Primeira Prova de Compiladores - 2/2010 - 24/09/2010

Aluno:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. (2,0) Identifique quais das tarefas abaixo se refere à análise léxica/sintática (S), à análise contextual (C), às duas (SC) ou deixe em branco se não for realizada por nenhuma das duas fases.

( C ) Verifica se uma variável foi declarada antes de ser utilizada

( C ) Realiza verificações referentes ao escopo das variáveis

( S ) verifica se o número de parâmetros passados em uma chamada a um método está correto

( C ) verifica se o tipo dos parâmetros passados em uma chamada a um método está correto

( C ) Verifica se a avaliação de uma expressão vai gerar um resultado de um determinado tipo

( SC ) Interage com a tabela de símbolos (lê ou escreve)

( S ) É especificada utilizando gramáticas livres de contexto (BNF / EBNF)

( C ) realiza anotações adicionais na parse tree (decora)

( S ) Cria a abstract syntax tree na memória

( ) É responsável pela geração de código

**A Gramática abaixo é usada nas próximas questões:**

Program ::= Statements “;” ***Programa***

Statements ::= Statement ***SimpleStmt***

| Statement “;” Statements ***MultStmt***

Statement ::= "print" Identifier ***PrintStmt***

| Identifier "=" Expression  ***AssignmentStmt***

Expression ::= Number ***IntLiteral***

| Expression "+" Number ***AddExpression***

| Expression "\*" Number ***MulExpression***

Identifier e Number são definidos pelas expressões regulares abaixo:

Identifier ::= [a-ZA-Z]+

Number ::= [0-9]+

1. (2.0) A gramática precisaria sofrer alguma modificação para que você possa fazer essa implementação de um recursive descent parser? Se sim, mostre como e quais produções teriam que ser modificadas.

Sim, pois o recursive descendent parser não pode ter recursos à esquerda, como é o casso de Expression := Expression “+” Number e Expression : = Expression “\*” Number. Uma possível modificação seria:

Expression ::= Number ***IntLiteral***

| Number “+” Expression ***AddExpression***

| Number "\*" Expression ***MulExpression***

1. (3.0) Escreva a implementação dos métodos (ou funções ou procedimentos) que seriam criados para implementar um recursive descent parser para a gramática acima. Em Java ou C ou Haskell.

void Program (){

switch (lookahead) {

case Statements() : Statements (); match (TO\_SEMI\_COL); break;

default : syntax\_error();

}

}

void Statements (){

switch (lookahead) {

case Statement() : Statement (); if (lookahead /= null) match (TO\_SEMI\_COL); if (lookahead /= null) Statements (); break;

default : syntax\_error();

}

}

void Statement (){

switch (lookahead) {

case (TO\_PRINT) : match (TO\_PRINT); match (TO\_IDENTIFIER); break;

case (TO\_IDENTIFIER) : match (TO\_IDENTIFIER); match (TO\_SEMI\_EQUALS); Expression(); break;

default : syntax\_error();

}

}

void Expression (){

switch (lookahead) {

case (TO\_Number) : match (TO\_Number);

if (lookahead /= null){

if (lookahead == “+”) {

match (TO\_PLUS);

Expression ();

break;

}

if (lookahead == “\*”) {

match (TO\_MULT);

Expression ();

break;

}

} else {

break; // lookahead é null e a produção gera apenas o number

}

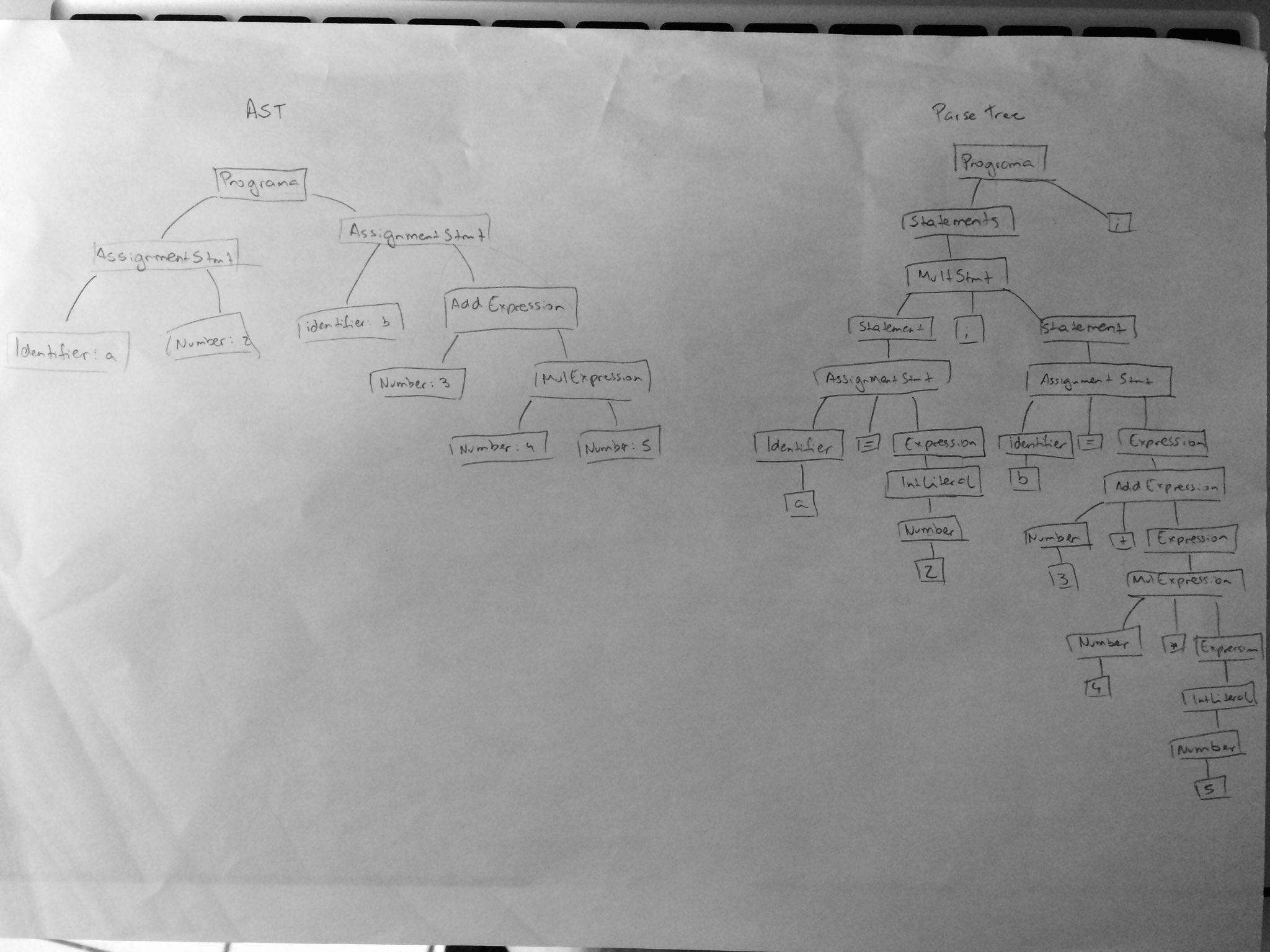
default : syntax\_error();

}

}

1. (3.0) Usando a gramática acima, sem modificações, desenhe uma árvore sintática abstrata e uma parse tree para o programa abaixo.

**a = 2; b = 3 + 4 \* 5;**

****

Primeira Prova de Compiladores - 2/2010 - 24/09/2010.

Aluno:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. (2,0) Identifique quais das tarefas abaixo se refere à análise léxica/sintática (S), à análise contextual (C), às duas (SC) ou deixe em branco se não for realizada por nenhuma das duas fases.

( C ) Verifica se a avaliação de uma expressão vai gerar um resultado de um determinado tipo

( SC ) Interage com a tabela de símbolos (lê ou escreve)

( S ) É especificada utilizando gramáticas livres de contexto (BNF / EBNF)

( C ) realiza anotações adicionais na parse tree (decora)

( S ) Cria a abstract syntax tree na memória

( ) É responsável pela geração de código

( C ) Verifica se uma variável foi declarada antes de ser utilizada

( C ) Realiza verificações referentes ao escopo das variáveis

( S ) verifica se o número de parâmetros passados em uma chamada a um método está correto

( C ) verifica se o tipo dos parâmetros passados em uma chamada a um método está correto

**A Gramática abaixo é usada nas próximas questões:**

Program ::= Statements “;” ***Programa***

Statements ::= Statement ***SimpleStmt***

| Statements “;” Statement ***MultStmt***

Statement ::= "print" Identifier ***PrintStmt***

| Identifier "=" Expression  ***AssignmentStmt***

Expression ::= Number ***IntLiteral***

| Number "+" Expression ***AddExpression***

| Number "\*" Expression ***MulExpression***

Identifier e Number são definidos pelas expressões regulares abaixo:

Identifier ::= [a-ZA-Z]+

Number ::= [0-9]+

1. (2.0) A gramática precisaria sofrer alguma modificação para que você possa fazer essa implementação (de um recursive descent parser? Se sim, mostre como e quais produções teriam que ser modificadas.

Sim, pois o recursive descendent parser não pode ter recursos à esquerda, como é o casso de Statements := Statements “;” Statement. Uma possível modificação seria:

Statements ::= Statement ***SimpleStmt***

| Statement “;” Statements ***MultStmt***

1. (3.0) Escreva a implementação dos métodos (ou funções ou procedimentos) que seriam criados para implementar um recursive descent parser para a gramática acima. Em Java ou C ou Haskell.

void Program (){

switch (lookahead) {

case Statements() : Statements (); match (TO\_SEMI\_COL); break;

default : syntax\_error();

}

}

void Statements (){

switch (lookahead) {

case Statement() : Statement (); if (lookahead /= null) match (TO\_SEMI\_COL); if (lookahead /= null) Statements (); break;

default : syntax\_error();

}

}

void Statement (){

switch (lookahead) {

case (TO\_PRINT) : match (TO\_PRINT); match (TO\_IDENTIFIER); break;

case (TO\_IDENTIFIER) : match (TO\_IDENTIFIER); match (TO\_SEMI\_EQUALS); Expression(); break;

default : syntax\_error();

}

}

void Expression (){

switch (lookahead) {

case (TO\_Number) : match (TO\_Number);

if (lookahead /= null){

if (lookahead == “+”) {

match (TO\_PLUS);

Expression ();

break;

}

if (lookahead == “\*”) {

match (TO\_MULT);

Expression ();

break;

}

} else {

break; // lookahead é null e a produção gera apenas o number

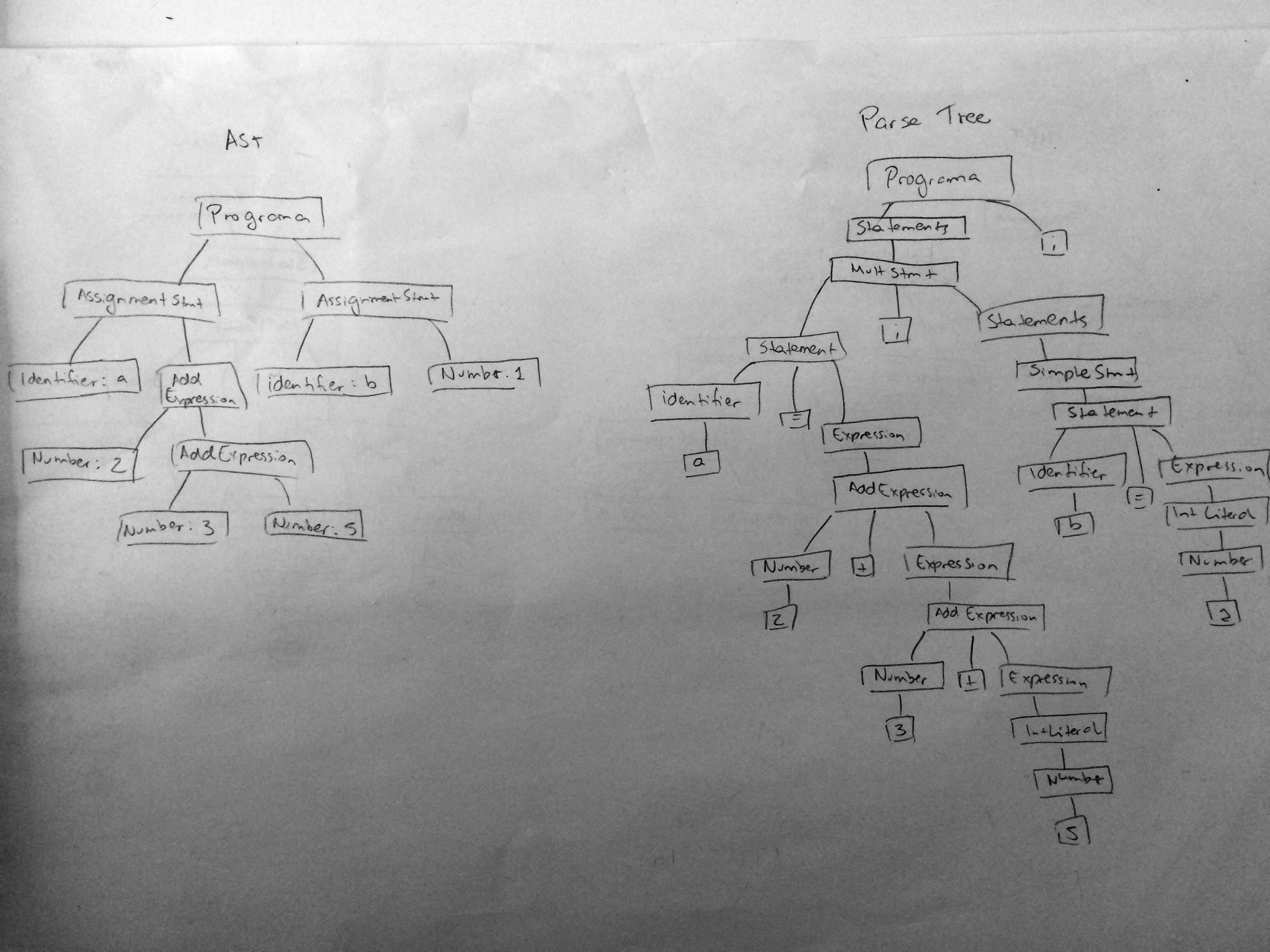
}

default : syntax\_error();

}

}

1. (3.0) Usando a gramática acima, sem modificações, desenhe uma árvore sintática abstrata e uma parse tree para o programa abaixo.

**a = 2 + 3 + 5; b = 1 ;**

Primeira Prova de Compiladores - 2/2010 - 24/09/2010..

Aluno:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. (2,0) Identifique quais das tarefas abaixo se refere à análise léxica/sintática (S), à análise contextual (C), às duas (SC) ou deixe em branco se não for realizada por nenhuma das duas fases.

( S ) verifica se o número de parâmetros passados em uma chamada a um método está correto

( C ) verifica se o tipo dos parâmetros passados em uma chamada a um método está correto

( C ) Verifica se a avaliação de uma expressão vai gerar um resultado de um determinado tipo

( SC ) Interage com a tabela de símbolos (lê ou escreve)

( S ) É especificada utilizando gramáticas livres de contexto (BNF / EBNF)

( C ) realiza anotações adicionais na parse tree (decora)

( S ) Cria a abstract syntax tree na memória

( ) É responsável pela geração de código

( C ) Verifica se uma variável foi declarada antes de ser utilizada

( C ) Realiza verificações referentes ao escopo das variáveis

**A Gramática abaixo é usada nas próximas questões:**

Program ::= Statements “;” ***Programa***

Statements ::= Statement ***SimpleStmt***

| Statement “;” Statements ***MultStmt***

Statement ::= "print" Identifier ***PrintStmt***

| Identifier "=" Expression  ***AssignmentStmt***

Expression ::= Number ***IntLiteral***

| Expression "+" Expression ***AddExpression***

| Expresion "\*" Expression ***MulExpression***

Identifier e Number são definidos pelas expressões regulares abaixo:

Identifier ::= [a-ZA-Z]+

Number ::= [0-9]+

1. (2.0) A gramática precisaria sofrer alguma modificação para que você possa fazer essa implementação (de um recursive descent parser? Se sim, mostre como e quais produções teriam que ser modificadas.

Sim, pois o recursive descendent parser não pode ter recursos à esquerda, como é o casso de Expression := Expression “+” Number e Expression : = Expression “\*” Number. Uma possível modificação seria:

Expression ::= Number ***IntLiteral***

| Number “+” Expression ***AddExpression***

| Number "\*" Expression ***MulExpression***

1. (3.0) Escreva a implementação dos métodos (ou funções ou procedimentos) que seriam criados para implementar um recursive descent parser para a gramática acima. Em Java ou C ou Haskell.

void Program (){

switch (lookahead) {

case Statements() : Statements (); match (TO\_SEMI\_COL); break;

default : syntax\_error();

}

}

void Statements (){

switch (lookahead) {

case Statement() : Statement (); if (lookahead /= null) match (TO\_SEMI\_COL); if (lookahead /= null) Statements (); break;

default : syntax\_error();

}

}

void Statement (){

switch (lookahead) {

case (TO\_PRINT) : match (TO\_PRINT); match (TO\_IDENTIFIER); break;

case (TO\_IDENTIFIER) : match (TO\_IDENTIFIER); match (TO\_SEMI\_EQUALS); Expression(); break;

default : syntax\_error();

}

}

void Expression (){

switch (lookahead) {

case (TO\_Number) : match (TO\_Number);

if (lookahead /= null){

if (lookahead == “+”) {

match (TO\_PLUS);

Expression ();

break;

}

if (lookahead == “\*”) {

match (TO\_MULT);

Expression ();

break;

}

} else {

break; // lookahead é null e a produção gera apenas o number

}

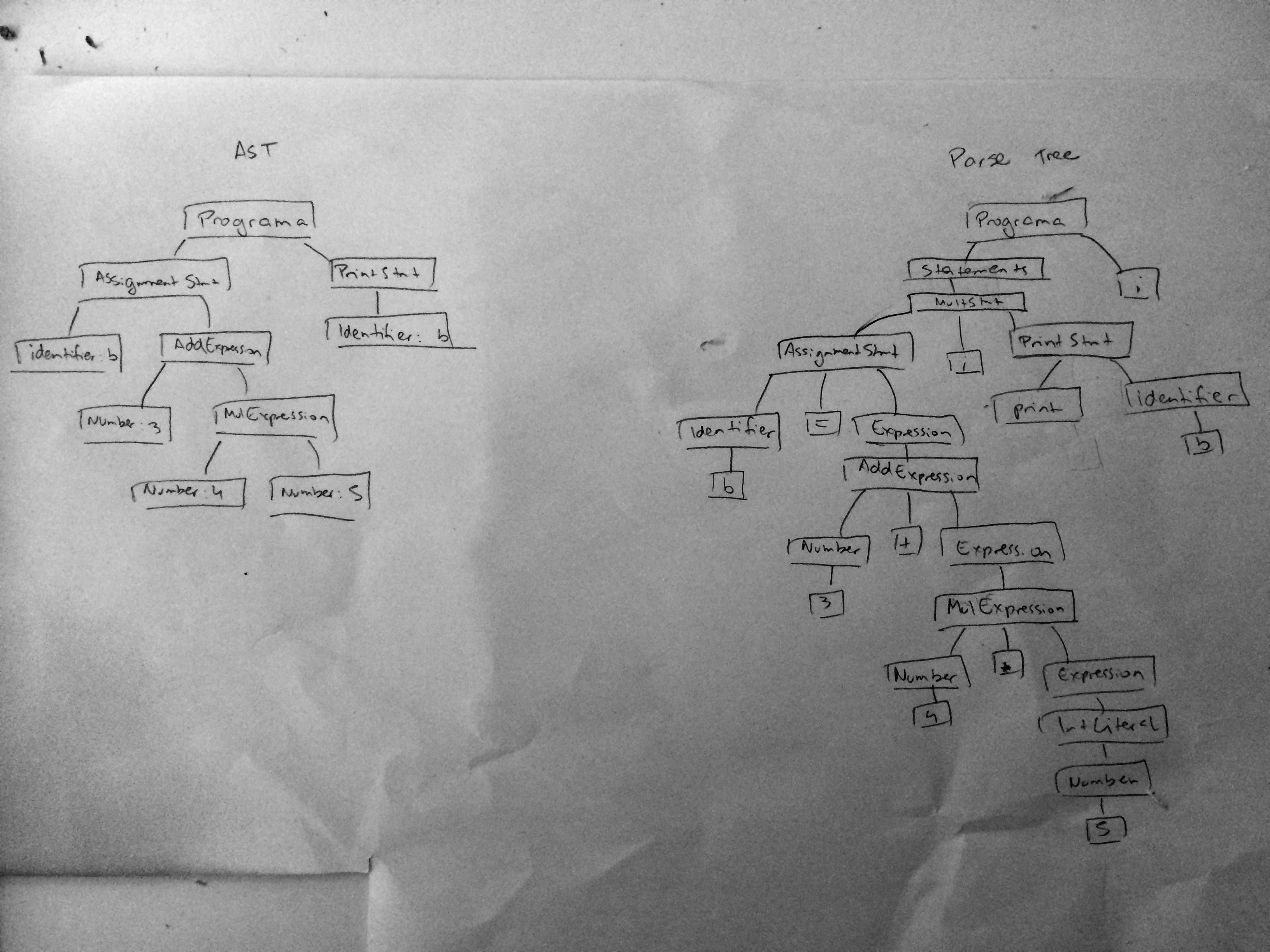
default : syntax\_error();

}

}

1. (3.0) Usando a gramática acima, sem modificações, desenhe uma árvore sintática abstrata e uma parse tree para o programa abaixo.

**b = 3 + 4 \* 5; print b;**

****