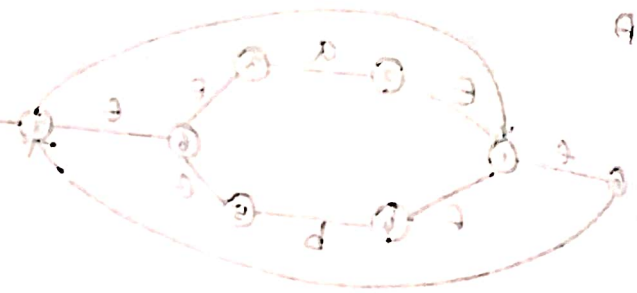
Example for phases of compiler

Source program:

```
int main() {
    int a=5, b=10, c;
    c = a+b;
    return c;
}
```



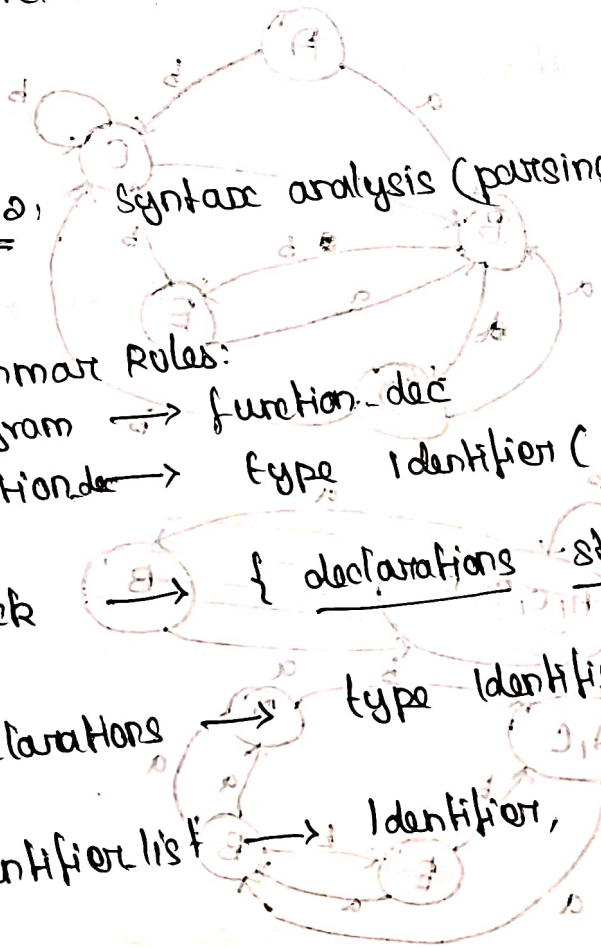
phase 1: lexical analysis i/p: char stream, o/p: tokens

<DT, int> <main> <(> <)> <{>

<DT, int> <ID, 1> <=> <5>, <ID, 2> <=> <10> <ID, 3> <;>

<ID, 3> <=> <ID, 1> <+> <ID, 2> <;>

<return> <ID, 3> <;> <}>



phase 2: Syntax analysis (parsing) i/p: tokens, o/p: AST (or) PT

Grammar Rules:

program → function_dec

functiondec → type identifier ( ) block

block → { declarations stmts }

declarations → type identifier_list ;

identifier_list → identifier, identifier_list | identifier

statements ⟶ statement statements / ε

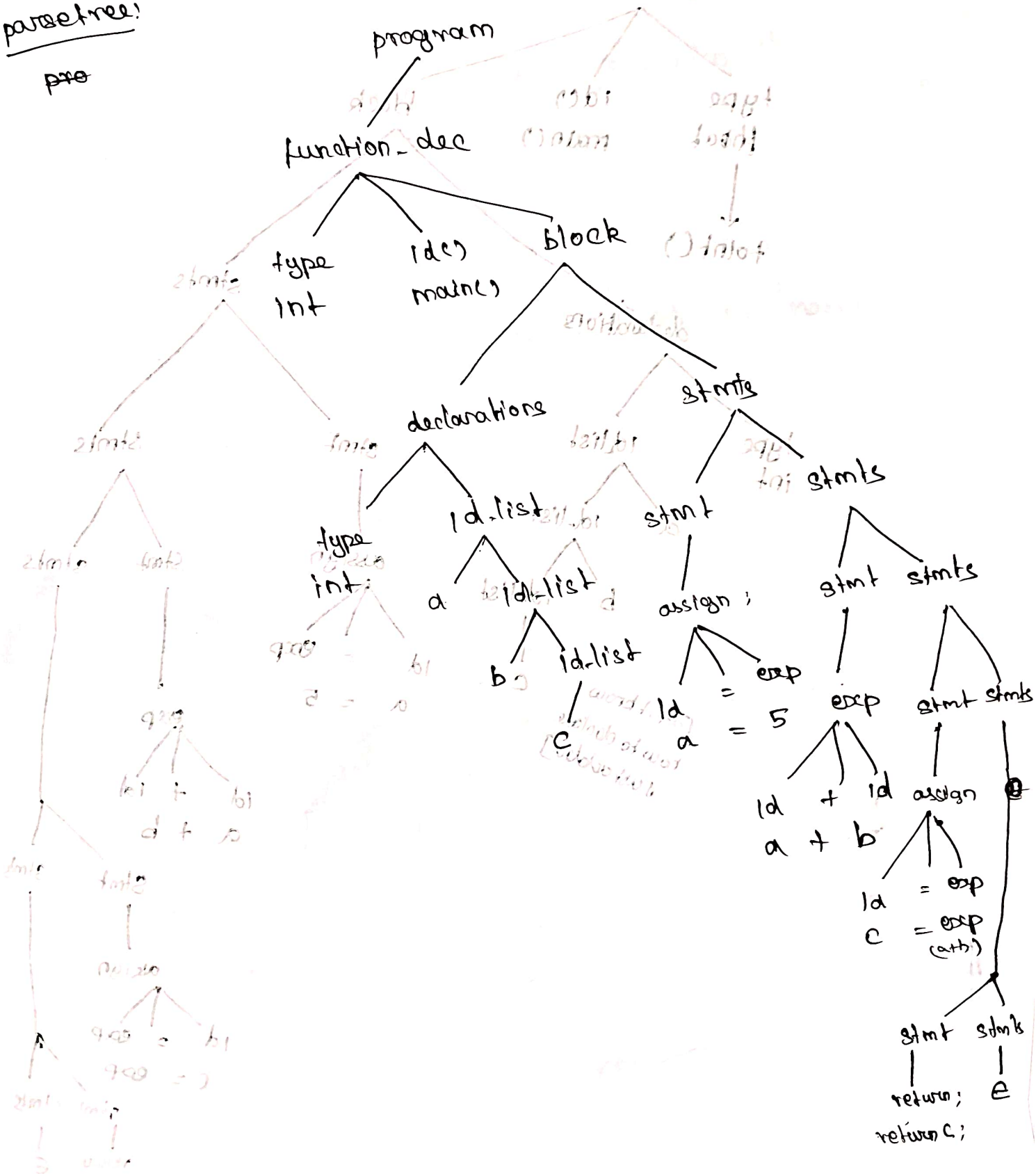
statement ⟶ assignment ; / return ;

assignment ⟶ identifier := expression

expression ⟶ identifier + identifier / literal

parsetree:

pre

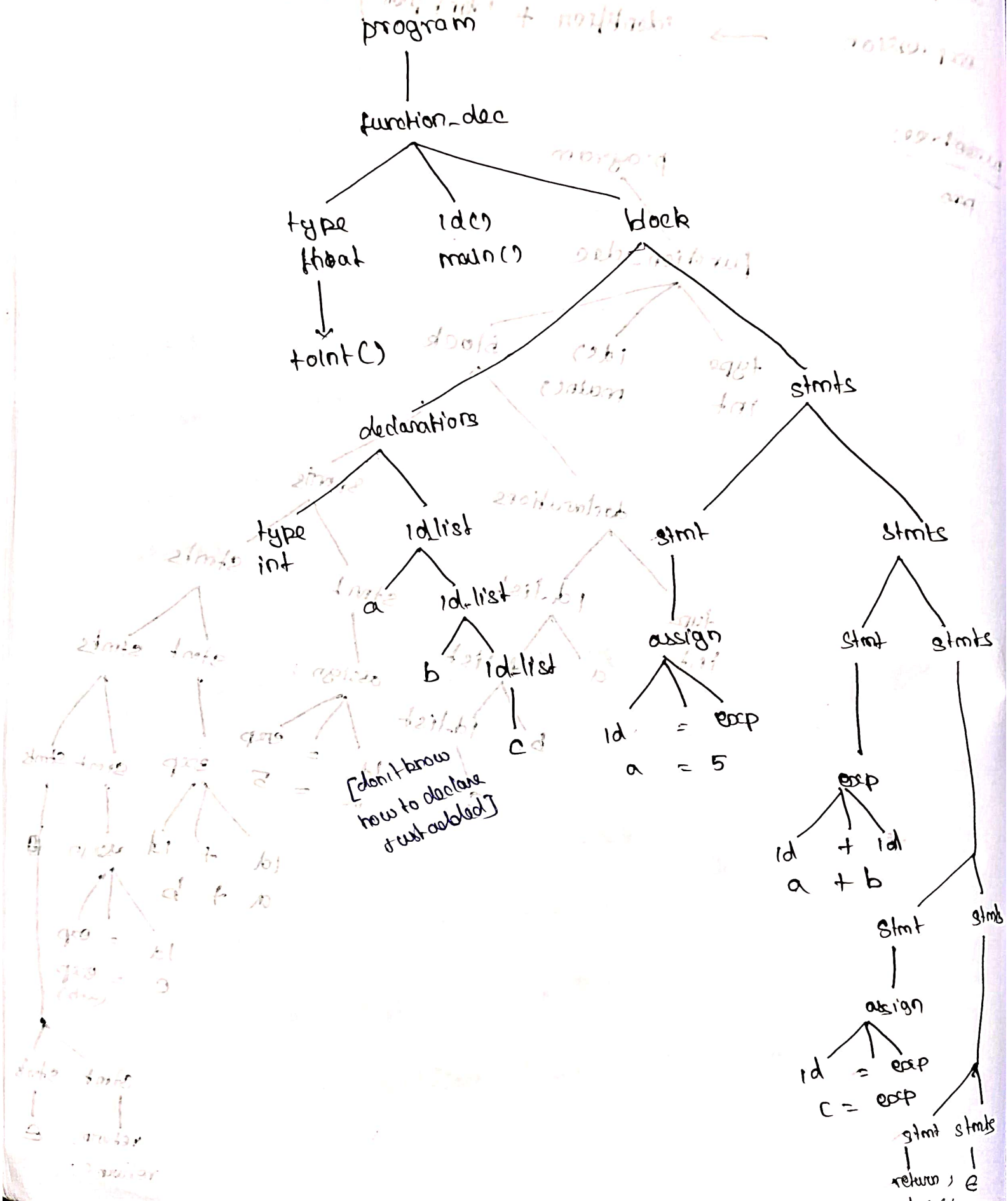


phase 3: semantic analyzer i/p : ST o/p : ST

NT : type consis
var dec
Ret type match

Minor change: float main() {...}

c is not declared but used.



phase 4 : Intermediate code generator I/p: ST, O/p: Intr, reps.

3 address code

```
t1 = 5
t2 = 10
t3 = t1 + t2
t4 = t3
return t4
```

phase 5 : Machine Independent code optimization I/p: Intr rep, O/p: (optin) Intrrep

```
t3 = 5 + 10
return t3
```

phase 6 : code generator I/p: Intr rep (opt) O/p: Machine code

```
MOV R1, 5
MOV R2, 10
ADD R1, R2
MOV [c], R1
ret
```

phase 7 : Machine dependent code opt I/p: MC O/p: (opt) MC

```
MOV R1, 5
MOV R2, 10
ADD R1, R2
MOV [c], R1
ret
```