CS3513 - Programming Languages
210173T
Gallage D
Homework 1

1) E → E or T
→ E nor T
→ E xor T
→ T

* Here we have left recursion there for it is non-LL(1)

T → F and T
→ F nand T
→ F

* Here we have common prefices there for it is non-LL(1)

2) Let's fix common prefices;

T → FX
X → and T
→ nand T
→

Then fix left recursion;

E → TY
Y → or TY
→ nor TY
→ xor TY
→

Therefore equivalent grammer;

T → FX
X → and T
→ nand T
→

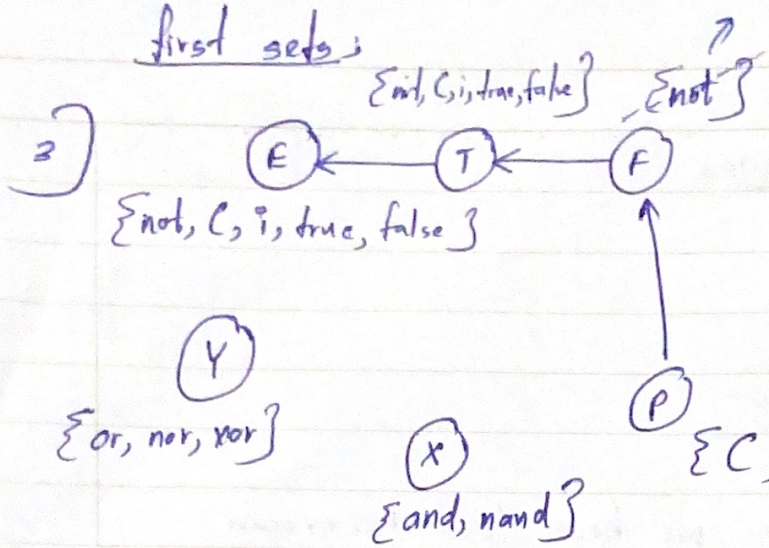
E → TY
Y → or TY
→ nor TY
→ xor TY
→

F → not F
→ P

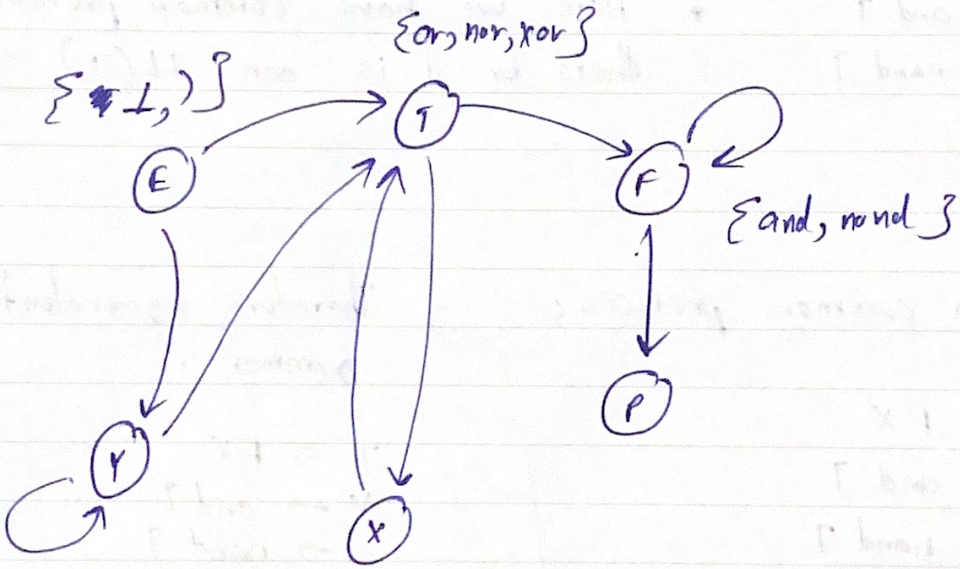
P → (E)
→ i
→ true
→ false

{not, (, i, true, false}

first sets:



follow sets:



follow (E) = {⊥,)}
follow (T) = {or, nor, xor, ⊥,)}
follow (F) = {and, nand, or, nor, xor, ⊥,)}
follow (P) = {and, nand, or, nor, xor, ⊥,)}
follow (X) = {or, nor, xor, ⊥,)}
follow (Y) = {⊥,)}

select sets

E → TY		{ not, (, i, true, false }
Y →	or TY	{ or }
→	nor TY	{ nor }
→	xor TY	{ xor }
→		{ ⊥,) }
T → FX		{ not, (, i, true, false }
X →	and T	{ and }
→	nand T	{ nand }
→		{ or, nor, xor, ⊥,) }
F →	not F	{ not }
→	P	{ (, i, true, false }
P →	(E)	{ (}
→	i	{ i }
→	true	{ true }
→	false.	{ false }

Parse table

	not	(	i	true	false	or	nor	xor	⊥	)	and	nand
E	E→TY	E→TY	E→TY	E→TY	E→TY							
Y						Y→ or TY	Y→ norTY	Y→ xorTY	Y→	Y→		
T	T→FX	T→FX	T→FX	T→FX	T→FX							
X						X→	X→	X→	X→	X→	X→ andT	X→ nandT
F	F→notF	F→P	F→P	F→P	F→P							
P		P→(E)	P→i	P→true	P→false							

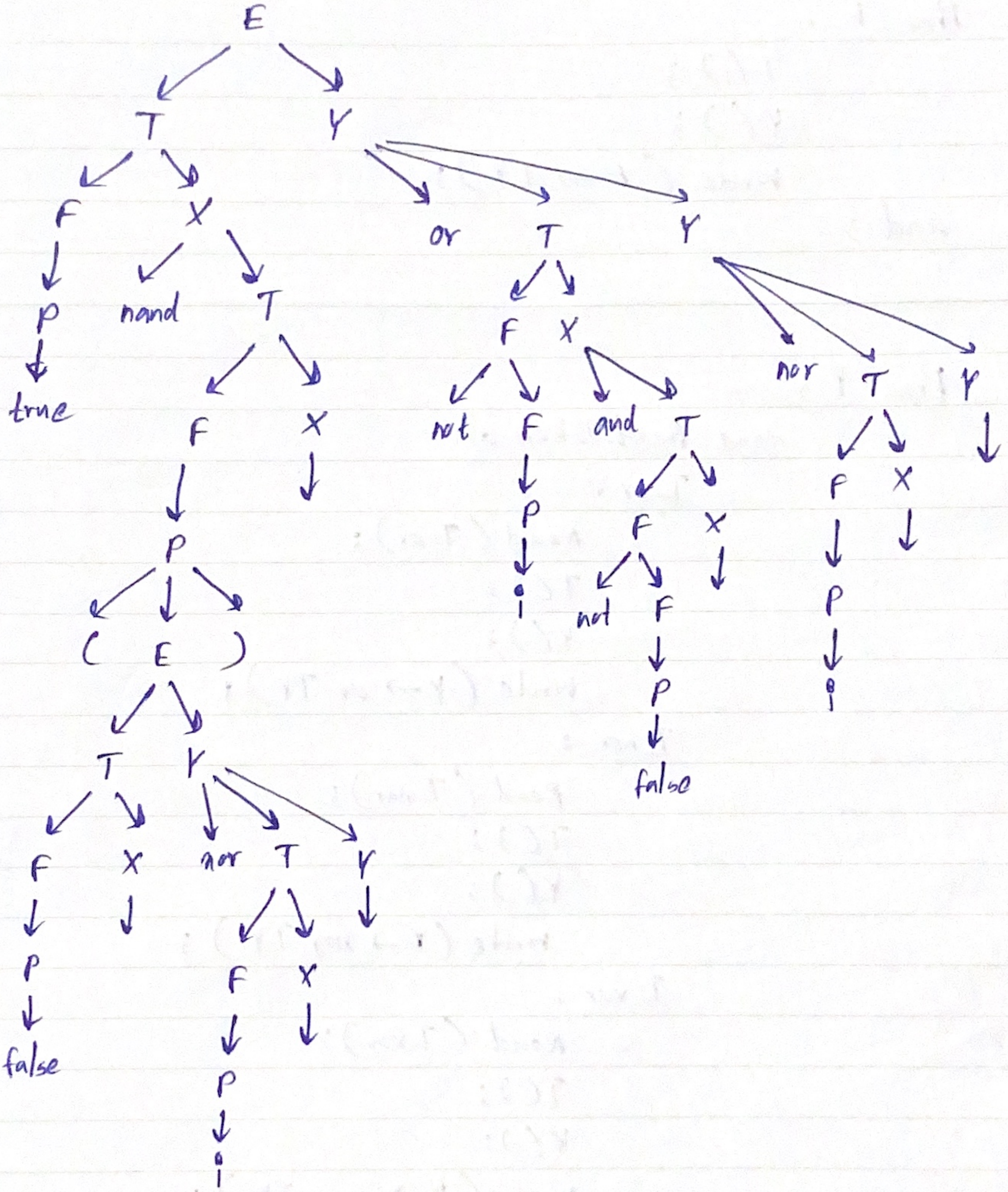
4)

Stack	Input	
E	true nand (false xor i) or not i and not false nor i ⊥	E → TY
TY	〃	T → FX
FXY	〃	F → P
PXY	〃	P → true
true XY	true nand ...	
XY	nand (false xor i) or ...	X → nand T
nand TY	nand (false xor i) or ...	
TY	(false xor i) or ...	T → FX
FXY	〃	F → P
PXY	〃	P → (E)
(E) XY	(false ...	
E) XY	false xor i) ...	E → TY
TY) XY	〃	T → FX
FXY) XY	〃	F → P
PXY) XY	〃	P → false
false XY) XY	false xor i) ...	
XY) XY	xor i) ...	X →
Y) XY	〃	Y → xor TY
xor TY) XY	xor i) ...	
TY) XY	i) ...	T → FX
FXY) XY	〃	F → P
PXY) XY	〃	P → i
iXY) XY	i) ...	
XY) XY	) or not i ...	X →
Y) XY	) or not i ...	Y →
) XY	) or not i ...	
XY	or not i ...	X →

Y	or not i ...	Y → or TY
or TY	or not i ...	
TY	not i ...	T → FX
FXY	"	F → not F
not FXY	not i and not ...	
FXY	i and not ...	F → P
PXY	"	P → i
i XY	i and not ...	
XY	and not ...	X → and T
and TY	and not false nor ...	
TY	not false nor ...	T → FX
FXY	"	F → not F
not FXY	not false nor ...	
FXY	false nor ...	F → P
PXY	"	P → false
false XY	false nor ...	
XY	nor i ⊥	X →
Y	nor i ⊥	Y → nor TY
nor TY	nor i ⊥	
TY	i ⊥	T → FX
FXY	"	F → P
PXY	"	P → i
i XY	i ⊥	
XY	⊥	X →
Y	⊥	Y →
-	⊥	

* therefore, the given input string is parsed.

05)



6)

```
Proc E;
        T();
        Y();
        Write (E → TY);
end;
```

```
Proc Y;
        case Next_Token of
            T_or:
                Read (T_or);
                T();
                Y();
                Write (Y → or TY);
            T_nor:
                Read (T_nor);
                T();
                Y();
                Write (Y → nor TY);
            T_xor:
                Read (T_xor);
                T();
                Y();
                Write (Y → xor TY);
            T_), T_⊥:
                Write (Y → );
            Otherwise:
                Error;
        end;
end;
```

```
Proc T ;
        F() ;
        X() ;
        write (T → FX) ;
end ;
```

```
Proc X ;
        case Next_Token of
            T_and :
                Read (T_and) ;
                T() ;
                write(X → and T) ;
            T_nand :
                Read (T_nand) ;
                T() ;
                write (X → nand T) ;
            T_or, T_nor, T_xor, T_+, T_) :
                write (X → ) ;
            Otherwise ;
                Error ;
        end ;
end ;
```

```
Proc F ;
        case Next_Token of
            T_not :
                Read (T_not) ;
                F() ;
                write (F → not F) ;
```

```
            T_(, T_;, T_true, T_false :
                P();
                Write(F→P);
            Otherwise :
                Error;
    end;
end;
```

```
Proc P;
    case Next_Token of
        T_(:
            Read(T_();
            E();
            Read(T_));
            Write(P→(E));
        T_;:
            Read(T_;);
            write(P→;);
        T_true:
            Read(T_true);
            write(P→true);
        T_false:
            Read(T_false);
            write(P→false);
        Otherwise:
            Error;
    end;
end;
```

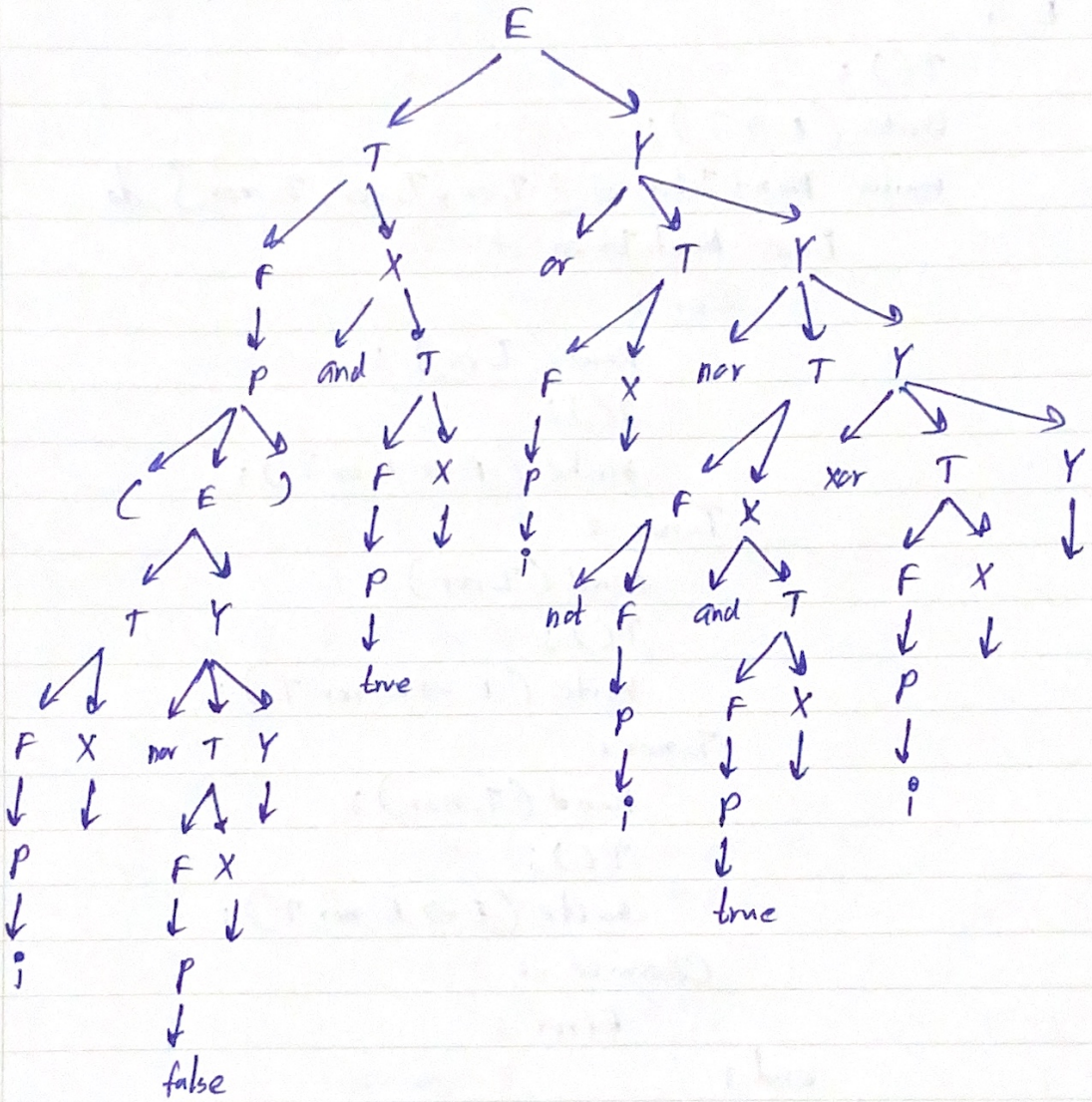
7)

P → ;
F → P
X →
T → FX
P → false
F → P
X →
T → FX
Y →
Y → nor TY
E → TY
P → (E)
F → P
P → true
F → P
X →
T → FX
X → and T
T → FX
P → ;
F → P

X →
T → FX
P → ;
F → P
F → not F
P → true
F → P
X →
T → FX
X → and T
T → FX
P → ;
F → P
X →
T → FX
Y →
Y → xor TY
Y → nor TY
Y → or TY
E → TY

Date No

a)



9) Proc E ;

```
        T();
        Write (E → T);
        While Next_Token ∈ { T_or, T_nor, T_nor } do
            Case Next_Token of
                T_or :
                    Read ( T_or) ;
                    T();
                    Write ( E → E or T );
                T_nor :
                    Read (T_nor);
                    T();
                    Write ( E → E nor T );
                T_nor:
                    Read (T_nor);
                    T();
                    write ( E → E nor T );
                Otherwise :
                    Error ;
            end ;
        end ;
end ;
```

```
Proc T ;
        F();
        Case Next_Token of
            T_and :
                Read(T_and);
                T();
                Write ( T → F and T );
```

```
            T_nand :
                Read (T_nand)
                T();
                Write (T → F nand T)
            Otherwise :
                Write (T → F);
        end;
end;
```

```
Proc F;
        case Next_Token of
            T_not :
                Read (T_not);
                F();
                Write (F → not F);
            T_(, T_i, T_true, T_false :
                P();
                Write (F → P)
            Otherwise :
                Error;
        end;
end;
```

```
Proc P;
        case Next_Token of
            T_( :
                Read (T_();
                E();
                Read (T_));
                Write (P → (E));
```

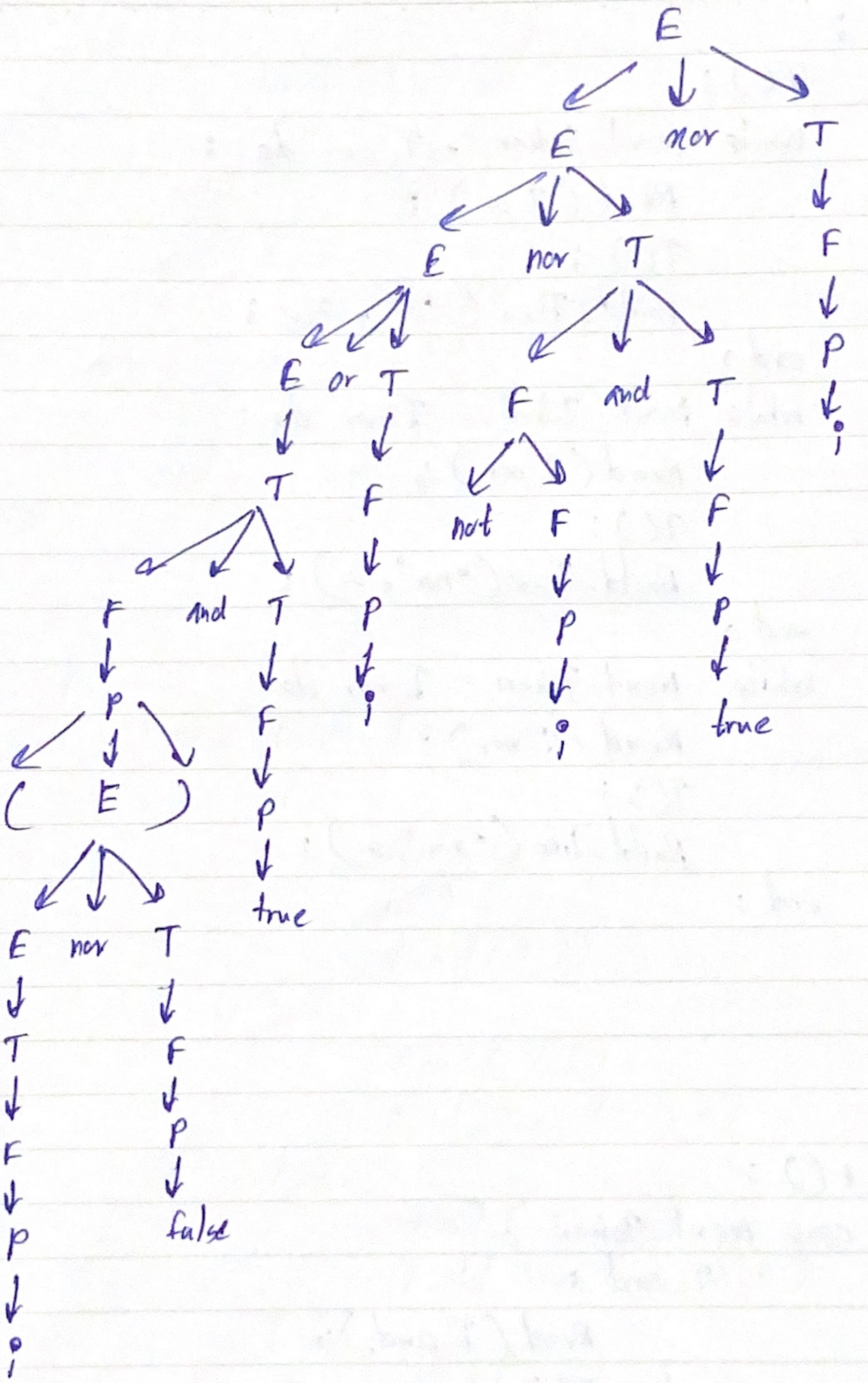
```
I_i:
        Read (I_i);
        write (P → i);
I_true :
        Read (I_true);
        write (P → true);
I_false :
        Read (I_false);
        write (P → false);
Otherwise
        Error;
    end;
end;
```

10

```
P → i
F → P
T → F
E → T
P → false
F → P
T → F
E → E nor T
P → (E)
F → P
P → true
F → P
T → F
T → F and T
E → T
```

```
P → i
F → P
T → F
E → E or T
P → i
F → P
F → not F
P → true
F → P
T → F
T → F and T
E → E nor T
P → i
F → P
T → F
E → E nor T
```

Date No



11)

```
Proc E ;
        T();
        While Next_Token = T_or do :
              Read (T_or) ;
              T() ;
              Build_Tree ( 'or', 2) ;
        end ;
        while Next_Token = T_nor do :
              Read (T_nor) ;
              T() ;
              Build_Tree('nor', 2) ;
        end ;
        while Next_Token = T_xor do:
              Read (T_xor) ;
              T() ;
              Build_Tree ('xor', 2) ;
        end ;
end ;
```

```
Proc T ;
        F() ;
        case Next_Token of
              T_and :
                    Read (T_and);
                    T();
                    Build_Tree ('and', 2) ;
              T_nand :
                    Read (T_nand);
                    T() ;
                    Build_Tree ('nand', 2) ;
        end ;
end ;
```

```
Proc F;
        case Next_Token of
            T_(, T_i, T_true, T_false :
                P();
            T_not :
                Read(T_not);
                F();
                Build_Tree('not', 1);
        end;
end;
```

```
Proc P;
        case Next_Token of
            T_( :
                Read(T_();
                E();
                Read(T_));
            T_i :
                Read(T_i);
                Build_Tree('i', 0)
            T_true :
                Read(T_true)
                Build_Tree('true', 0)
            T_false :
                Read(T_false)
                Build_Tree('false', 0)
        end;
end;
```

12)

```
BT( i,0)
BT( false ,0)
BT( nor , 2)
BT( true, 0)
BT( and, 2)
BT( i,0)
BT (or , 2)
BT (i,0)
BT ( not, 1)
BT ( true, 0)
BT (and, 2)
BT (nor, 2)
BT (i, 0)
BT ( nor, 2)
```