5. (5 points) use Denotational semantics to define your selection statement

// <IF\_STMT> --> `if` `(` <BOOL\_EXPR> `)` <BLOCK>  [ `else`  <BLOCK> ]

ifThenElseStatment:

if M\_expr(Expression,s) == Error

return Error

if M\_expr(Expression,s) == False

return s

else if M\_expr(Expression,s) == True

if M\_stmt(Statement,s) == Error

return Error

return s = M\_stmt(Statement,s)

6. (5 points) use Denotational semantics to define your loop statement

// <WHILE\_LOOP> --> `while` `(` <BOOL\_EXPR> `)` <BLOCK>

While(Expression) Statement

If M\_expr(Expression,s)==Error

Return Error

if M\_expr(Expression,s) == False

return s

If M\_expr(Expression,s)==True

M\_stmt(Statement,s)==Error

Return Error

Return M\_While(while(Expression) Statement, M\_stmt(Statement,s))

7. (10 points) use Denotational semantics to define your Expr statement

<EXPR> --> <TERM> {(`+`|`-`) <TERM>}

<TERM> --> <FACT> {(`\*`|`/`|`%`) <FACT>}

<FACT> --> ID | INT\_LIT | FLOAT\_LIT | `(` <EXPR> `)`

/\* <EXPR> --> <TERM> {(`+`|`-`) <TERM>}\*/

M\_expr(<TERM1> '+' <TERM2>}, s) ==>

if M\_expr(<TERM1>, s) == Error

return Error

if M\_expr(<TERM2>, s) == Error

return Error

else

return <TERM1> + <TERM2>

M\_expr(<TERM2> '-' <TERM1>}, s) ==>

if M\_expr(<TERM1>, s) == Error

return Error

if M\_expr(<TERM2>, s) == Error

return Error

else

return <TERM1> - <TERM2>

M\_expr(<TERM>, s) ==>

if M\_expr(<TERM>, s) == Error

return Error

else

return <TERM>

/\* <TERM> --> <FACT> {(`\*`|`/`|`%`) <FACT>}\*/

M\_term(<FACT1> '\*' <FACT2>}, s) ==>

if M\_term(<FACT1>, s) == Error

return Error

f M\_term(<FACT2>, s) == Error

return Error

else

return <FACT1> \* <FACT2>

M\_term(<FACT1> '/' <FACT2>}, s) ==>

if M\_expr(<FACT1>, s) == Error

return Error

f M\_term(<FACT2>, s) == Error

return Error

else

return <FACT1> / <FACT2>

M\_term(<FACT1> '%' <FACT2>}, s) ==>

if M\_expr(<FACT1>, s) == Error

return Error

f M\_term(<FACT2>, s) == Error

return Error

else

return <FACT1> & <FACT2>

/\* <FACT> --> ID | INT\_LIT | FLOAT\_LIT | `(` <EXPR> `)`\*/

M\_fact(ID,s) ==>

if M\_id(id,s) == Error

return Error

else

return s

M\_fact(INT\_LIT, s) ==>

if M\_int\_lit(int\_lit, s) == Error

return Error

else

return s

M\_fact(FLOAT\_LIT) ==>

if M\_float\_lit(int\_lit, s) == Error

return Error

else

return s

M\_fact('('<EXPR'>')' ) ==>

if M\_expr(<expr>, s) == Error

return M\_fact(<EXPR>,s)

8. (5 points) use Denotational semantics to redefine your Expr statement so it can return a Boolean solution

/\* <BOOL\_EXPR> --> <BTERM> {(`>`|`<`|`>=`|`<=`) <BTERM>}\*/

M\_boolexpr(<BTERM1> '>'<BTERM2, s>) ==>

if M\_bterm(<BTERM1>, s) == Error

return Error

if M\_bterm(<BTERM2>, s) == Error

return Error

else

return m\_boolexpr(<BTERM1> '>'<BTERM2>, s)

M\_boolexpr(<BTERM1> '<'<BTERM2>, s) ==>

if M\_bterm(<BTERM1>, s) == Error

return Error

if M\_bterm(<BTERM2>, s) == Error

return Error

else

return m\_boolexpr(<BTERM1> '<'<BTERM2>, s)

M\_boolexpr(<BTERM1> '<='<BTERM2>, s) ==>

if M\_bterm(<BTERM1>, s) == Error

return Error

if M\_bterm(<BTERM2>, s) == Error

return Error

else

return m\_boolexpr(<BTERM1> '<='<BTERM2>, s)

M\_boolexpr(<BTERM1> '>='<BTERM2>, s) ==>

if M\_bterm(<BTERM1>, s) == Error

return Error

if M\_bterm(<BTERM2>, s) == Error

return Error

else

return m\_boolexpr(<BTERM1> '>='<BTERM2>, s)

M\_boolexpr(<BTERM>, s) ==>

if M\_bterm(<BTERM>, s) == Error

return Error

else

return m\_boolexpr(<BTERM>, s)

9. (10 points) Define the attribute grammar for your assignment statement, make sure it follows the following rules

a. String + String does concatenation (hello +world =helloworld)

ASSIGN 🡪 VAR = EXPR

EXPR 🡪 VAR

EXPR 🡨 VAR

EXPR1 🡪 VAR \* EXPR2

EXPR1.VALUE 🡨 VARVARVAR

EXPR1.TYPE 🡨 IF VAR1 is String and VAR2 is String return String.

ASSIGN 🡪 VAR = EXPR

EXPR 🡪 VAR

EXPR 🡨 VAR

EXPR1 🡪 VAR + EXPR2

EXPR1.VALUE 🡨 VAR + EXPR2

IF VAR is String and EXPR2 is Int return String

ASSIGN 🡪 VAR = EXPR

EXPR 🡪 VAR

EXPR 🡨 VAR

EXPR1 🡪 VAR \* EXPR2

EXPR1.VALUE 🡨 VARVARVAR

IF VAR is INT and EXPR2is BOOLEAN return BOOLEAN.

IF VAR is Boolean and EXPR2is int return Int.

IF VAR is Int and VAR2 is EXPR2return CHAR.

IF VAR is CHAR and EXPR2is INT return INT.

IF VAR is int and EXPR2is float return float.

IF VAR is STRING and VAR2 is EXPR2 return ERROR (Type mismatch).

IF VAR is STRING and EXPR2 is CHAR return ERROR (Type mismatch).

IF VAR is STRING and EXPR2 is BOOLEAN return ERROR (Type mismatch).

IF VAR is STRING and EXPR2 is FLOAT return ERROR (Type mismatch).

IF VAR is INT and EXPR2 is STRING return ERROR (Type mismatch).

IF VAR is CHAR and EXPR2 is STRING return ERROR (Type mismatch).

IF VAR is BOOLEAN and EXPR2 is STRING return ERROR (Type mismatch).

IF VAR is FLOAT and EXPR2 is STRING return ERROR (Type mismatch)

IF VAR IS INT and EXPR2 equals zero return Error (Divid by zero error)

IF VAR IS float and EXPR2 equals zero return Error (Divid by zero error)

IF VAR IS Boolean and EXPR2 equals zero return Error (Divid by zero error)

IF VAR IS INT and EXPR2 equals zero return Error(undefined)

IF VAR IS float and EXPR2 equals zero return Error(undefined)

IF VAR IS Boolean and EXPR2 equals zero return Error(undefined)

EXPR1 🡪 VAR + EXPR2

EXPR1.VALUE 🡨 VAR + EXPR2

EXPR1.TYPE 🡨 IF VAR is String and EXPR2 is String return String.

IF VAR is String and EXPR is Int return String

IF VAR is INT and EXPR2 is BOOLEAN return BOOLEAN.

IF VAR is Boolean and EXPR2 is int return Int.

IF VAR is Int and EXPR2is CHAR return CHAR.

IF VAR is CHAR and EXPR2 is INT return INT.

If VAR is int and EXPR2 is floating return float.

IF VAR is STRING and EXPR2is INT return ERROR (Type mismatch).

IF VAR is STRING and EXPR2is CHAR return ERROR (Type mismatch).

IF VAR is STRING and EXPR2is BOOLEAN return ERROR (Type mismatch).

IF VAR is STRING and EXPR2is FLOAT return ERROR (Type mismatch).

IF VAR is INT and EXPR2 is STRING return ERROR (Type mismatch).

IF VAR is CHAR and EXPR2 is STRING return ERROR (Type mismatch).

IF VAR is BOOLEAN and EXPR2 is STRING return ERROR (Type mismatch).

IF VAR is FLOAT and EXPR2 is STRING return ERROR (Type mismatch)

IF VAR IS INT and EXPR2 equals zero return Error (Divid by zero error)

IF VAR IS float and EXPR2 equals zero return Error (Divid by zero error)

IF VAR IS Boolean and EXPR2 equals zero return Error (Divid by zero error)

IF VAR IS INT and EXPR2 equals zero return Error(undefined)

IF VAR IS float and EXPR2 equals zero return Error(undefined)

IF VAR IS Boolean and EXPR2 equals zero return Error(undefined)

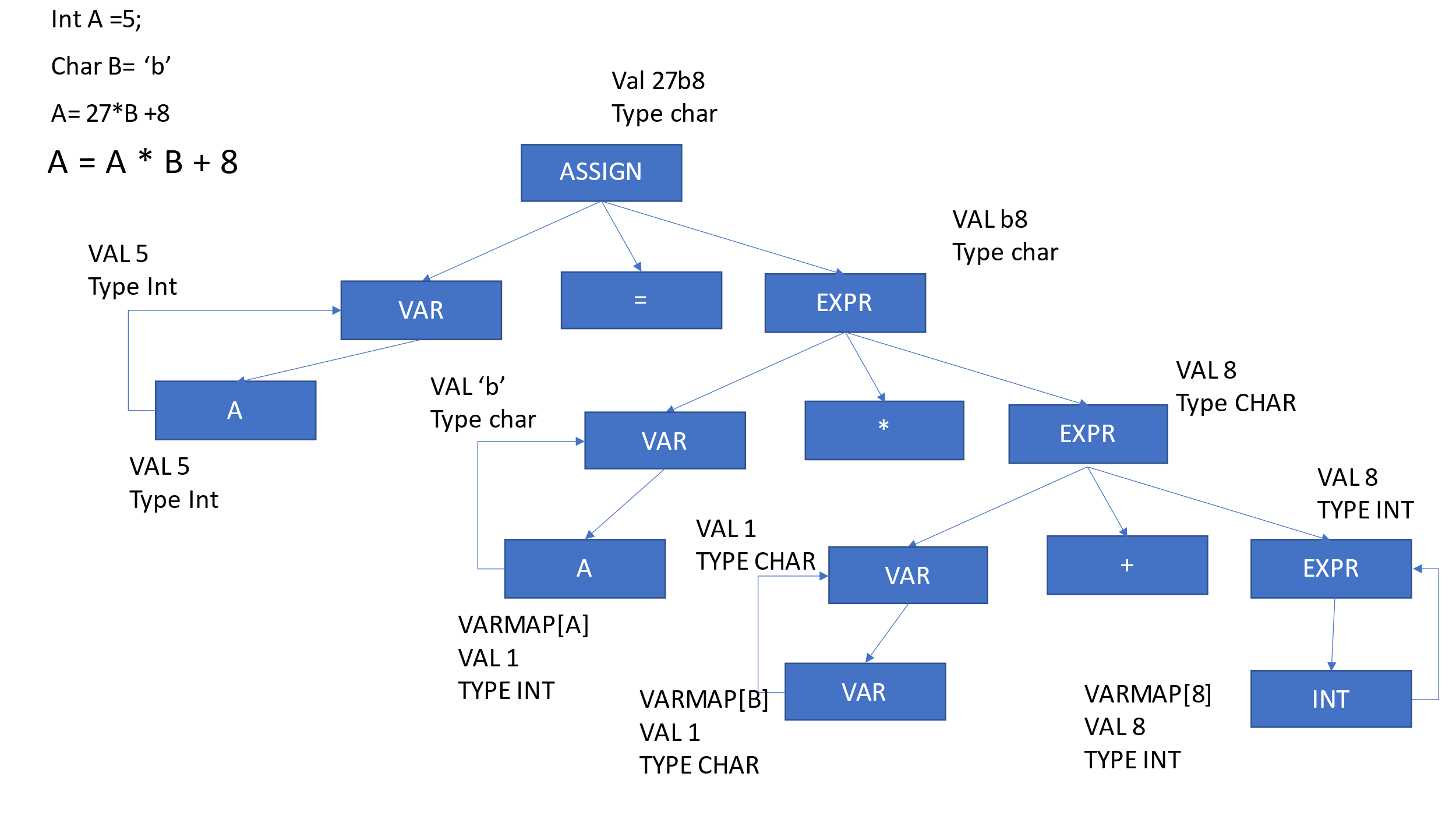
10. (15 point) choose 3 syntactically valid assignment statements with at least 7 tokens to show these rules failing or passing semantic rules.

1. Char to Natural 2.Boolean to Natural Natural to Real

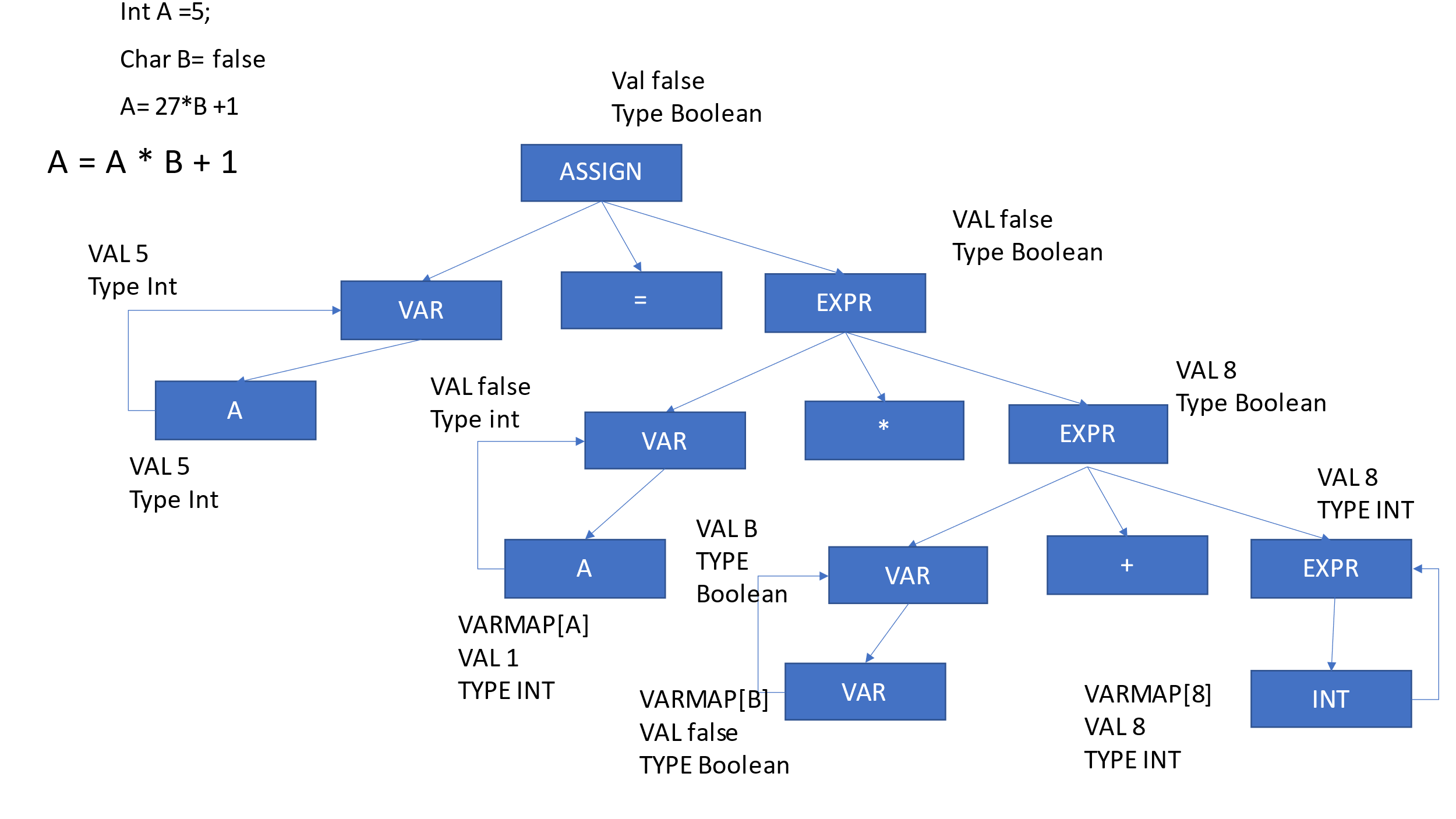
Int A =0 Int A =0 Int A =0

Char B =’b’ Boolean B= false float B=2.99

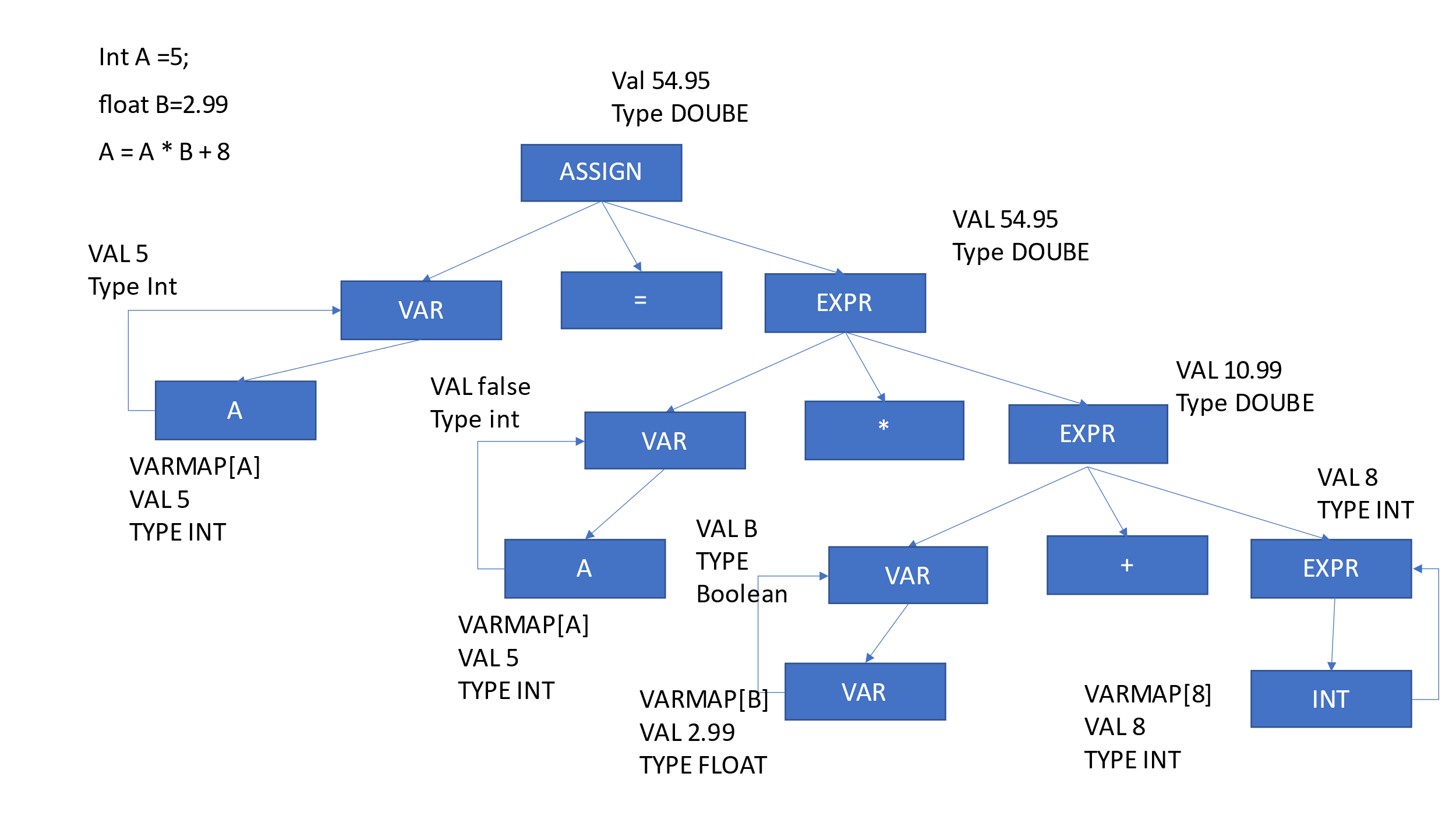
A = A\* B + 1 A = A\* B + 1 A = A\* B + 1











11. ( 10 points ) Axiomatic Semantics ( find the weakest precondition) :

a.

a = 2 \* (b - 1) - 1 {2 \* (b - 1) - 1 > 0 }

{a > 0}

b.

if (x < y) {3x <0 }

x = x + 1

else

x = 3 \* x

{x < 0}

c.

y = a \* 2 \* (b - 1) - 1

if (x < y)

x = y + 1

else

x = 3 \* x weakest precondition{3x<0}

{x < 0}

d.

a = 3 \* (2 \* b + a); { 2 \* 3 \* (2 \* b + a) – 1 > 5 }

b = 2 \* a - 1 {2 \* a - 1 >5}

{b > 5}