**AYNEL & ANDREA**

**ASSIGNMENT 3**

**ORIGINAL GRAMMAR**

Expr -> ID

| Expr + Expr

| -Expr

| Expr++

| Expr--

**11**

Expr -> Expr + Term

| Term

Term -> -Term

| Factor

Factor -> Factor --

| Factor ++

| ID

**12**

Expr -> Term Expr’

Expr’ -> + Expr Expr’

| epsilon

Term -> Factor Term’

Term’ -> -Term Term’

| epsilon

Factor -> ID Factor’

Factor’ -> Factor ++

| Factor --

| ID

| epsilon

**13**

Start -> Expr

Expr -> Term Expr’

Expr’ -> + Expr Expr’

| epsilon

Term -> Factor Term’

Term’ -> - Term Term’

| epsilon

Factor -> ID Factor’

Factor’ -> ++ Factor’

| -- Factor’

| epsilon

**14**

Derive pseudo code for a top-down recursive-descent parser from the start-separated, predictive grammar of Assignment 13.

Start() {

return Expr() && (nextToken() == eof);

}

Expr() {

return Term() && ExprP() ;

}

ExprP() {

Token = nextToken();

Switch (token)

case (add): return Expr() && ExprP();

default: ungetToken(token);

return true;

}

Term() {

return Factor() && TermP();

}

TermP() {

Token = nextToken();

Switch (token)

case (unarymin): return Term() && TermP();

default: ungetToken(token);

return true;

}

Factor() {

Return (nextToken() == ID) && FactorP();

}

FactorP() {

Token = nextToken();

Switch (token)

case (incr): return FactorP();

case (decr): return FactorP();

default: ungetToken(token);

return true;

}