Student :- Ali Abdul Jabbar   LIF22BSCS 0501

CC   1/1/26

Assignment -02

Task - 01 :-

→ Purpose :- Hogwarts++ is a magical - themed programming language designed to make it more fun and engaging for beginner

→ Style of Syntax :- Follows a procedural and block - structured syntax
- { } → defining blocks of code
- $ → used as statement terminator
- ( ) → used in conditional loops.
- Keywords → are lowercase spell-like words.

→ Reason for choosing Keywords :-
- → Creativity and fun.
- → Maintain similarity with C/C++
- → Help begginners relate programming concepts with familiar spell names.

Keywords :-

1. numspell - Integer
2. textspell - String
3. house - class
4. ifcharm - if
5. listen - input

Operators :-

1. #= (Equal to) 2. #+ (Increment) 3. #! (Not equal)

Punctuators :-

1. $ - Statement Terminator
2. { } - Block of Code

Task-2: Non-Terminals starts with Capitals

Program → ~~Function~~ ~~/ Function~~ FunctionList
Function → ReturnType identifier ( ) { StatementList }
/ ReturnType identifier (ParameterList)
{ Statement List }
FunctionList → function / Function FunctionList
ReturnType → numspell / textspell / floatspell
/ truth charm / void charm
ParameterList → Parameter / Parameter , Parameter List.
Parameter → DataType identifier.
DataType → numspell / textspell / floatspell / truthcharm
StatementList → Statement StatementList / λ
Statement → Declaration / Assignment /
Conditional Stmt / Loop Stmt /
Input Stmt / Output Stmt / ReturnStmt
/ Break Stmt / Continue Stmt
Declaration → DataType IdentifierList $
IdentifierList → Identifier / Identifier , IdentifierList
/ Identifier = Expression
/ Identifier = Expression , IdentifierList
Assigment → Identifier = Expression $ /
Identifier #= Expression /
Identifier #+ $ /
Identifier #- $ ~~/ Identifier~~
Conditional Stmt → if charm (Expression)
{ Statement List } / if charm (Expression)
{ StatementList } else charm
{ Statement List }
Loop Stmt → loopcharm (Expression) { StatementList }
/ spellcycle (Assingment Expression $ Assigment
{ StatementList }

```
InputStmt-> listen (Identifier List) $
Output Stmt-> reveal (Expression) $
            / reveal (ExpressionList) $
Expression List -> Expression / Expression, ExpressionList
Return Stmt-> return charm Expression $ /
              return charm $
Break Stmt-> break curse $
Continue Stmt -> skip curse $
Expresson -> Logical Or Expr
Logical Or Expr-> Logical And Expr /
   Logical Or Expr || Logical And Expr
Logical And Expr-> Logical And Expr && Equality Expr
                 / Equality Expr
Equality Expr-> Relational Expr / Equality Expr
               #= Relational Expr / Equality Expr
               #! Relational Expr
Relational Expr-> Additive Expr / Relational Expr (RE) < Additive Expr (AE)

                 / RE > AE / RE <= AE / RE >= AE
Additive Expr-> Multiplicative Expr /
                Additive Expr + Multiplicative Expr /
                AE - ME
Multiplicative Expr-> Unary Expr / ME * UE /
                      ME / UE / ME % UE
Unary Expr-> Primary Expr / ! Unary Expr / - UE / +UE
Primary Expr-> Identifier / String_Literal /
             Char_Literal / (Expression) / Number_int
             / number_float / number_exp
```

Task-03:-

Program → Function
Function → voidcharm beginmagic() { StmtList }
StmtList → Stmt StmtList / λ
Declaration → numspell id $
Declaration → numspell id = Expr $
Assignment → id = Expr $
CondStmt → if charm (Expr) { StmtList }
else charm { StmtList]
LoopStmt → loopcharm (Expr) { StmtList }
Output → reveal (Expr) $
Expr → Expr + Expr

Task-04:-

① → First(Statement (Stmt))
{ numspell, textspell, floatspell, truthcharm, identifier, ifcharm, loopcharm, spellcycle, listen, reveal, returncharm, breakcurse, skipcurse }

→ Follow
{ first of Stmt }

② → First (Expression (Expr))
{ identifier, Number-int, number-float, Number-exp, string-literal, char-literal, (, !, -, + }

Follow
{ $,), +, -, /, %, <, >, <=, >=, #=, #!, [illegible], ||, &&, , , {}

Task-05 :-

Ambiguous -? → Yes

Example :-

```
    if charm (a>0)
        if charm (b>0)
            reveal ("Both positive")$
    else charm
            reveal (" Not positive")$
```

This else charm can belong to either outer / inner if charm, creating two possible parse trees

Resolution :- Precedence Rules & Parsing Strategy

→ After parsing the first block {}, immediately check for else charm

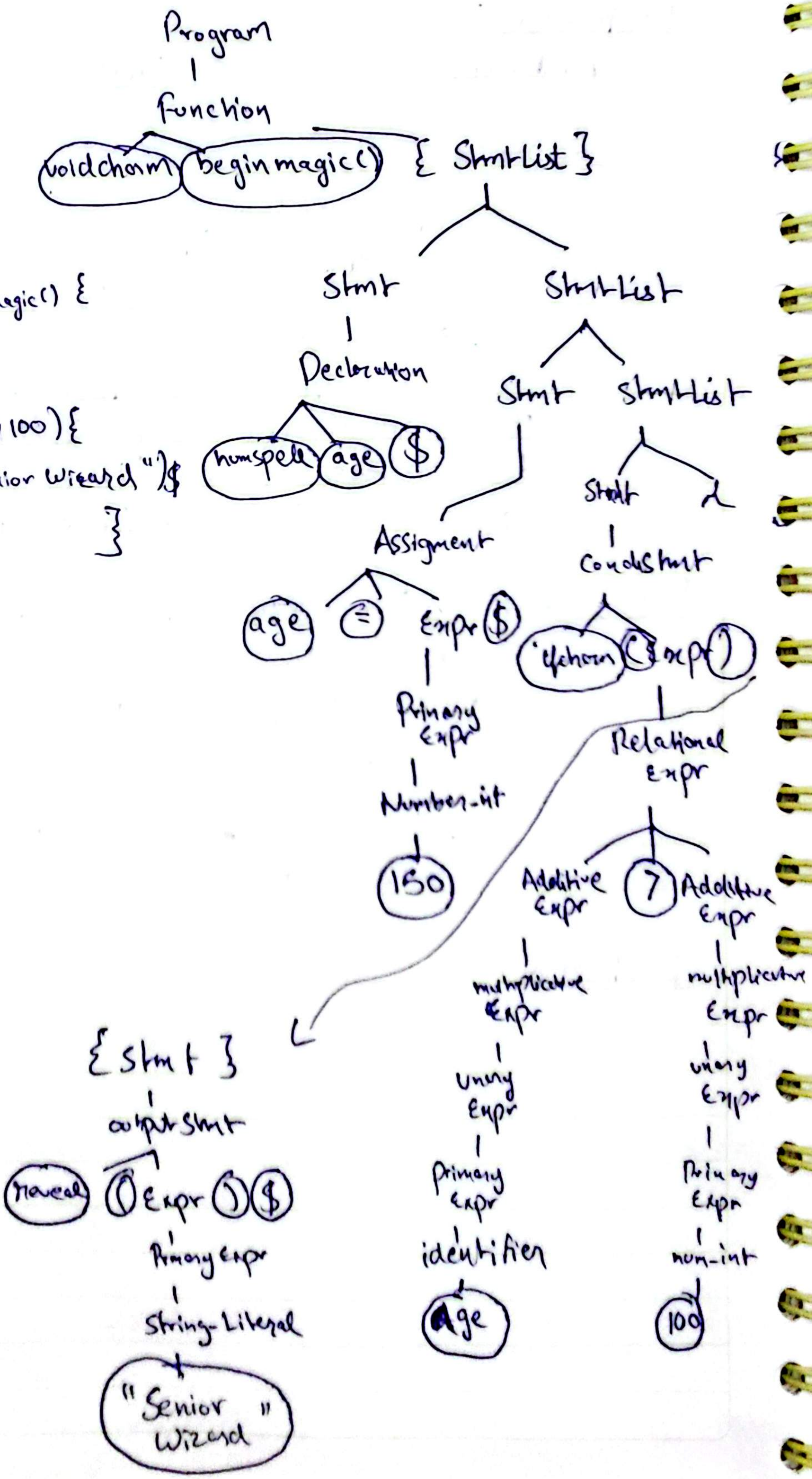
→ if found, consume it as of current if charm.

→ Binding else charm to nearest unmatched if charm.

Task-06 :-

→

```
void charm beginmagic() {
  numspell age $
  age = 150 $
if charm (age > 100) {
  reveal ("Senior Wizard")$
                         }
```



Task - 07 :-

①

```
1  void charm beginmagic () {
2        numspell power
3        power = 9000 $
4     }
```

Error Analysis:-

-> Line 2

-> • Declaration -> numspell identifier $

-> ($) Expected token

②

```
1  void charm beginmagic() {
2     numspell score = 100 $
3     if charm score > 50 {
4          reveal ("Pass") $
5          }
6     }
```

Error Analysis:-

-> Line 3

-> CondStmt -> if charm (Expr) { StmtList }

-> '(' expected after if charm

')' expected after Expr.