**BÁO CÁO ĐỒ ÁN**

**NGHIÊN CỨU VÀ CÀI ĐẶT CHƯƠNG TRÌNH PHÂN TÍCH CÚ PHÁP CỦA CHƯƠNG TRÌNH C**

Môn: Nguyên lí và phương pháp lập trình

Giảng viên: Trịnh Quốc Sơn

Sinh viên thực hiện: Hoàng Đình Hữu

MSSV: 20521384

Lớp: CS111.M21.KHCL

1. **Khái niệm về NNLT C:**

- Khái niệm: Ngôn ngữ lập trình là một ngôn ngữ nhân tạo, có thể sử dụng để điều khiển máy tính và giúp nó có thể hiểu. Ngôn ngữ lập trình cũng giống như ngôn ngữ con người, được định nghĩa thông qua quy tắc cú pháp và ngữ nghĩa để xác định cấu trúc và ý nghĩa tương ứng.

- Lịch sử của C:

Ngôn ngữ C là một trong những ngôn ngữ lập trinh phổ biến và dùng làm phương tiện giảng dạy trong khoa học máy tính:

+ Được phát triển bởi Dennis Ritchie vào năm 1972.

+ Đươc sử dụng rộng rãi và có ảnh hưởng tới một vài ngôn ngữ sau này.

+ Năm 2000, C là một trong hai top ngôn ngữ được sử dụng phổ biến nhất. Chủ yếu sử dụng để lập trình phần mềm hệ thống.

-Ưu điểm:

+ Linh hoạt vì C liên kết chặt chẽ với hệ thống Unix nhưng không bị trói buộc với bất cứ hệ điều hành nào.

+ Thực thi nhanh: C thực thi rất nhanh như hợp ngữ (Assembly). Trình biên dịch C dịch nhanh và cho ra mã đối tượng không lỗi.

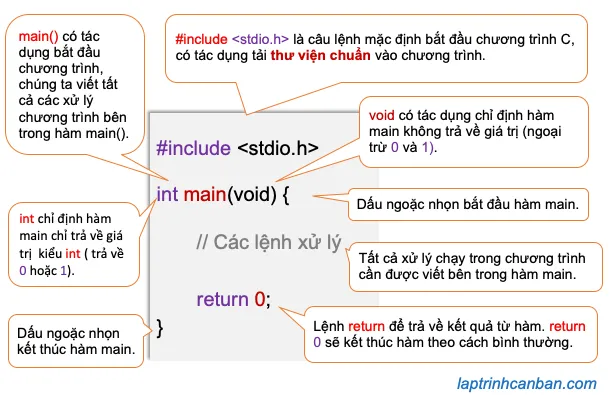
+ Hiệu quả: Trình biên dịch C có sẵn cho hầu hết các máy tinh, có thể tạo ra và bảo trì thư viện hàm để tái sử dụng cho chương trình khác mà không có lỗi.

**II.Nguyên tắc của NNLT C:**

Khi viết chương trình trong C, chúng ta cần tuân thủ một số các quy tắc nhằm xác định cách thức để viết và dịch trong ngôn ngữ lập trình C như sau:

1. Kết thúc câu lệnh bằng dấu chấm than “;”
2. Sử dụng thụt lề (các khoảng trắng đầu câu lệnh) cũng như xuống dòng để làm cấu trúc chương trình dễ hiểu hơn.
3. Sử dụng ký tự // để comment một dòng lệnh, và cặp dấu /\* và \*/ để comment nhiều dòng lệnh. Tham khảo: Comment trong C
4. Comment có tác dụng giúp người đọc mã nguồn C dễ hiểu hơn.
5. Bắt đầu hàm bởi dấu “{“ và kết thúc hàm bởi dấu “}”.

Ví dụ: Cấu trúc cơ bản của 1 chương trình C

****

**III.Phân tích từ vựng và cú pháp của NNLT C:**

**1.Phân tích từ vựng (lexical analysis):**

Lexical analysis: đọc chương trình nguồn từ trái sang phải (linear analysis/scanning) để tách ra thành các từ tố (token).

Đối với ngôn ngữ lập trình C thì từ vựng thường được xây dựng trên bộ chữ gồm có:

* Chữ cái: A…Z , a…z
* Chữ số: 0..9
* Các kí hiệu toán học: +, -, \*, /, (, ), =, <, >, ! , %
* Các kí hiệu toán học khác: [, ], ……

Các từ vựng được ngôn ngữ hiểu bao gồm các từ khóa, tên hàm, tên hằng, tên biến, các phép toán,…

Ví dụ: Câu lệnh viết bằng ngôn ngữ C: a=b+c\*2;

Chương trình phân tích từ vựng sẽ trả về:

* a,b,c là tên định danh
* = là toán tử gán
* + là toán tử cộng
* \* là toán tử nhân
* 2 là một số

Kết quả phân tích từ vựng sẽ là: (tên, a) , phép gán, (tên, b) ,phép cộng, (tên , c), phép nhân, (số, 2).

**2.Phân tích cú pháp (syntax analysis):**

Cú pháp (Syntax): thành phần quan trong nhất của một ngôn ngữ. Ngôn ngữ là tập hợp các câu thỏa mãn văn phạm của ngôn ngữ đó.

Phân tích cú pháp (Syntax analysis): Phân tích cú pháp của chương trình. Các từ tố (token) trở thành các cụm từ văn phạm theo cấu trúc phân cấp:

* Ngôn ngữ được định nghĩa bởi các luật sinh.
* Phân tích cú pháp dựa vào luật sinh để xây dựng cây phân tích cú pháp.

Trong NNLT C, cú pháp của nó được thể hiện bởi một bộ luật cú pháp. Bộ này dùng để mô tả cấu trúc của chương trình, các câu lệnh,…

* Các khai báo
* Biểu thức số học, biểu thức logic
* Các lệnh: lệnh gán, lệnh gọi hàm, …
* Các lệnh điều kiện if
* Câu lệnh lặp: for, while
* Chương trình con

Nhiệm vụ:

* Tìm tất cả các lỗi cú pháp , đưa ra các thông báo lỗi
* Xây dựng cây phân tích cú pháp (lưu vết ) cho chương trình

Có hai kỹ thuật phân tích:

* Top-down : xây dựng cây phân tích cú pháp bắt đầu từ gốc
* Bottom up : xây dựng cây phân tích cú pháp bắt đầu từ lá

Ví dụ: Ngôn ngữ được tả bởi luật sau:

Statement ten: = exp

Experience exp + exp|exp \* exp|ten|so

Áp dụng cho chuỗi a=b+c\*2 ta có cây dẫn xuất như sau:

c

exp

+

exp

b

exp

+

exp

a

exp

:=

ten

statement

2

**Vai trò của bộ phân tích cú pháp**

Đây là giai đoạn thứ 2 của chương trinh dịch (giai đoạn phức tạp nhất)

- Tìm tất cả các lỗi cú pháp , đưa ra các thông báo lỗi

- Xây dựng cây phân tích cú pháp (lưu vết) cho chương trình

**Các mục tiêu của PTCP**

- Chính xác: đây là mục tiêu quan trọng nhất, kết quả

phân tích cần trả về chính xác cây phân tích

- Tốc độ: khó xây dựng các bộ PTCP tuyến tính theo độ

dài của chuỗi vào (ngoại trừ ngôn ngữ đầu vào có văn

phạm quá đơn giản), nhưng bộ PTCP cần hoạt động đủ

nhanh (nên cận tuyến tính)

- Chịu lỗi: bộ PTCP cần có khả năng chịu lỗi và có chiến

lược khắc phục lỗi phù hợp

- Hiệu quả về bộ nhớ: bộ PTCP cần sử dụng bộ nhớ một

cách hiệu quả (do việc phải lưu trữ toàn bộ cây phân

tích cho mã nguồn)**PTCP có thể áp dụng với ngôn ngữ tự nhiên, nhưng:**

-Ngôn ngữ tự nhiên có luật ngữ pháp phức tạp

- Ngôn ngữ tự nhiên có yếu tố ngữ cảnh

- Ngôn ngữ tự nhiên có nhiều lỗi (trong thực tế sử dụng)

**Các bước xây dựng bộ PTCP**

* Mô tả các luật văn phạm của ngôn ngữ nguồn

+ Các mô tả này ban đầu có thể ở dạng ngôn ngữ tự nhiên

+ Đặc tả ý nghĩa các kí hiệu không kết thúc (non-terminal)

+ Chuyển thành các luật văn phạm ở dạng chặt chẽ

* Phân tích bộ văn phạm để lựa chọn phương pháp

phân tích cú pháp phù hợp nhất

+ Văn phạm có nhập nhằng hay không?

+ Văn phạm có đệ quy trái hay không?

+ Văn phạm có sự mơ hồ hay không?

+ Văn phạm có độ phức tạp ở mức độ nào?

* Lựa chọn phương pháp phân tích cú pháp phù hợp

+ Xây dựng bộ PTCP trực tiếp (dành cho các ngôn ngữ có

độ phức tạp đơn giản)

+ Xây dựng bộ PTCP 2 bước

• Dựa trên văn phạm đầu vào, xây dựng automat đoán nhận

• Sử dụng automat để xử lý dãy từ tố từ PTTV

+ Xây dựng bộ PTCP vạn năng: trường hợp văn phạm quá

phức tạp, có thể sử dụng các phương pháp phân tích vạn

năng để xây dựng bộ PTCP

* Lựa chọn cách xử lý trong tình huống lỗi cú pháp,

sinh các gợi ý sửa lỗi và các tình huống cần phải tổ

hợp ngữ nghĩa

Ví dụ: Phân tích cú pháp khai báo kiểu dữ liệu và biến

Input: string x;

Lexical analysis:

(‘string’, specifier), (‘x’,identifier), (‘;’, semicolon)

Grammar:

stmt -> spec id expStmt

expStmt -> semicolon

id -> identifier

spec -> specifier

|  |  |  |  |
| --- | --- | --- | --- |
| Stack | Current Sym | Rest of input | Action |
| $ | specifier | identifier semicolon$ | Shift |
| $specifier | identifier | semicolon$ | Reduce: spec->specifier |
| $spec | identifier | semicolon$ | Shift |
| $spec identifier | semicolon | $ | Reduce: id->identifier |
| $spec id | semicolon | $ | Shift |
| $spec id semicolon | $ |  | Reduce: expStmt-> semicolon |
| $spec id expStmt | $ |  | Reduce: stmt->spec id expStmt |
| $stmt | $ |  | Accept |

Ví dụ 2:

Input: if(a>b){a=b+1;}

Lexical analysis:

(‘a’,var), (‘b’,var), (‘{‘,ocb), (‘+’,op), (‘1’,number), (‘}’,ccb), (‘;’,expStmt)

Note:

ocb – open curly bracket

ccb – close curly bracket

op – operator

Grammar:

stmt -> ifStmt

ifStmt -> if (exp) ocb assign occ

assign -> id = expStmt

expStmt -> id op num expStmt

exp -> id compare id

compare -> ‘>’

num -> number

id -> var

|  |  |  |  |
| --- | --- | --- | --- |
| Stack | Curr Sym | Rest of input | Action |
| $ | if | (var >var) ocb var = var op number expStmt occ$ | Shift |
| $if | ( | var>var) ocb var = var op numer expStmt occ$ | Shift |
| $if ( | var | >var) ocb var = var op numer expStmt occ$ | Shift |
| $if (var | > | var) ocb var = var op numer expStmt occ$ | Reduce: id->var |
| $if (id> | var | ) ocb var = var op numer expStmt occ$ | Shift |
| $if (id> | var | ) ocb var = var op numer expStmt occ$ | Reduce: compare-> ‘>’ |
| $if (id compare var | ) | ocb var = var op numer expStmt occ$ | Shift |
| $if (id compare var | ) | ocb var = var op numer expStmt occ$ | Reduce: id->var |
| $if (id compare id | ) | ocb var = var op numer expStmt occ$ | Reduce: exp->id compare id |
| $if (exp | ) | ocb var = var op numer expStmt occ$ | Shift |
| $if (exp) | ocb | var = var op numer expStmt occ$ | Shift |
| $if (exp) ocb | var | = var op numer expStmt occ$ | Shift |
| $if (exp) ocb var | = | var op numer expStmt occ$ | Reduce: id->var |
| $if (exp) ocb id = | var | op numer expStmt occ$ | Shift |
| $if (exp) ocb id = var | op | numer expStmt occ$ | Shift |
| $if (exp) ocb id = var | op | numer expStmt occ$ | Reudce: id->var |
| $if (exp) ocb id = id op | number | expStmt occ$ | Shift |
| $if (exp) ocb id = id op number | expStmt | occ$ | Shift |
| $if (exp) ocb id = id op number | expStmt | occ$ | Reduce: num->number |
| $if (exp) ocb id = id op num expStmt | occ | $ | Shift |
| $if (exp) ocb id = id op num expStmt | occ | $ | Reduce: expStmt -> id op num expStmt |
| $if (exp) ocb id = expStmt | occ | $ | Reduce: assign-> id = expStmt |
| $if (exp) ocb assign occ | $ |  | Shift |
| $if (exp) ocb assign occ | $ |  | Reduce: ifStmt -> if (exp) ocb assign occ |
| $ifStmt | $ |  | Reduce: stmt->ifStmt |
| $stmt | $ |  | Accept |

**Link tham khao:**

<https://timoday.edu.vn/bai-1-nhung-khai-niem-co-ban-ve-ngon-ngu-c/>

DCCT - BG bài giảng chương trình dịch.pdf

<https://en.wikipedia.org/wiki/C_(programming_language)>

https://devfull.me/2019/02/20/nguyen-tac-cua-ngon-ngu-lap-trinh/

http://txnam.net/wp-content/uploads/txnam/Bai%20Giang/Chuong%20Trinh%20Dich%20K53II/Chuong%20trinh%20dich%20K53II%20-%2006.pdf

http://www.thuvientailieu.vn/tai-lieu/chuong-trinh-dich-bai-9-phan-tich-van-pham-bang-thuat-toan-bottom-up-47208/

<http://marvin.cs.uidaho.edu/Teaching/CS445/c-Grammar.pdf>

<https://www.geeksforgeeks.org/shift-reduce-parser-compiler/>