implement various phases of a compiler.

These tools use specialized programming languages for implementing specific components using many sophisticated and complex algorithms. Most of the tools provide high degree of abstraction means the details of implementation is hidden from outside world.

Some commonly used compiler-construction (CC) tools include:

1. Parser generators that automatically produce syntax analyzers from a grammatical description of a programming language.
2. Scanner generators that produce lexical analyzers from a regular- expression description of the tokens of a language.
3. Syntax-directed translation engines that produce collections of routines for walking a parse tree and generating intermediate code.
4. Code-generator generators that produce a code generator from a collection of rules for translating each operation of the intermediate language into the machine language for a target machine.
5. Data-flow analysis engines that facilitate the gathering of information about how values are transmitted from one part of a program to each other part. Data-flow analysis is a key part of code optimization.
6. Compiler-construction toolkits that provide an integrated set of routines for constructing various phases of a compiler.

IV. SOFTWARE TOOLS IN COMPILER CONSTRUCTION

Computing involves two main activities: program development and use of application software. Language processors and operating system pay an obvious role in these activities. A less obvious but vital role is played by program that help in developing and using other programs. These programs, called software tools, perform various housekeeping tasks involved in program development and application usage.

Software tools is a system program which:

- Interface a program with the entity generating its input data
- Interfaces the results of a program with the entity consuming them.

Generally there are two kinds of software tools:

1) Software tools for program development
2) Software tools for user interfaces.

The fundamental steps in program development are:

1) Program design and coding
2) Program documentation.
3) Program translation, linking and loading.
4) Program testing and debugging
5) Performance tuning
6) Reformatting the data and / or results of a program to suit other programs.

1) Program design and coding:

Two categories of tools used in program design and coding are:

- Program generators
- Programming environment.

A program generator generates a program which performs a set of functions. User of a program generator saves substantial design effort since a programmer merely specifies what functions a program should perform rather than how the function should be implemented. A programming environment supports program coding by incorporating awareness of the programming language syntax and semantics in the language editor.

1) Program Documentation

Most programming projects suffer from lack of up-to-data documentation. Automatic documentation tools are motivated by the desire to overcome this efficiency. These tools work on the source program to produce different forms of documentation, e.g. flow charts, IO specifications showing files and their records etc.

2) Program Translation, Linking and Loading

These steps require use of language processors.

3) Program Testing and Debugging

Important steps in program testing and debugging are selection of test data for the program, analysis of test results to detect errors and debugging. Software tools to assist these steps come in the following forms:

- Test data generators help the user in selecting test data for his program.
- Automated test drives help in regression testing; Regression testing is performed as follows: Many steps of test data are prepared for a program. These are given as input to the test drivers. The driver selects one set of test data at a time and organizes execution of the program on the data.
- Debug monitors help in obtaining information for localization of errors.
- Source code control systems help to keep track of modifications in the source code.

4) Performance Tuning

Program efficiency depends on two factors- the efficiency of the algorithm and the efficiency of its coding. An optimizing compiler can improve efficiency of the code but it cannot improve efficiency of the algorithm. Only a program designer can improve efficiency designer can improve efficiency of an algorithm. A performance tuning tool helps in identifying such parts, those sections of a program which consume a considerable amount of execution time. A profile monitor is a software tool that collects information regarding the execution behavior of a program.

5) Program Preprocessing

Program preprocessing techniques are used to support static analysis of programs. Tools generating cross reference listings and lists of unreferenced symbols: test data generators and documentation aids use this technique.

V. SOFTWARE TOOL FOR PROGRAM DEVELOPMENT

A. EDITORS

These tools are mainly used to program entry and editing, as a front ends Tool. The editor functions in two modes- command mode and data mode. In command mode, it accepts user commands specifying the editing function to be performed. In the data mode, the user keys in the text to be added to the file. Text Editors come in the following forms:

1) Line editors
2) Stream editors
3) Screen editors
4) Word processors
5) Structured editors

1) Line Editors:

The scope of edit operations in a line editor is limited to a line of text. The line is designated positionally. e.g. by specifying its serial number in the text, or contextually e.g. by specifying a context which uniquely identifies it. The primary advantage of line editors is their simplicity.

2) Stream editors:

A stream editor views the entire text as a stream of characters. These permits edit operations to cross line boundaries. Stream editors typically support character, line and context oriented commands based on the current editing context indicated by the positions of a text pointers.

3) Screen Editors:

A line or stream editor does not display the text in the manner it would appear if printed. A screen editor uses the what you-see-is-what-you-get principle in editor design. The screen editor displays a screen full of text at a time. The user can move the cursor over the screen, position it at the point where he desires to perform some editing and proceed with the editing directly. Thus it is possible to see the effect of an edit operation on the screen. This i is very useful while formatting the text to produce printed documents.

4) Word Processors:

Word processors are basically documents editor with additional features to produce will formatted hard copy output. Essential features of word processors are

commands for moving sections of text from one place to another, merging of text, and searching and replacements of words. Many word processors support a spell-check option. WordStar is a popular editor of this class.

5) Structure Editors: A structure editors incorporates an awareness of the structure of a document. This is useful in browsing through a document e.g. if a programmer wishes to edit a specific function in a program file. A special class of structure editors, called syntax directed editors, is used in programming environments.

VI. DEBUG MONITORS

A debug monitor is software which provides debugging support for a program. The debug monitor executes the program being debugged under its own control. This provides execution efficiency during debugging. A debug monitor can be made language independent, in which case it can handle programs written in many languages.

Debug monitors provide the following facilities for dynamic debugging:

1) Setting breakpoints in the program
2) Initiating a debug conversation when control reaches a breakpoint.
3) Displaying values of variables
4) Assuming new values to variables
5) Testing user defined assertions and predicates involving program variables.

The sequence of steps involved in dynamic debugging of a program is as follows:

1) The user compiles the program under the debug option. The compiler produces two files the command code file and the debug information file.
2) The user activities the debug monitors and indicates the name of a program to be debugged. The debug monitor opens the compiled code and debugs information files for the program.
3) The user specifies his debug requirements a list of breakpoints and actions to be performed at breakpoints. The debug monitor instruments the program and builds a debug table containing the pairs (statement number, debug action).
4) The instrumented program gets control and executes up to a breakpoint.
5) The software interrupt is generated when the <SI instn> is executed. Control is given to the debug monitor who consults the debug table and performs the debug actions specified for the breakpoint. A debug concession is now opened during which the user may issue some debug commands or modify breakpoints and debug actions associated with breakpoints. Control now returns to the instrumented program.
6) Step – above two steps are repeated until the end of the debug session.

VII. PROGRAMMING ENVIRONMENTS

A programming environment is a software system that provides integrated facilities for program creation, editing, execution, testing and debugging. It consists of the following components:

1) A syntax directed editor
2) A language processor
3) A debug monitor
4) A dialog monitor

All components are accessed through the dialog monitor.

The syntax directed editor incorporates a front for the programming language as a user keys in his program. The editor performs syntax analysis and converts it into an intermediate representation (IR), typically an abstract syntax tree. The compiler (or interpreter) and the debug monitor share the IR. If a compiler is used, it is activated after the editor has converted a statement to IR. The compiler works incrementally to generate code for the statement. Thus, program execution or interpretation can be supported immediately after the last statement has been input.

The dialog monitor may also provide other program development and testing functions. For example, it may permit a programmer to execute a partially completed program. The programmer can be alerted if an undeclared variable or an incomplete statement is encountered during execution. The programmer can insert necessary declarations or statements and resume execution. This permits major interfaces in the program to be tested prior to the development of a module. Some programming environments also support reversible execution.

VIII. MACRO AND MACRO EXPANSION

Macros are used to provide a program generation facility through macro expansion. Many languages provide built-in-facilities for writing macros well known examples of these are the higher level language PL/L, C, Ada and C++.

A macro is a unit of specification for program generation through expansion. A macro consists of a name, a set of formal parameters and a body of code. The use of a macro name with a set of actual parameters is replaced by some code generated from its body. This is called macro expansion. Macros differ from subroutines in one fundamental respect. Use of a macro name in the mnemonic field of an assembly statements leads to its expansion.

IX. USER INTERFACES

A user interface (UI) plays a vital role in simplifying the interaction of a user with an application. Classically, UI functionalities have two important aspects issuing of commands and exchange of data. Basically user interface is a combination of menus; screen design, keyboard commands and language which together create the way a user interact with the system. User interface establishes the dialog between the users and the computers the dialog provides user friendly instructive approach to starting the system, setting options, getting helps etc. use of menus, hyperlinks, dialog boxes and drop down list provides a very pleasant interface but design a programming are complicated.

A UI can be visualized to consist of two components a dialog manger and presentation manager.

- The dialog manager manages the conversation between the user and the application. This involves prompting the use for a command and transmitting the command to the application.
- The presentation manager displays the data

produced by the application in an appropriate manner on the user's displays or printer device.

The following points should be kept in mind while designing a user interface:

- Know the requirements of the users.
- Involve these users as much as possible in the screen and dialog design.
- User interface must be helpful.
- The interface must be robust.
- It feasible graphical ion must be used.
- Test the user interface on actual users.

X. STRUCTURE OF USER INTERFACE

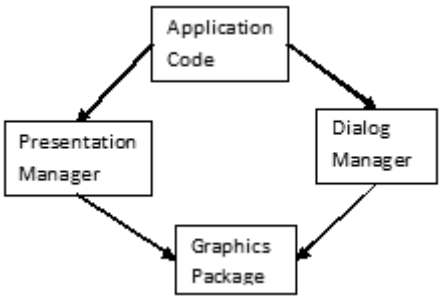

Fig. 1: Structure of User Interface

Figure shows a UI schematic using a standard graphics package. The UI consists of two main components, presentation manager and dialog manager.

- The presentation manager is responsible for managing the user screen and for accepting data and presenting results.
- The dialog manager is responsible for interpretation user commands and implementing them by invoking different modules of the application code. The dialog manager is also responsible for error messages and on line help functions.

XI. DESIGNING USER INTERFACE

Designing the user interface involves the following steps:

1) Sketch the user interface dialogs.
2) Prototype the user interface.
3) Obtain user feedback.
4) If negative feedback is received, repeat steps 1 to 3 again.

User Interface design is an area of work in which user should play a significant role. However he needs the professional assistance of the system analyst. Following are the three common method for data entry input:

1) Menu Method: A menu method is used to select one out of many alternatives. Selecting a menu will result in the appearance of related sub-menus.
2) Template Method: Templates are like forms on computer screen. The user is requested to fill in the form. Labeled fields are provided and user enters data into the blank spaces.
3) Command Method: - In the command method the computer asks the user for specific inputs. On getting the input the computer may either ask for further input or provide some output.

ACKNOWLEDGEMEN

I am heartily thankful to my Principal and my institute to encourage me to do this research in this area and giving me the opportunity to share this knowledge with others.

REFERENCES

[1] www.cse.iitd.ernet.in
[2] Principles_of_Compiler_Design.htm
[3] www.cs.usfca.edu/~galles/compilerdesign
[4] www.eis.mdx.ac.uk/staffpages/r_bornat/books/compiling.pdf
[5] Compiler design by Aho Ullman

