CSE 420

15 March 2025
Saturday

বইয়ের practice problem solve করতে হবে,

Syntax Analyzer · Error Detection Mechanisms

Syntax Analyser have many error recovery methods

1. Panic Mode Recovery

- input string থেকে একটা একটা করে বাদ দিয়ে মিলানোর try করবে,

id * id

- When an error is detected, the parser discards the input symbols until it finds a synchorizing token

- This method is simple and ensures quick recovery, preventing infinite loops.

- Drawback: it may skip a large portion of the code, losing multiple valid statements

2. Phrase Level Recovery:

- When an error is detected, the parser attempts to replace a small portion of the input with something that allows parsing to continue.

- example: ~~missi~~ inserting a missing semicolon or bracket.

- Advantage: Only minor modifications are needed.

- Drawback: Might introduce incorrect assumptions, leading to further errors

3. Error Productions

- The grammar is augmented with productions that describes common errors. When these patterns are recognized, specific error messages can be shown.

- example: if ~~grammar~~ programmers often forget a closing bracket, a rule like E → (E|E) could help catch and handle such errors.

- Advantage: Allows detection of specific errors at the parsing stage.
- Drawback: Adding too much error rules makes the grammar ~~rules~~ complex

- last এ ; না দেওয়া if a==b না দিয়ে দিলে If a=b actual rule এ match না করলে ঐ rule এ match করে দিবে

4. Global Corrections.

- The parser analyzes the entire code and makes the minimum number of changes to make it syntactically correct.

- example: if an opening brace { is missing, then the parser might suggest adding one where it seems most appropriate

- Advantage: Produces a more meaningful error message.

~~- Drawback: Requires complex~~

- Drawback: Computationally expensive and not practical for all compilers

- কিছু instructions থাকে.
  - Structure এর সাথে match রেখে আবার code modify করার try করে, যতটুকু সম্ভব authentic রাখে

method
↳ Under a class

function
↳ not under a class

Symbol Table

- Symbol table compiler এর সব phase এর সাথে connected থাকে,
- Symbol table এ identifier এর info রাখি।

While → While INT → int

▨ Why ID এর info ই symbol table এ রাখছি ; অন্যান্য lexeme এর token info কেন রাখছি না,

⇒ - as memory space waste হবে সবকিছুর info store করতে গেলে.

- Searching time বেশি লাগবে।

Lex Syntax Semantic	Symbol Table store & access	Semantic intermediate code optimizer code generator	Access.

Semantic Action

↳ Grammar rule match করলে যে action perform হবে.,

array নেওয়া যাবে না [because কোন index এ আছে তা জানা যায় না.] link list, stack, queue use করা যাবে না symbol table এ,

hashtable এবং forward chaining use হয়.

↳ Searching এর জন্য best option.

☑ Why we store only ID related info in the symbol table

☑ Why these are called scope table.

☑ Why symbol table important.

☑ Why it is preffered to use hashtable in symbol table instead of other data structures

☑ কোন scenario তে কোন ধরনের symbol table implement করা ভালো, pros and cons of both.

☑ কোন variable কোন scope এ আছে তা কিভাবে বুঝবো?

☑ Difference between Static scoping and Dynamic scoping.

Symbol Table আমরা 2 ভাবে implement করতে পারি।

⇒ - list of hashtables (অনেকগুলো hashtable list আকারে থাকে)

- Hashtable of lists. (একটা hashtable multiple list)

List of Hashtables:

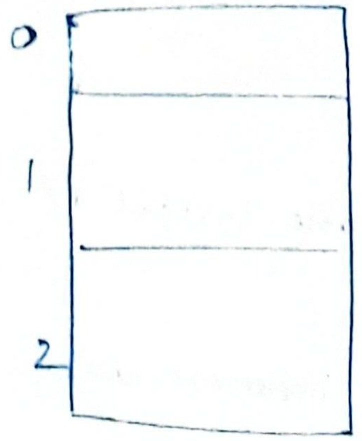
ascii value sum করে bucket count দিয়ে mod করে hash function.
exam এ custom hashfunction থাকতে পারে.

```
1.  int ab
2.  void main (int b) {        function scope ও local variable
3.          int c
4.          if (b == c) {
5.                  float d
6.  semantic error (type mismatch)  d = "0"
7.          }
8.          int bd;
9.  }
10. int a
```

if scope

function scope

Global Scope.



Searching হবে child থেকে parent এ হয়

linked list এ key: value pair এ থাকে।

- প্রত্যেকটা hashtable এর জন্য এক একটা scope এর জন্য,
- Per scope hashtable হবে.
- প্রত্যেকটা hashtable কে তাদের scope এর উপর base করে বানানো হয় বিধায় তাদেরকে scope table বলি.

hashfunction ( )

↳ আমাকে index generate করে দেয়. ওই number index এ [illegible] করবে.

index = hashfunction.

ab = 0

ab এর জন্য index = 0 generate হবে।

ab already exist করে কিনা search করবে.

defined variables.

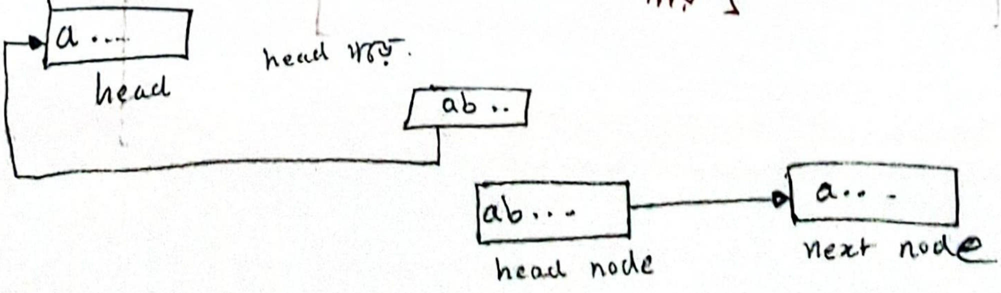
- bucket_count = 3
  ↳ length of the hashtable.
- level = 0 initialize করবে.
  ↳ scope track এর জন্য use হয়.
  ↳ কোনটা কোন scope denote করবে

hashtable এ collision হলে, solve করার way.

↳ linear probing (next available index)

↳ double hashing (hash করে index generate করা)
(length 3 so কাজ করবে না।)

↳ forward chaining (efficient way) [Link list দিয়ে add করতে পারি]



List of Hashtables।

⇒ অনেকগুলো hashtable list এর মতো থাকবে।

⇒ 4 টা action হবে,

i) Scope entry

ii) Process. a Declaration

iii) Process a Use

iv) Scope exit.

same level এ same নামের 2 variable থাকতে পারবে না,

Actions।

1. Scope entry:

⇒ level ++

⇒ create a hashtable

2. Process a Declaration।

⇒ কখন scope table এ data insert করছি.

⇒ same scope এ same নামের 2 variable থাকতে পারবে না.

⇒ if identifier already exists in the current scope table then multiple declare variable.

⇒ if not put the information in the symbol table using hash function.

3. Process a Use: # operation perform হওয়ার সময়,

⇒ Searching

⇒ if It is found

⇒ if not found : Undeclared variable

⇒ look up the identifier In the current scope table

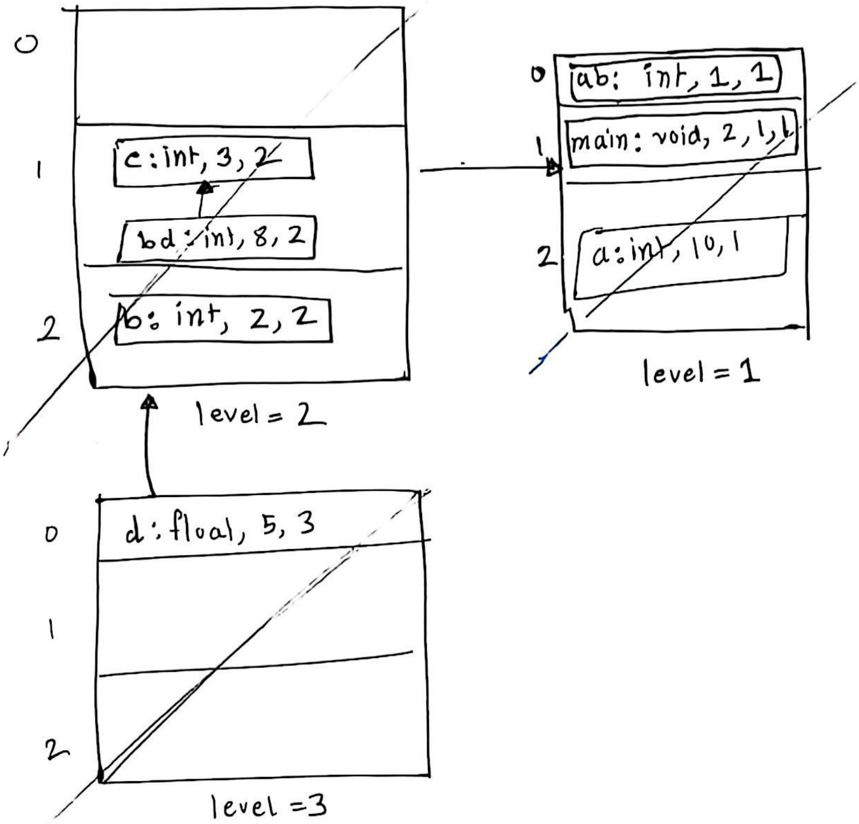
if it's not there : go to the parent scope --- finally

not found

4. Scope exit

⇒ Delete the hashtable

⇒ level--



level = ~~0 1 2 3 2 1~~ 0

bucket_count = 3

সবসময় head এ insert করব।

ab = 0
main = 1
b = 2
c = 1
d = 0
bd = 1
a = 2

arrow থাকবে child থেকে parent দিকে

key তে actual lexeme থাকবে; value তে কি কি info store করতে চাচ্ছি।

main function টা global scope এ define করা so তার info level 1 এ থাকে।

Hashtable: of lists.

Action হবে 4 টাই-

1. Scope entry:

⇒ level ++

2. Process a Declaration,

⇒ Search whether the identifier already exists in the symbol table with the same level number

if yes: multiple declare variable

if not: declare variable

3. Process a use:

⇒ check if the identifier, if found match the levels.

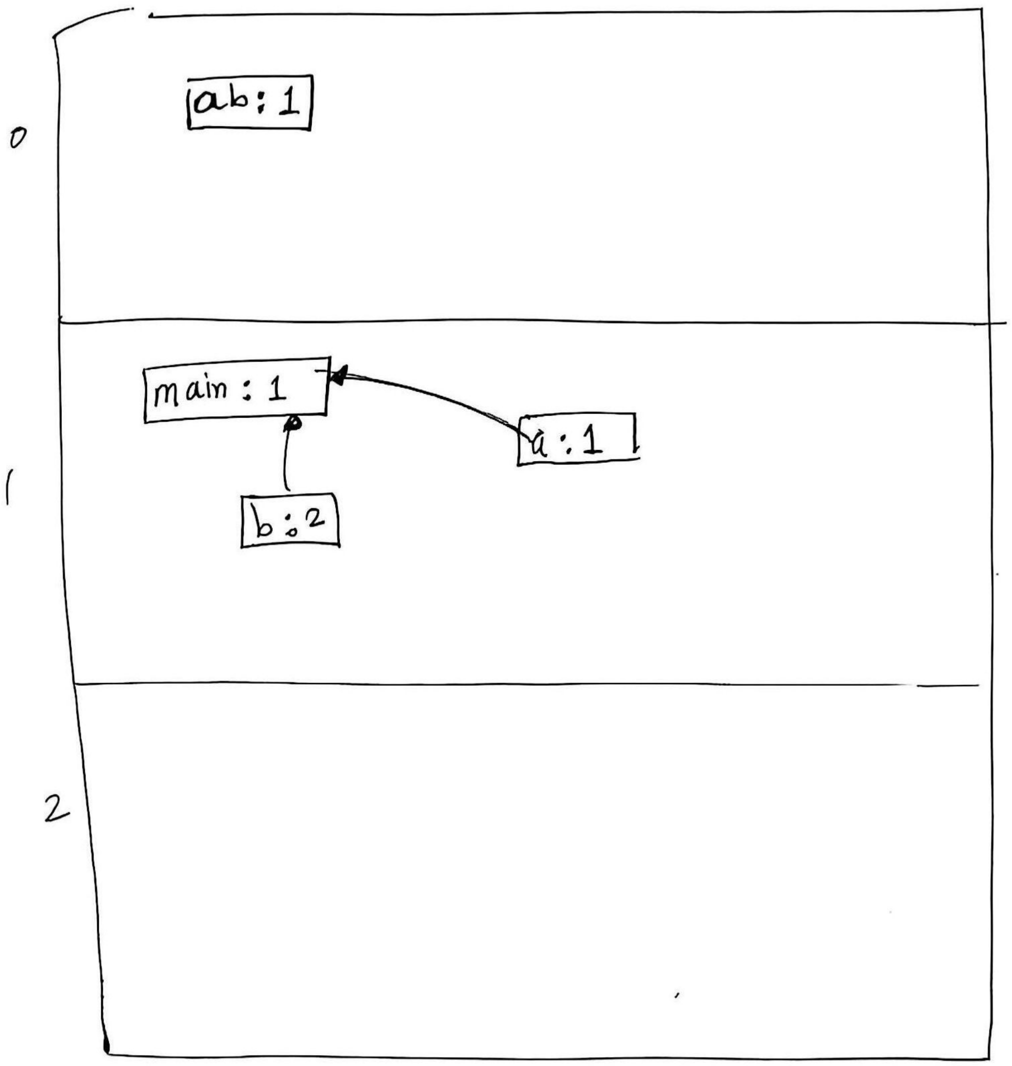
⇒ if not declared ⇒ undeclared variable

-1 করে করে search

4. Scope exit:

⇒ level --

⇒ node delete (অমুক level- এর অন্য যত node ছিলো সব destroy।)



level = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~2~~ 1 0

bucket_ count = 3

ab = 0

main = 1

b = 1

c = 2

d = 2

bd = 2

hashtable of lists এ linktable হিসেবেই data রাখতে হবে

list of hashtable এ যদি এমন ভালো hash function বানাই যেখানে collision হওয়ার chances 0 তখন linked list হিসেবে data রাখা not mandatory.

level এর base করে different different table নিলা।