CMPE 260 ASSIGNMENT #1

**Question 1 –**

**a)**

<expr> -> <expr> a | <B\_expr>

<B\_expr> -> <B\_expr> <A> b <A> b | Ɛ

<A> -> <A> a | Ɛ

**b)** To better illustrate my parsing, we can delimit the string after every second occurrence of b

aaabaab abab a

<expr>

**<expr>** **a**

**<B\_expr>** **<A> b <A> b**

**<A> b <A> b** **a** *b* **a** *b*

**<A> a** *b* **<A> a** *b*

**<A> a** *a* *b* **a** *a b*

**a** *a a b a a b*

**Question 2 –**

a) 1 statement -> expression = expression

2 expression -> operation | term

3 addition -> term + addition | term + term

4 division -> term / division | term / term | addition

5 multiplication -> term \* multiplication | term \* term | division

6 operation -> multiplication | unary operation

7 unary operation -> term ++ | term -- | ++ term | -- term

8 term -> ( operation ) | id | operation

9 id -> A | B | C | D | E

**b)** A = --C / (B + ++ E \* D)

|  |  |
| --- | --- |
| **Statement** | 1 |
| **Expression** = **expression** | 2 2 |
| **term** = **operation** | 8 6 |
| **id** = **multiplication** | 9 5 |
| A = **division** | 4 |
| A = **term** / **term** | 8 8 |
| A = **operation** / ( **operation** ) | 6 6 |
| A = **unary operation** / ( **multiplication** ) | 7 5 |
| A = -- **term** / ( **term** \* **term** ) | 8 8 8 |
| A = -- **id** / ( **operation** \* **id** ) | 9 6 9 |
| A = -- C / ( **multiplication** \* D) | 5 |
| A = -- C / ( **division** \* D ) | 4 |
| A = -- C / ( **addition** \* D ) | 3 |
| A = -- C / ( **term** + **term** \* D ) | 8 8 |
| A = -- C / ( **id** + **operation** \* D ) | 9 6 |
| A = -- C / ( B + **unary** **operation** \* D ) | 7 |
| A = -- C / ( B + ++ **term** \* D ) | 8 |
| A = -- C / ( B + ++ **id** \* D ) | 9 |
| A = -- C / ( B + ++ E \* D ) |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| statement | | | | | | | | | | |
| expression | = | Expression | | | | | | | | |
| term |  | Operation | | | | | | | | |
| İd |  | multiplication | | | | | | | | |
| A |  | Division | | | | | | | | |
|  |  | term | | / | term | | | | | |
|  |  | operation | |  | ( operation ) | | | | | |
|  |  | Unary operation | |  | Multiplication | | | | | |
|  |  | -- | Term |  | Term | | | | \* | term |
|  |  |  | İd |  | Operation | | | |  | id |
|  |  |  | C |  | Multiplication | | | |  | D |
|  |  |  |  |  | Division | | | |  |  |
|  |  |  |  |  | Addition | | | |  |  |
|  |  |  |  |  | term | + | Term | |  |  |
|  |  |  |  |  | İd |  | Operation | |  |  |
|  |  |  |  |  | B |  | Unary operation | |  |  |
|  |  |  |  |  |  |  | ++ | Term |  |  |
|  |  |  |  |  |  |  |  | İd |  |  |
|  |  |  |  |  |  |  |  | E |  |  |

**Question 3 –**

**a)** Suppose our while loop is

while ( var1 relop var2 ) {

list\_of\_statements;

}

Formally, without any modification, it could be expressed as follows :

Beg:

if var1 relop var2 goto Inside\_while;

…

Inside\_while:

list\_of\_statements;

Goto Beg;

However, with a slight modification, I believe the following translation is more straight-forward ( the modification is negating the conditional statement)

Beg :

If var1 relop\_negated var2 goto End\_of\_while;

list\_of\_statements;

goto Beg;

End\_of\_while :

…

**b)** Using my latter translation (where I negate the conditional statement)

c = 1;

Beg:

a = b;

b = b – 1;

if a > 0 goto after\_if

c = c-1;

after\_if :

if c == true goto Beg