BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC ĐÀ NẴNG

TRẦN VĂN KHÁNH

NGHIÊN CỨU ỨNG DỤNG LEX/YACC ĐỂ HỖ TRỢ PHÁT SINH MÃ NGUỒN TRONG LẬP TRÌNH ỨNG DỤNG

Chuyên ngành : **KHOA HỌC MÁY TÍNH**
Mã số : 60.48.01

TÓM TẮT LUẬN VĂN THẠC SĨ KỸ THUẬT

Đà Nẵng – Năm 2011

Công trình được hoàn thành tại
ĐẠI HỌC ĐÀ NẴNG

Người hướng dẫn khoa học: **PGS.TS. VÕ TRUNG HÙNG**

Phản biện 1: **PGS.TSKH. TRẦN QUỐC CHIẾN**

Phản biện 2:**TS.TRƯƠNG CÔNG TUẤN**

Luận văn sẽ được bảo vệ trước Hội đồng chấm Luận văn tốt nghiệp Thạc sĩ Kỹ thuật họp tại Đại học Đà Nẵng vào ngày 18 tháng 6 năm 2011

Có thể tìm hiểu luận văn tại:
- Trung tâm Thông tin - Học liệu, Đại học Đà Nẵng
- Trung tâm Học liệu, Đại học Đà Nẵng

MỞ ĐẦU

1. Tính cấp thiết

Trong công nghệ thông tin, cùng với sự phát triển các thiết bị phần cứng là sự hình thành và phát triển của các kỹ thuật lập trình. Câu hỏi "làm thế nào để lập trình nhanh và hiệu quả nhất cho một bài toán cụ thể nào đó?" luôn là cơ sở để các phương pháp, kỹ thuật và ngôn ngữ lập trình ra đời. Một trong những kỹ thuật ra đời sớm, tồn tại và phát triển cho đến tận bây giờ đó là kỹ thuật tự động sinh mã nguồn.

Tự động sinh mã nguồn là một ý tưởng táo bạo và mang lại hiệu quả cao trong lập trình. Thay vì lập trình viên phải trực tiếp viết mã để giải quyết một bài toán cụ thể nào đó thì họ chỉ cần đặc tả theo một hệ qui ước nhất định (các ngôn ngữ đặc tả) và trên cơ sở đó, máy tính sẽ tự động sản sinh ra mã nguồn tương ứng. Một trong những ngôn ngữ, công cụ như vậy là Lex/Yacc.

Lex/Yacc có chức năng phát sinh mã nguồn trong lĩnh vực tạo các chương trình dịch và sau này được mở rộng sang nhiều lĩnh vực khác. Để cài đặt một trình biên dịch, chúng ta chỉ cần cung cấp các đặt tả ngữ pháp, Lex/Yacc sẽ tự động sinh ra mã nguồn C hoặc Pascal. Việc này cho phép tiết kiệm rất nhiều thời gian cũng như hạn chế lỗi (bugs) trong mã nguồn.

Hiện nay, các trường phổ thông và các trường đại học trong nước hầu hết đã đưa ngôn ngữ lập trình bậc cao như C/C++, Pascal,... vào giảng dạy. Vấn đề đặt ra là làm thế nào để giảm chi phí về nghiên cứu, tiếp cận và tăng hiệu quả sử dụng các ngôn ngữ lập trình bậc cao. Với công cụ này, học sinh, sinh viên các trường có điều kiện tiếp cận và sử dụng có hiệu quả các công cụ lập trình của mình.

Hiện nay chưa có một nghiên cứu mang tính hệ thống về công cụ Lex/Yacc ở nước ta (đã có một số trường giới thiệu về Lex/Yacc trong môn học Chương trình dịch nhưng chưa đầy đủ). Vì vậy, việc nghiên cứu ứng dụng công cụ Lex/Yacc để nâng cao hiệu quả trong lập trình ứng dụng là vấn đề cấp thiết.

2. Mục đích của đề tài

Nghiên cứu ứng dụng Lex/Yacc để tăng hiệu quả khi lập trình ứng dụng. Sau khi nghiên cứu, sẽ trình bày lại một cách hệ thống về cách cài đặt, qui trình sử dụng và minh họa qua một số bài toán cụ thể.

3. Ý nghĩa của đề tài

Phân tích, đánh giá hiệu quả trong lập trình ứng dụng bằng cách sử dụng công cụ Lex/Yacc. Cung cấp tài liệu về công cụ Lex/Yacc một cách hệ thống và đầy đủ nhất có thể để phục vụ cho việc phát triển ứng dụng sau này. Góp phần thúc đẩy việc ứng dụng công cụ Lex/Yacc trong lập trình ứng dụng tại Việt Nam. Giúp cho học sinh, sinh viên có thể tham khảo dễ dàng về Lex/Yacc trong học tập cũng như lập trình sau này.

4. Phương pháp nghiên cứu

Khi thực hiện đề tài, chúng tôi đã kết hợp giữa phương pháp nghiên cứu lý thuyết và phương pháp nghiên cứu thực nghiệm. Về mặt lý thuyết, chúng tôi tiến hành nghiên cứu tổng quan về chương trình dịch, các bộ phân tích từ vựng và cú pháp, các bộ phân tích và phát sinh mã nguồn Lex/Yacc. Về mặt thực nghiệm, chúng tôi tiến hành một số công cụ hỗ trợ phân tích từ vựng và cú pháp và áp dụng chúng trên một số bài toán tiêu biểu với Lex/Yacc.

5. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu chính của đề tài là chương trình dịch, các ngôn ngữ đặc tả, các công cụ phát sinh tự động mã nguồn. Tuy nhiên, chúng tôi giới hạn phạm vi nghiên cứu của mình trên Lex/Yacc và phát sinh mã nguồn trong ngôn ngữ lập trình C/C++.

6. Cấu trúc luận văn

Báo cáo của luận văn tốt nghiệp này được tổ chức thành 3 chương. Trong chương 1, chúng tôi trình bày các kết quả nghiên cứu tổng quan về chương trình dịch, ... Chương 2 chúng tôi trình bày một cách hệ thống về Lex/Yacc. Trong chương cuối, chúng tôi nêu một số ví dụ qua Lex/Yacc và trang web giới thiệu về Lex/Yacc.

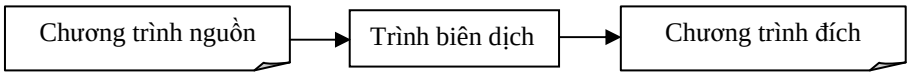
Chương 1. NGHIÊN CỨU TỔNG QUAN

Chương này trình bày các vấn về liên quan đến khái niệm chương trình dịch, cấu trúc của một trình biên dịch và bộ phân tích từ vựng và phân tích cú pháp.

1.1. Chương trình dịch

1.1.1. Khái niệm

Chương trình dịch, hay còn gọi là trình biên dịch, đơn giản là một chương trình làm nhiệm vụ đọc một chương trình nguồn được viết bằng một ngôn ngữ (gọi là ngôn ngữ nguồn và thường là các ngôn ngữ lập trình bậc cao) rồi dịch nó thành một chương trình đích tương đương ở một ngôn ngữ khác (gọi là ngôn ngữ đích và đa số các trường hợp thì nó là ngôn ngữ máy). Một phần quan trọng trong quá trình dịch là ghi nhận lại các lỗi có trong chương trình nguồn để thông báo lại cho người viết chương trình.

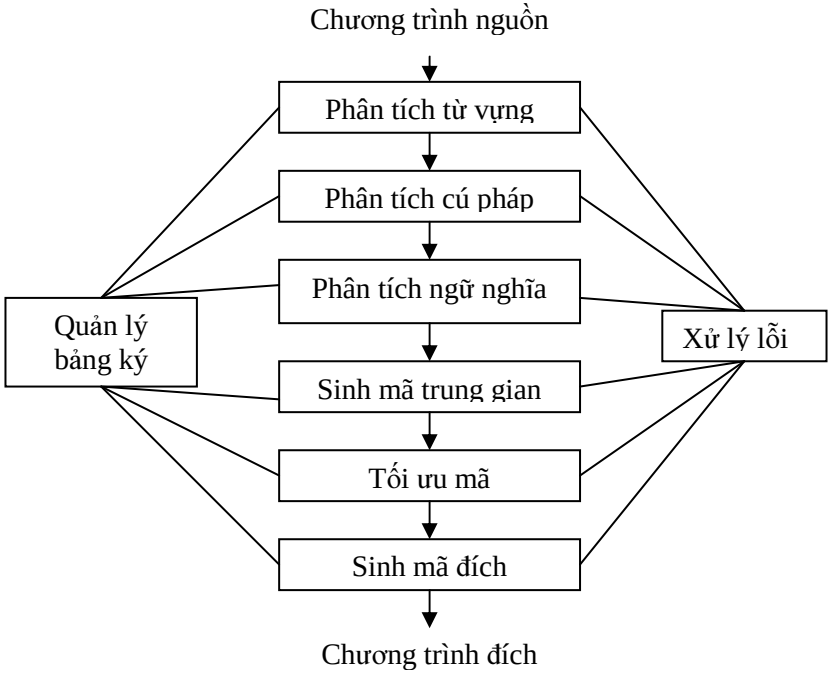


Hình 1.1 Mô hình của một trình biên dịch

Ví dụ, chúng ta có một chương trình nguồn viết bằng ngôn ngữ lập trình bậc cao (Pascal, C...). Để có thể thực hiện được chương trình này trên máy tính thì phải sử dụng một trình biên dịch (Compiler) để chuyển nó sang chương trình đích là ngôn ngữ máy (dạng mã nhị phân).

1.1.2. Qui trình dịch

Qui trình thông thường của một trình biên dịch được trình bày như sau:



Hình 1.2 Các giai đoạn của một trình biên dịch

Phân tích từ vựng: đọc từng ký tự gộp lại thành token. Token có thể là một danh biểu, từ khóa, một ký hiệu,... Chuỗi ký tự tạo thành một token gọi là trị từ vựng của token đó.

Phân tích cú pháp và phân tích ngữ nghĩa: xây dựng cấu trúc phân cấp cho chuỗi các token, biểu diễn bởi cây cú pháp và kiểm tra ngôn ngữ theo cú pháp.

Sinh mã trung gian: sau khi phân tích ngữ nghĩa, một số trình biên dịch sẽ tạo ra một dạng biểu diễn trung gian của chương trình nguồn. Chúng ta có thể xem dạng biểu diễn này như một chương trình dành cho một máy trừu tượng. Chúng có hai đặc tính quan trọng: dễ sinh và dễ dịch thành chương trình đích. Dạng biểu diễn trung gian có rất nhiều loại. Thông thường, người ta sử dụng dạng "mã máy 3 địa chỉ" (three-address code), tương tự như dạng

hợp ngữ cho một máy mà trong đó mỗi vị trí bộ nhớ có thể đóng vai trò như một thanh ghi. Mã máy 3 địa chỉ là một dãy các lệnh liên tiếp, mỗi lệnh có thể có tối đa 3 đối số.

Tối ưu mã: giai đoạn tối ưu mã cố gắng cải thiện mã trung gian để có thể có mã máy thực hiện nhanh hơn.

Sinh mã: giai đoạn cuối cùng của biên dịch là sinh mã đích, thường là mã máy hoặc mã hợp ngữ. Các vị trí vùng nhớ được chọn lựa cho mỗi biến được chương trình sử dụng. Sau đó, các chỉ thị trung gian được dịch lần lượt thành chuỗi các chỉ thị mã máy. Vấn đề quyết định là việc gán các biến cho các thanh ghi.

Quản lý bảng ký hiệu: một nhiệm vụ quan trọng của trình biên dịch là ghi lại các định danh được sử dụng trong chương trình nguồn và thu thập các thông tin về các thuộc tính khác nhau của mỗi định danh. Những thuộc tính này có thể cung cấp thông tin về vị trí lưu trữ được cấp phát cho một định danh, kiểu của định danh và nếu định danh là tên của một thủ tục thì thuộc tính là các thông tin về số lượng và kiểu của các đối số, phương pháp truyền đối số và kiểu trả về của thủ tục nếu có.

Bảng ký hiệu (symbol table) là một cấu trúc dữ liệu mà mỗi phần tử là một mẫu tin dùng để lưu trữ một định danh, bao gồm các trường lưu giữ ký hiệu và các thuộc tính của nó. Cấu trúc này cho phép tìm kiếm, truy xuất danh biểu một cách nhanh chóng.

Trong quá trình phân tích từ vựng, danh biểu được tìm thấy và nó được đưa vào bảng ký hiệu nhưng nói chung các thuộc tính của nó có thể chưa xác định được trong giai đoạn này.

Xử lý lỗi: Mỗi giai đoạn có thể gặp nhiều lỗi, tuy nhiên sau khi phát hiện ra lỗi, tùy thuộc vào trình biên dịch mà có các cách xử lý lỗi khác nhau, chẳng hạn :

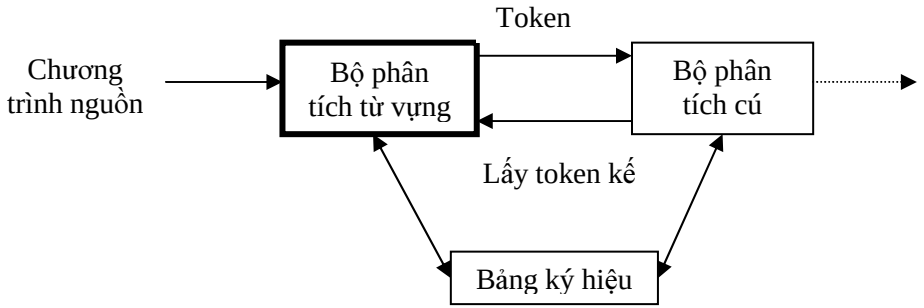
- Dừng và thông báo lỗi khi gặp lỗi đầu tiên (Pascal).
- Ghi nhận lỗi và tiếp tục quá trình dịch (C).

Giai đoạn phân tích từ vựng thường gặp lỗi khi các ký tự không thể ghép thành một token. Giai đoạn phân tích cú pháp gặp lỗi khi các Token không thể kết hợp với nhau theo đúng cấu trúc ngôn ngữ. Giai đoạn phân tích ngữ nghĩa báo lỗi khi các toán hạng có kiểu không đúng yêu cầu của phép toán hay các kết cấu không có nghĩa đối với thao tác thực hiện mặc dù chúng hoàn toàn đúng về mặt cú pháp.

1.2. Phân tích từ vựng

1.2.1. Khái niệm

Phân tích từ vựng là giai đoạn đầu tiên của mọi trình biên dịch. Nó có chức năng là phân tích chương trình nguồn. Nhiệm vụ chủ yếu của nó là đọc các ký hiệu nhập rồi tạo ra một chuỗi các Token được sử dụng bởi bộ phân tích cú pháp. Sự tương tác này được thể hiện như hình sau:



Hình 1.3 Giao diện của bộ phân tích từ vựng

Trong đó bộ phân tích từ vựng được thiết kế như một thủ tục được gọi bởi bộ phân tích cú pháp, trả về một token khi được gọi. Bảng ký hiệu (symbol table) là một cấu trúc dữ liệu mà mỗi phần tử là một mẩu tin dùng để lưu trữ một định danh, bao gồm các trường lưu giữ ký hiệu và các thuộc tính của nó. Cấu trúc này cho phép tìm kiếm, truy xuất danh biểu một cách nhanh chóng.

1.2.2. Ứng dụng

1.2.2.1. Một số vấn đề liên quan đến giai đoạn phân tích từ vựng

1.2.2.2. Đặc tả thẻ từ (Specification of Token)

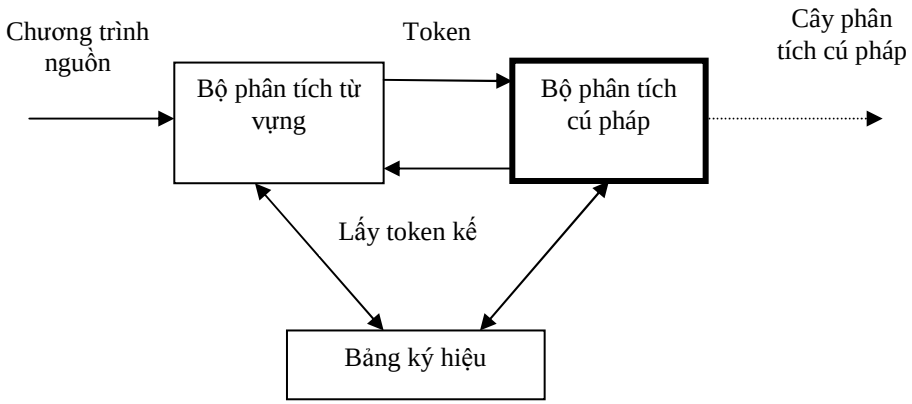
1.2.3. Một số phương pháp, công cụ hiện có

1.3. Phân tích cú pháp

1.3.1. Khái niệm

Mỗi ngôn ngữ lập trình đều có các quy tắc diễn tả cấu trúc cú pháp của các chương trình có định dạng đúng. Các cấu trúc cú pháp này được mô tả bởi *văn phạm phi ngữ cảnh*. Bộ phân tích cú pháp có chức năng là phân tích cú pháp chương trình nguồn. Nhiệm vụ chủ yếu của nó là nhận chuỗi các Token từ bộ phân tích từ vựng và xác nhận rằng chuỗi này có thể được sinh ra từ văn phạm của ngôn ngữ nguồn bằng cách tạo ra cây phân tích cú pháp cho chuỗi. Bộ phân tích cú pháp cũng có cơ chế ghi nhận các lỗi cú pháp theo một phương thức linh hoạt và có khả năng phục hồi được các lỗi thường gặp để có thể tiếp tục xử lý phần còn lại của chuỗi đầu vào.

Bảng ký hiệu (symbol table) là một cấu trúc dữ liệu mà mỗi phần tử là một mẫu tin dùng để lưu trữ một định danh, bao gồm các trường lưu giữ ký hiệu và các thuộc tính của nó. Cấu trúc này cho phép tìm kiếm, truy xuất danh biểu một cách nhanh chóng.



Hình 1.10 Giao diện của bộ phân tích cú pháp trong trình biên dịch.

1.3.2. Ứng dụng

1.3.2.1. Lỗi và các chiến lược phục hồi lỗi của giai đoạn phân tích cú pháp

1.3.2.2. Phân tích cú pháp từ trên xuống

1.3.3. Một số phương pháp, công cụ hiện có

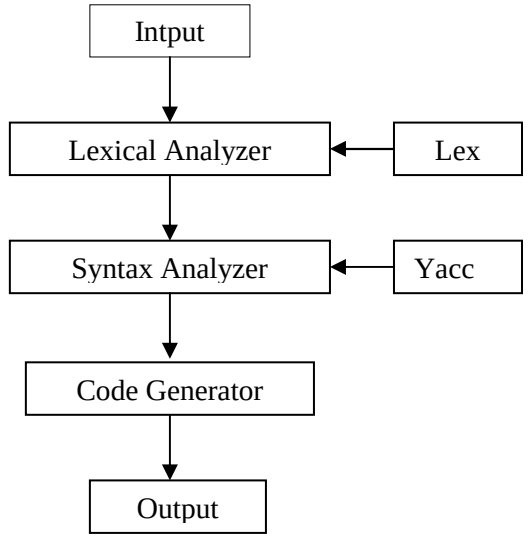
.

Chương 2. NGHIÊN CỨU CÔNG CỤ LEX/YACC

Chương này trình bày những kết quả nghiên cứu về việc sử dụng công cụ Lex và Yacc. Phần đầu giới thiệu mối liên hệ giữa công cụ Lex, Yacc với trình biên dịch và các phần tiếp theo được sử dụng để mô tả một cách chi tiết về Lex và Yacc.

2.1. Giới thiệu

Công cụ Lex và Yacc được sử dụng để phân tích từ vựng và phân tích cú pháp. Lex và yacc có mối liên hệ qua lại với nhau, tìm hiểu ở Hình 2.1.



Hình 2.1 Mối liên hệ giữa Lex và Yacc

Trong đó, Input là xâu ký tự đầu vào. Lexical Analyzer là bộ phân tích từ vựng được công cụ Lex phân tích. Bộ phân tích này có

chức năng chuyển xâu ký tự ở đầu vào thành các Token. Syntax Analyzer là bộ phân tích cú pháp được công cụ Yacc phân tích. Bộ phân tích này có chức năng nhận các Token từ bộ phân tích cú pháp và phân tích tạo thành cây cú pháp. Code Generator có chức năng sinh mã từ cây cú pháp được cung cấp bởi bộ phân tích cú pháp. Output là đoạn mã được sinh ra.

2.2. LEX

2.2.1. Giới thiệu

Lex là một bộ lập chương trình được thiết kế để xử lý từ vựng của những dòng ký tự đầu vào. Nó chấp nhận ngôn ngữ bậc cao, đặt tả định hướng cho việc so khớp chuỗi ký tự, và sinh ra chương trình đích phục vụ việc đoán nhận biểu thức chính quy. Biểu thức chính quy được chỉ định bởi người dùng từ nguồn đặc tả được cho bởi Lex. Lex viết mã đoán nhận những biểu thức ở dòng được nhập vào và phân chia dòng được nhập vào trong những chuỗi so khớp với biểu thức. Tập tin nguồn Lex là sự tập hợp những biểu thức chính quy và các đoạn chương trình. Khi mỗi biểu thức xuất hiện trong đầu vào tới chương trình được viết bởi Lex, đoạn tương ứng được thực thi.

Người sử dụng cung cấp bổ sung thêm mã ngoài biểu thức cho khớp mẫu với thao tác của nó cần hoàn thành, bao gồm mã được viết bởi bộ sinh mã khác. Chương trình đoán nhận những biểu thức được phát sinh trong ngôn ngữ lập trình mục đích chung được gọi như những đoạn chương trình của người sử dụng. Như vậy, một ngôn ngữ bậc cao được cung cấp để viết những biểu thức dạng chuỗi sẽ được so khớp với biểu thức được sinh ra thì không có sự khác biệt. Điều này tránh bắt buộc người sử dụng muốn sử dụng một ngôn ngữ thao tác chuỗi cho việc phân tích đầu vào để viết những chương trình xử lý trong chuỗi như thế và thường không thích hợp trình bày ngôn ngữ dạng chuỗi.

Lex không là một ngôn ngữ đầy đủ, nhưng khá tốt cho việc phát sinh mã đang đại diện cho một đặc tính ngôn ngữ mới mà có thể phù hợp những ngôn ngữ lập trình khác nhau, gọi là ngôn ngữ chủ (host languages). Chỉ có chức năng ngôn ngữ chung có thể sinh mã để chạy trên máy tính khác nhau, Lex có thể viết mã trong những ngôn ngữ chủ khác nhau. Ngôn ngữ chủ được sử dụng cho sinh mã đích bởi Lex và cũng cho những đoạn chương trình thêm bởi người sử dụng. Những thư viện thực thi tương thích cho những ngôn ngữ chủ khác nhau cũng được cung cấp. Điều này làm Lex có thể tương thích đối với những môi trường và người sử dụng khác nhau. Mỗi ứng dụng có thể trực tiếp kết hợp tới phần cứng và thao tác với ngôn ngữ chủ thích hợp. Hiện nay, ngôn ngữ chủ được hỗ trợ như là C. Lex tích hợp trên UNIX, GCOS, OS/ 370, nhưng Lex phát sinh mã bất cứ với trình biên dịch tồn tại thích hợp.

2.2.2. Các chức năng

2.2.2.1. Nguồn Lex (Lex Source)

Khuôn dạng chung của nguồn Lex gồm ba phần như sau:

```
{definitions}
%%
{rules}
%%
{user subroutines}
```

Phần định nghĩa (definitions): đặt ở phần đầu chương trình, dùng để định nghĩa các cú pháp của công thức, các biến, các kiểu,…

Phần luật (rules): phần này được đặt trong cặp dấu %%, trình bày nội dung các luật.

Phần chương trình con (user subroutines): đặt ở phần cuối chương trình, trình bày các hàm, các thủ tục con, …

2.2.2.2. Những biểu thức chính quy Lex (Lex Regular Expressions)

2.2.2.3. Những hoạt động Lex (Lex Actions)

2.2.2.4. Những định nghĩa nguồn Lex (Lex Source Definitions)

2.2.2.5. Tóm lược của nguồn định dạng (Summary of Source Format)

2.2.3. Cách sử dụng

2.2.4. Nhận xét

2.3. YACC

2.3.1. Giới thiệu

Yacc cung cấp một công cụ chung để vận dụng cấu trúc trên đầu vào tới một chương trình máy tính. Yacc chuẩn bị sử dụng đặt tả của quá trình được nhập vào, điều này bao gồm những luật đang mô tả cấu trúc được nhập vào, mã sẽ được kéo theo khi những luật được đoán nhận, và một mức thấp thông thường để làm đầu vào cơ bản. Yacc sau đó phát sinh một hàm để kiểm soát quá trình được nhập vào. Chức năng này, gọi là bộ phân tích, lệnh do người sử dụng cung cấp là thủ tục đầu vào bậc thấp (bộ phân tích từ vựng) để thu nhặt những mục cơ bản (gọi là những Token) từ dòng được nhập vào. Những token này được tổ chức theo những luật cấu trúc được nhập vào, gọi là những luật ngữ pháp, khi một trong số luật này đã được đoán nhận, rồi sử dụng mã được cung cấp cho luật này, một hoạt động được kéo theo, những hoạt động có những khả năng trả lại những giá trị và làm sử dụng những giá trị của hoạt động khác.

Yacc được viết trong một ngữ pháp linh hoạt, những hoạt động và thủ tục con đầu ra, cũng như trong C. Hơn nữa, nhiều quy ước cú pháp của Yacc đi theo sau C.

2.3.2. Các chức năng

2.3.2.1. Những đặc tả cơ bản (Basic Specifications)

Những tên tham chiếu tới những Token hoặc những ký hiệu chưa kết thúc. Yacc yêu cầu những tên token như đã được khai báo. Ngoài ra, những lý do được bàn luận Trong mục 2.3.2.3, nó thường

ước lượng để bao gồm bộ phân tích từ vựng như một phần của đặc tả tệp. Nó có thể hữu ích đến bao gồm các chương trình tốt khác. Như vậy, mọi đặc tả tệp gồm có ba phần: phần khai báo, luật(ngữ pháp) và chương trình. Những phần được tách ra bởi hai dấu “%%". Nói cách khác, một đặc tả đầy đủ về tệp

```
 declarations
%%
rules
%%
programs
```

Phần khai báo có thể trống rỗng. Hơn nữa, nếu phần chương trình bị bỏ sót, hai dấu %% cũng có thể bị bỏ sót. Như vậy, đặc tả Yacc nhỏ nhất hợp lệ là

```
%%
rules
```

những chỗ trống, những bảng, và những dòng mới được lờ đi chỉ có điều chúng có thể không xuất hiện trong những tên hoặc những ký hiệu được lưu trữ nhiều đặc tính. Những chú dẫn có thể xuất hiện ở mọi nơi là một tên thì hợp lệ, chúng được bao bởi cặp /*. . . */, như trong C.

2.3.2.2. Những hoạt động (Actions)

2.3.2.3. Sự phân tích từ vựng (Lexical Analysis)

2.3.2.4. Bộ phân tích làm việc như thế nào (How the Parser Works)

2.3.2.5. Sự nhập nhằng và những xung đột (Ambiguity and Conflicts)

2.3.2.6. Mức ưu tiên (Precedence)

2.3.3. Cách sử dụng

2.3.4. Nhận xét

Chương 3. NGHIÊN CỨU ỨNG DỤNG LEX/YACC

Chương này tập trung nghiên cứu quy trình vận dụng Lex và Yacc và đưa ra ví dụ về cách sử dụng. Để sử dụng Lex và Yacc chúng ta cần cài đặt các công cụ hỗ trợ biên dịch và phát sinh mã nguồn tương ứng với từng giai đoạn.

3.1. Cài đặt các ứng dụng

Hiện tại có rất nhiều ứng dụng hỗ trợ xử lý cho Lex và Yacc, tuy nhiên trong thử nghiệm của mình chúng tôi đã cài đặt các công cụ sau:

3.1.1. Bison

Bison là một bộ phân tích cú pháp theo tiêu chuẩn đặc tả của Yacc. Nó sẽ phát sinh tự động một chương trình xử lý tương ứng với các tập tin đầu vào thiết kế cho Yacc.

Bison được viết bởi Robert Corbett và Richard Stallman trong dự án mã nguồn mở để xây dựng bộ phân tích cú pháp theo tiêu chuẩn đặc tả của Yacc. Chúng ta có thể tải phần mềm mã nguồn mở này tại trang http://www.gnu.org/software/bison/.

Tập tin đầu vào cần thực hiện theo các quy ước của Yacc và tên tập tin được đặt tùy ý nhưng có phần mở rộng là ***.y*** (ví dụ: ***calc.y***). Không giống như Yacc gốc, các tập tin được tạo ra không có tên cố định, nhưng thay vào đó là nó sử dụng các tiền tố của tập tin đầu vào. Hơn nữa, nếu chúng ta cần phải đặt mã lệnh C++ trong tập tin đầu vào thì có thể kết thúc tên của tập tin bằng phần mở rộng giống như của C++ (ví dụ: ***.ypp*** hoặc ***.y++***), sau đó Bison sẽ theo phần mở rộng của chúng ta để đặt tên cho tập tin xuất (***.cpp*** hoặc ***.c++***). Ví dụ, một mô tả ngữ pháp tập tin có tên ***parse.yxx*** sẽ cho bộ phân tích cú pháp tạo ra trong một tập tin với tên tương ứng ***parse.tab.cxx***.

3.1.2. Flex

Flex là một công cụ để tạo ra các bộ kiểm tra nhằm công nhận các mẫu từ vựng trong tập tin đầu vào được đặc tả theo tiêu chuẩn của Lex. Flex đọc các tập tin đầu vào cho trước viết theo tiêu chuẩn đặc tả Lex và tự động sinh ra chương trình trong mã nguồn được chọn (thường là C/C++) nhằm kiểm tra tính đúng đắn và thực thi các qui tắc tính toán. Mô tả viết ở dạng các cặp biểu thức thông thường và mã C, gọi là các quy tắc. Flex tạo ra kết quả là một tập tin mã nguồn C (mặc định với tên gọi là ***lex.yy.c***). Tập tin này được biên dịch và liên kết với thư viện ***–lfl*** để sản xuất một thực thi. Khi thực thi được chạy, nó phân tích đầu vào của các biểu thức thông thường và khi biểu thức nhập vào đúng đắn thì sẽ thực thi các đoạn mã C/C++ tương ứng.

Flex là phần mềm được cung cấp bởi Đại học California và phiên bản đầu tiên được cấp phép vào năm 1990 theo tiêu chuẩn mã nguồn mở. Sau đó, mã nguồn này được tiếp tục phát triển và phân phối miễn phí bởi Vern Paxson thuộc Đại học Berkeley. Chúng ta có thể tải Flex tại http://www.gnu.org/software/flex/.

3.1.3. Dev-C++

Dev-C++ là một môi trường phát triển tích hợp (IDE) khá đầy đủ các tính năng dành cho các ngôn ngữ lập trình C/C++. Nó sử dụng MinGW của GCC (GNU Compiler Collection) làm trình biên dịch các tập tin chương trình. Dev-C++ cũng có thể được sử dụng kết hợp với Cygwin hay bất kỳ trình biên dịch GCC khác.

Một số tính năng chính của Dev-C++ là:

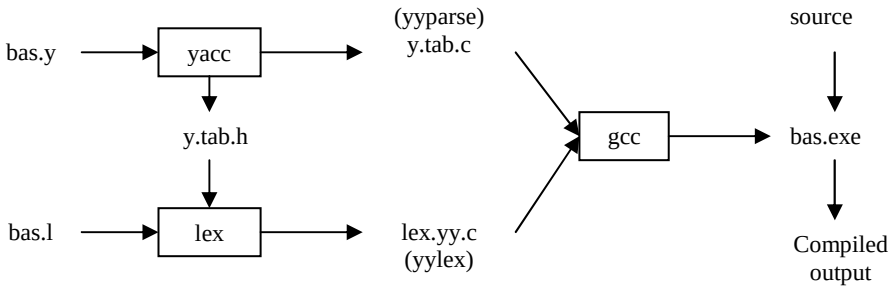
- Hỗ trợ các trình biên dịch dựa GCC.
- Tích hợp sửa lỗi (bằng cách sử dụng GDB).
- Quản lý dự án.
- Tùy chỉnh các chế độ soạn thảo và phát hiện lỗi cú pháp trong chương trình.
- Duyệt các lớp (Class Browser).
- Nhanh chóng tạo ra các cửa sổ, giao diện điều khiển, thư viện tĩnh và động (các tập tin DLL).
- Hỗ trợ các mẫu cho việc tạo ra các loại dự án của riêng bạn.

- Cho phép tạo ra các Makefile.
- Chỉnh sửa và biên dịch tập tin mã nguồn.
- Hỗ trợ CVS.

Chúng ta có thể tải Dev-C++ từ http://www.bloodshed.net/devcpp.html.

3.2. Qui trình vận dụng LEX/YACC

Quy trình chung để sử dụng Yacc và Lex khi phát sinh ứng dụng được thể hiện qua sơ đồ sau:



Hình 3.1 Minh họa quy trình vận dụng và cách đặt tên với Lex và Yacc

Đây là các bước mà chúng ta sẽ thực hiện khi muốn viết một trình biên dịch trong ngôn ngữ lập trình C/C++. Đầu tiên, chúng cần mô tả tất cả mẫu đang so khớp những luật cho Lex (**bas.l**) và luật ngữ pháp cho Yacc (**bas.y**). Những lệnh để tạo ra trình biên dịch với tên **bas.exe.** Các lệnh thực hiện như sau:

```
bison –d bas.y            # sinh ra y.tab.h, y.tab.c
Flex bas.l                # tệp được sinh ra lex.yy.c
gcc lex.yy.c y.tab.c –o bas.exe  # biên dịch tạo tệp .exe
```

Yacc đọc sự mô tả ngữ pháp trong **bas.y** và phát sinh một bộ phân tích cú pháp (bộ phân tích), mà bao gồm hàm **yyparse**, trong tệp **y.tab.c**. Được bao gồm trong tệp bas.y là token khai báo. Tuỳ chọn **-d** gây ra yacc để phát sinh những định nghĩa cho những token và đặt chúng trong tệp **y.tab.h**. Lex đọc sự mô tả mẫu trong **bas.l**, bao gồm tệp **y.tab.h**, và phát sinh một bộ phân tích từ vựng, mà bao gồm hàm **yylex**, trong tệp **lex.yy.c**.

Cuối cùng, **gcc** (GNU Compiler Collection) liên kết để tạo ra

bas.exe có thể thực thi được. Từ phần chính chúng gọi **yyparse** để chạy trình biên dịch. Hàm **yyparse** tự động gọi **yylex** để thu được mỗi token.

Cụ thể các bước như sau:

Bước 1: Soạn thảo phần đặt tả Yacc trên công cụ Notepad và lưu lại tên tệp với phần mở rộng **.y** (ví dụ **bas.y**), bỏ vào thư mục bin của Bison vừa cài đặt ở trên.

Bước 2: Soạn thảo phần đặt tả Lex trên công cụ Notepad và lưu lại tên tệp với phần mở rộng **.l** (ví dụ **bas.l**), bỏ vào thư mục bin của Flex vừa cài đặt ở trên.

Bước 3: Thực hiện biên dịch trên môi trường Win với Command Prompt như sau: Từ cửa sổ hệ điều hành Win, chọn **Start → Programs → Accessories → Command Prompt**, hộp thoại **Command Prompt** xuất hiện, chuyển dấu nhắt con trỏ về thư mục gốc **C:\>**.

Bước 3.1. Thực hiện lệnh biên dịch cho tệp Yacc. Chuyển con trỏ về thư mục **bin** của Bison đã được cài đặt **C:\>bison\bin** và thực hiện theo cú pháp:

```
Bison {options} source-file
```

Trong đó: **Bison** là công cụ đặt tả Yacc.

source-file là tên của tệp được đặt tả bởi nguồn Yacc, với phần đuôi mở rộng **.y**.

options là những tùy chọn có thể được chỉ rõ tập tin được sinh ra từ dòng lệnh, được cho bởi bảng sau:

Bảng 3.1 Mô tả các tùy chọn (options)

Tuỳ chọn (options)	Mô tả
-o	Tên tệp được sinh ra.
-c	Sử dụng tên tệp thích hợp lex.yy
-d	Bao gồm phần tên tệp mở rộng với đuôi .h tệp thư viện và đuôi .c tệp chương trình mặt định cho C và C++
…	…

Yacc sinh ra tệp thư viện với tên phần mở rộng **.h** và tệp chương trình xử lý với tên phần mở rộng **.c**.

Bước 3.2. Thực hiện lệnh biên dịch cho tệp Lex. Chuyển con trỏ về thư mục **bin** của Flex đã được cài đặt **C:\>Flex\bin** và thực hiện theo cú pháp:

```
FLex source-file
```

Trong đó: **Flex** là công cụ đặt tả Lex.

source-file là tên của tệp được đặt tả bởi nguồn Lex, với phần đuôi mở rộng **.l**. Tệp được Lex sinh ra với tên phần mở rộng **.c**.

Bước 3.3. Thực hiện liên kết Lex và Yacc bằng cách sử dụng Dev-C++ đã được cài đặt ở trên để sinh ra tệp đích thực thi được với phần đuôi mở rộng **.exe**, theo cú pháp sau:

```
gcc lexname.c yaccname.c –o<name>
```

Trong đó:**gcc** là tệp hỗ trợ biên dịch các tệp chương trình của Dev-C++.

lexname.c là tệp Lex vừa được sinh ra ở trên.

yaccname.c là tệp Yacc vừa được sinh ra ở trên.

-o <name> là tên tệp đích được sinh ra với phần mở rộng **.exe**.

3.3. Ứng dụng thử nghiệm

3.3.1. Phát biểu bài toán

Xây dựng một chương trình cho phép tính toán một biểu thức số học nhập vào từ bàn phím (dưới dạng 1 xâu ký tự) và in ra kết quả sau khi tính toán biểu thức đó. Ví dụ: nhập vào xâu ***"2+(3+4)*2"*** và kết quả trả về là ***16***.

3.3.2. Các bước triển khai

Về mặt hình thức, ta có thể định nghĩa cú pháp của công thức ở mức đơn giản như sau:

```
hằngsố = chuỗi từ 1 tới n ký số thập phân
 toántử = + | - | * | /
 dấu ngăn = ( | )
 côngthức = hằngsố
          | (côngthức)
          | côngthức + côngthức
          | côngthức - côngthức
          | côngthức * côngthức
          | côngthức / côngthức
```


Để triển khai ứng dụng bằng Lex/Yacc, trước hết chúng ta phải đặc tả các biểu thức chính qui miêu tả các token được dùng để xây dựng công thức và đặc tả cú pháp của công thức và tính giá trị công thức. Tiếp đó, dùng các công cụ như Bison hoặc Ayacc, Lex hoặc Flex và công cụ sinh mã cygwin để phát sinh chương trình thực thi.

Cụ thể các bước triển khai như sau:

Bước 1: Sử dụng hệ soạn thảo bất kỳ để tạo tệp tin calc.y chứa nội dung theo ngôn ngữ đặc tả Yacc để đặc tả cú pháp của công thức và tính giá trị công thức như sau:

```
%{
    #include <stdio.h>
    void yyerror(char *);
    int yylex(void);
    int sym[26];
%}
%token INTEGER VARIABLE
%left '+' '-'
%left '*' '/'
%%
program:
        program statement '\n'
        | /* NULL */
        ;
statement:
        expression { printf("%d\n", $1); }
        | VARIABLE '=' expression { sym[$1] = $3; }
        ;
expression:
        INTEGER
        | VARIABLE                  { $$ = sym[$1]; }
        | expression '+' expression  { $$ = $1 + $3; }
        | expression '-' expression  { $$ = $1 - $3; }
        | expression '*' expression  { $$ = $1 * $3; }
        | expression '/' expression  { $$ = $1 / $3; }
        | '(' expression ')'         { $$ = $2; }
        ;
%%
void yyerror(char *s)
```

```
{
    fprintf(stderr, "%s\n", s);
}
int main(void)
{
    yyparse();
}
```

Bước 2: Sử dụng hệ soạn thảo bất kỳ để tạo tệp tin calc.l chứa nội dung theo ngôn ngữ đặc tả Lex để đặc tả các biểu thức chính qui miêu tả các token được dùng để xây dựng công thức như sau:

```
/* calculator #1 */
%{
    #include "y.tab.h"
    #include <stdlib.h>
    void yyerror(char *);
%}
%%
[a-z]       {
                yylval = *yytext - 'a';
                return VARIABLE;
                }
[0-9]+      {
                yylval = atoi(yytext);
                return INTEGER;
            }
[-+()=/*\n] { return *yytext; }

[ \t]   ;       /* skip whitespace */

.               yyerror("Unknown character");
%%
int yywrap(void)
{
    return 1;
}
```

Trong nội dung của tệp tin calc.l có chứa khai báo #include "y.tab.h" và đây là thư viện được tự động sinh ra từ tệp calc.y thông qua công cụ xử lý của Yacc (trong trường hợp này chúng tôi sử dụng phần mềm Bison).

Bước 3: sử dụng các công cụ để sinh mã tự động như sau:

Bước 3.1. Thực hiện lệnh biên dịch cho tệp Yacc:

```
bison –d calc.y
```


Phần mềm này sẽ tự động phát sinh ra 2 tệp. Tệp thư viện y.tab.h và tệp chương trình y.tab.c trong C với nội dung như sau:

y.tab.h có nội dung:

```
/* Tokens.  */
#ifndef YYTOKENTYPE
# define YYTOKENTYPE
enum yytokentype
   {
     INTEGER = 258,
     VARIABLE = 259
   };
#endif
#if    !    defined    YYSTYPE    &&    !    defined
YYSTYPE_IS_DECLARED
typedef int YYSTYPE;
# define YYSTYPE_IS_TRIVIAL 1
# define yystype YYSTYPE /* obsolescent; will be
withdrawn */
# define YYSTYPE_IS_DECLARED 1
#endif
extern YYSTYPE yylval;
```

và tệp chương trình xử lý y.tab.c có nội dung như trong phụ lục 1.

Bước 3.2: Thực hiện lệnh biên dịch cho tệp ***Lex***:

```
Flex calc.l
```

Lúc này Flex sẽ sinh ra một tệp lex.yy.c có nội dung như sau:

```
#define YY_FLEX_MAJOR_VERSION 2
#define YY_FLEX_MINOR_VERSION 5
#include <stdio.h>
/*  cfront  1.2  defines  "c_plusplus"  instead  of
"__cplusplus" */
#ifdef c_plusplus
#ifndef __cplusplus
#define __cplusplus
#endif
#endif
…
```

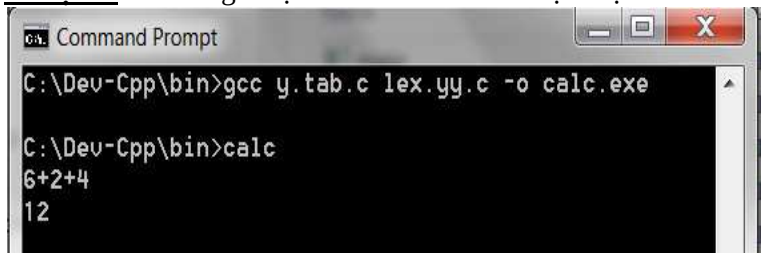
Nội dung đầy đủ của tệp lex.yy.c được trình bày trong phụ lục 2.

Bước 3.3. Sử dụng trình biên dịch ***gcc*** của Dev-Cpp để thực hiện liên kết Lex và Yacc để sinh ra tệp thực thi với phần mở rộng là ***.exe***:

```
gcc y.tab.c lex.yy.c –o calc.exe
```

Lúc này máy tính sẽ sản sinh ra tệp thực thi calc.exe và khi thực hiện nó thì ta có thể nhập vào một biểu thức và máy sẽ cho kết quả là giá trị của biểu thức đó.

Ví dụ 3.1: để tính giá trị biểu thức ***6+2+4*** ta thực hiện như sau:



Ví dụ 3.2: để tính giá trị biểu thức ***(4*3-8/2)*5+7*** ta thực hiện như sau:

3.4. Đánh giá

Việc tính giá trị của một biểu thức có thể thực hiện bằng nhiều cách khác nhau. Ví dụ, ta có thể sử dụng kỹ thuật Kí pháp Ba Lan do nhà Lô-gíc toán Jan Łukasiewicz đề xuất khoảng năm 1920.

Việc tính giá trị một biểu thức viết dưới dạng phép toán sau rất thuận tiện như trên, tuy nhiên, theo thói quen thông thường, việc nhập biểu thức đó vào lại không dễ, người ta thường nhập vào một công thức dưới dạng thông thường (phép toán giữa) rồi dùng chương trình chuyển đổi nó sang dạng phép toán sau. Chúng ta hãy xét biểu thức trong ví dụ trên ***Q=a*(b+c)-d^5***

Kí hiệu biểu thức ghi dưới dạng phép toán sau là P. Trong quá trình chuyển đổi ta dùng một stack S để lưu các phần tử trong P chưa sử dụng đến. Khi đọc từ trái sang phải biểu thức Q là lần lượt có:

1. Đọc và ghi nhận giá trị a, ghi giá trị a vào P. Vậy P = "a".
2. Đọc toán tử "*". Đưa toán tử này vào stack S: S = "*"
3. Đọc dấu ngoặc mở "(", đưa dấu ngoặc này vào stack: S = "*(".
4. Đọc hạng tử b, đưa b vào P: P= "a b"
5. Đọc toán tử "+", đặt "+" vào stack: S ="*(+"
6. Đọc hạng tử "c", đưa c vào cuối P: P="a b c"
7. Đọc dấu ngoặc đóng ")". Lần lượt lấy các toán tử ở cuối stack ra khỏi stack đặt vào cuối P cho đến khi gặp dấu ngoặc mở "(" trong stack thì giải phóng nó: S= "*"; P="a b c +"
8. Đọc toán tử "-". Cuối stack S có toán tử "*" có mức ưu tiên lớn hơn toán tử "-", ta lấy toán tử "*" ra khỏi stack, đặt vào cuối P, đặt toán tử "-" vào stack: S="-"; P=" a b c + * "
9. Đọc hạng tử d, đưa d vào cuối P. P="a b c + * d"
10. Đọc toán tử "^", đưa toán tử "^" vào cuối stack: S="-^"
11. Đọc hằng số 5, đưa 5 vào cuối P: P="a b c + * d 5"
12. Đã đọc hết biểu thức Q, lần lượt lấy các phần tử cuối trong stack đặt vào P cho đến hết. P="a b c + * d 5 ^ -".

Thuật toán chuyển từ ký pháp trung tố sang ký pháp tiền tố hoặc hậu tố rất gần với cách xử lý các phép tính trong máy tính bấm tay (hay máy tính bỏ túi). Một biểu thức chỉ gồm các phép toán hai ngôi bất kỳ luôn có thể được tính bằng máy tính bấm tay mà không cần dùng dấu ngoặc. Các phép toán ở trước nếu có độ ưu tiên (ưu tiên bởi toán tử hoặc bởi dấu ngoặc) thấp hơn một phép toán ở sau được đẩy vào một hàng chờ (stack), chỉ khi nào các phép toán ưu tiên hơn ở sau được tính xong, các phép toán ở trước mới được xử lý.

Việc lập trình theo kỹ thuật này rất phức tạp và tốn nhiều thời gian. Ngoài ra, người sử dụng phải tự viết mã lệnh rất nhiều nên tốn nhiều thời gian và dễ sai sót khi xét các trường hợp.

Chúng ta so sánh như vậy để thấy rằng việc ứng dụng Lex/Yacc trong trường hợp này, cũng như trong một số trường hợp khác, là rất có lợi.

3.5. Xây dựng Website giới thiệu về Lex/Yacc

3.5.1. Giới thiệu

Việc sử dụng Lex/Yacc thường được giới thiệu trong một số môn học như Chương trình dịch, Kỹ thuật lập trình, ...nhưng chưa được trình bày một cách đầy đủ nhất về lý thuyết cũng như trong ứng dụng.

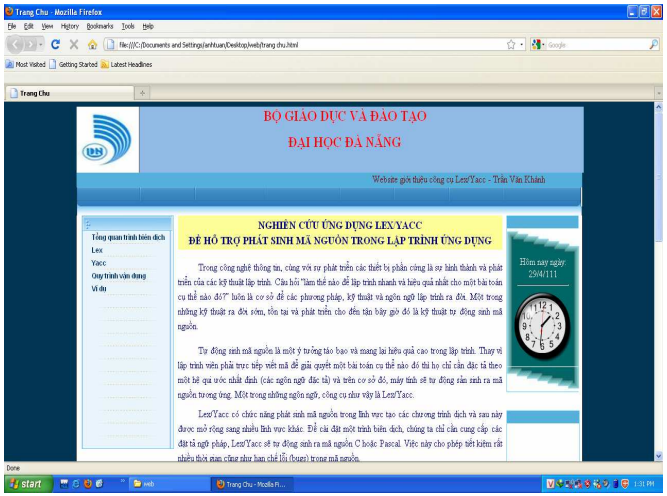
Trong quá trình tìm hiểu và nghiên cứu gặp nhiều khó khăn về lý thuyết và nhất là trong triển khai ứng dụng đối với công cụ này.

Mục đích xây dựng Website là nhằm giới thiệu đầy đủ các tính năng của Lex/Yacc và giúp tiếp cận, nắm bắt đầy đủ về quy trình vận dụng chúng trong môi trường Win.

3.5.2. Thiết kế Website

3.5.3. Website giới thiệu công cụ Lex/Yacc

Để tìm hiểu về công cụ Lex/Yacc và quy trình vận dụng nó trên trang web được trình bày bao hàm các nội dung từ giao diện trang chủ như sau:



Hình 3.2 Trang chủ của Website giới thiệu về Lex/Yacc

KẾT LUẬN

Hiện nay, các trường phổ thông và các trường đại học trong nước hầu hết đã đưa ngôn ngữ lập trình bậc cao như C/C++, Pascal..., vào giảng dạy. Vấn đề đặt ra là làm thế nào để giảm chi phí về nghiên cứu, tiếp cận và tăng hiệu quả sử dụng các ngôn ngữ lập trình bậc cao.

Tự động sinh mã nguồn là một ý tưởng táo bạo và mang lại hiệu quả cao trong lập trình. Thay vì lập trình viên phải trực tiếp viết mã để giải quyết một bài toán cụ thể nào đó thì họ chỉ cần đặc tả theo một hệ qui ước nhất định (các ngôn ngữ đặc tả) và trên cơ sở đó, máy tính sẽ tự động sản sinh ra mã nguồn tương ứng. Một trong những ngôn ngữ, công cụ như vậy là Lex/Yacc.

Qua quá trình tìm hiểu và nghiên cứu ứng dụng công cụ Lex/Yacc để phát sinh mã nguồn trong lập trình ứng dụng đã cho thấy kết quả tương đối tốt. Công cụ này đã cho thấy những ưu điểm, sự tiện lợi, có khả năng ứng dụng trong thực tiễn ở Việt Nam. Với công cụ này, học sinh, sinh viên các trường có điều kiện tiếp cận và sử dụng có hiệu quả các công cụ lập trình của mình.

Lex/Yacc có chức năng phát sinh mã nguồn trong lĩnh vực tạo các chương trình dịch và sau này được mở rộng sang nhiều lĩnh vực khác. Để cài đặt một trình biên dịch, chúng ta chỉ cần cung cấp các đặt tả ngữ pháp, Lex/Yacc sẽ tự động sinh ra mã nguồn C. Việc này cho phép tiết kiệm rất nhiều thời gian cũng như hạn chế lỗi (bugs) trong mã nguồn.

Như vậy, sau một thời gian tiến hành nghiên cứu, tác giả đã cơ bản hoàn thành các nội dung mà đề cương đề tài đã đặt ra. Sản phẩm đề tài này có thể được ứng dụng tốt trong việc tìm hiểu, nghiên cứu và triển khai trong lập trình ứng dụng đặc biệt là trong dạy và học ở bậc phổ thông và đại học ở nước ta.