**Simple Program of Military Compiler**

#acquire <headern>; //<decl> -> #acquire headn ; <decl>

// <decl> -> λ

#standby <idn> ; // <dec> -> #standby idn <standn>; <dec>

// <dec> -> λ

// <standn> -> λ

PrimaryMission() // <main> -> PrimaryMission() { <statements> }

{

//clrscr();

//<statements> -> <clears> commence();

// <dec> <end>

//<dec> → <dtype> id <init> ; <dtype>

// <dtype> → unit

// <dtype> → company

// <dtype> → digit

// <dtype → joe

//<dtype> → response

// <dtype> → λ

//<init> → = <value>

// <init> →  λ

//<value> → AFFIRMATIVE

//<value> → NEGATIVE

**LOOP CFG**

<for> → inquire (<dtype> idn = <value> ; idn <logOp> <value> ; idn <unOp> ) { <statement> <for> }

<for> → λ

**Source Code:**

inquire (i = 1; I >= 5; i++)

{

post(“Number: “ i);

capture(%d, x)

}

<logOp> → <=

<logOp> → >=

<unOp> → ++

<unOp> → --

<while> → phase (<condition>)

{<statement> <while>}

<while> → λ

**Source Code:**

phase(i < 5){

post(“Enter Number: ”);

captured(%d, num);

i++;

}

<do> → go {

<statement>

<do>

} phase (<condition>);

<do> → λ

**Source Code:**

go{

}phase(i < 5 );

<if> → inorder (<condition>) {

<statement> <cont>

} <cond> <if>

<if> → λ

<cont> → roll();

<cont> → λ

<cond> → otherorder (<condition>){ <statement> }

<cond> → order { <statement> }

<cond> → λ

<switch> → campaign(<condition>) { <case> }

<case> → operation <idn> : <statement> <def>

<def> → auto : <statement>

<def> → λ

**Input and Output**

<print> → post (<pstatement>); <print>

<print> → λ

<pstatement> → “ <strstatement> “

<pstatement> → <idn>

<pstatement> → λ

<scan> → captured (sstatement); <scan>

<sstatement> → “ %<ph> ”, <idn>

<ph> → d

<ph> → f

<ph> → c

**Program:**  Program that compute the SUM of all numbers inputted in the Program.

**Structure:**

primarymission(){

unit num = 0, sum = 0, j;

company Q = “Number: ”, A = “How Many?: ”;

post(A);

captured(%d, i);

inquire( j = 1; j < i; j++){

post(Q + “[” + j + “]”);

captured(%d, num);

sum += num;

}

}

**Syntax:**

primarymission = main

unit = int

company = string

post = printf

captured = scanf

inquire = for

**Context Free Grammar:**

<program> → <main> <end>

<main> → primarymission(){<statement>}

<dec> → <dtype> idn <init>; <dec>

<dec> → λ

<dtype> → unit

<dtype> → company

<dtype> → joe

<dtype> → digit

<dtype> → response

<init> → = <value>

<init> → λ

<statement> → <dec> <print> <scan> <loop> <arith> ;

<print> → post(<sVal>); <print>

<print> → post(idn + “[“ idn + “]”); <print>

<print> → λ

<sVal> → idn

<sVal> → “ <value> ”

<sVal> → λ

<arith> → idn <Op> idn <Opn>; <arith>

<arith> → λ

<Op> → +=

<Op> → -=

<Op> → =+

<Op> → =-

<Opn> → +

<Opn> → -

<Opn> → /

<Opn> → \*

<Opn> → λ

<scan> → captured (sState); <scan>

<scan> → λ

<sState> → (“ %<ph> ”, idn);

<ph> → d

<ph> → <value>f

<ph> → c

<value> → λ

<loop> → <for> <if> <while> <do> <switch>

<for> → (idn = <value>; idn <logOp> <value>; idn <unOp>){<statement>}

<for> → λ

<logOp> → <=

<logOp> → >=

<unOp> → ++

<unOp> → --

<if> → λ

<while> → λ

<do> → λ

<switch> → λ

**Context Free Grammar**

**Variable Declaration**

<dec> → <dtype> id <init> ; <dtype>

<dtype> → unit

<dtype> → company

<dtype> → digit

<dtype → joe

<dtype> → response

<dtype> → λ

<init> → = <value>

<init> →  λ

<value> → AFFIRMATIVE

<value> → NEGATIVE

**Array**

**Single Dimension:**

<arr> → <dtype> idn[<size>] <init>; <arr>

**Double Dimension:**

<arr> → <dtype> idn[<size>][<size>] <init>; <arr>

<arr> → λ

<size> → numeric

<init> → = {<elements>}

<init> → λ

**Functions**

<funct> → <dtype> id (<dtype> id, <dtype> id) {<statement>}

<funct> → <dtype> id (<dtype> id) {<statement>}

<funct> → λ

**Struct**

<struct> → struct <structN>{

<dec> → <dtype> (<mem>); <dec>

<dec> → λ

};

**Defining Variable:**

<decstruct> → miss (<structN>)

**Accessing members:**

<AccMem> → <structvarN>.<mem>;

1. <Program> → <header> <globalDec> <main> <dec> <body> <end
2. <header> → #standby <HeadN>; <header>
3. <header> → λ
4. <globalDec> →#standby idn <initN>; <dec>
5. <globalDec> -> λ
6. <initN> -> = <value>
7. <initN> → λ
8. <main> → PrimaryMission(){ <statements> }
9. <dec> → <LocalDec> <arr> <funct> <struct>
10. <LocalDec> → <dtype> id <init> ; <dtype>
11. <LocalDec> → λ
12. <dtype> → unit
13. <dtype> → company
14. <dtype> → digit
15. <dtype → joe
16. <dtype> → response
17. <dtype> → λ
18. <init> → = <value>
19. <init> →  λ
20. <arr> → <dtype> idn[<size>] <init>; <arr>
21. <arr> → <dtype> idn[<size>][<size>] <init>; <arr>
22. <arr> → λ
23. <size> → numeric
24. <init> → = {<elements>}
25. <init> → λ
26. <funct> → <dtype> id (<dtype> id, <dtype> id) {<statement>}
27. <funct> → <dtype> id (<dtype> id) {<statement>}
28. <funct> → λ
29. <struct> → struct <structN>{ <dec> → <dtype> (<mem>); <dec> <dec> → λ }
30. <struct> → λ

**Source Code:**

**Global Declaration:**

unit A = 0, B = 0;

company Q = “HELLO WORLD”, C;

PrimaryMission(){

}

**Local Declaration:**

PrimaryMission(){

unit A = 0, B = 0;

company Q = “HELLO WORLD”, C;

}

**Array Declaration:**

PrimaryMission(){

unit A[ ] = {‘A’, ‘B’, ‘C’};

}

**Functions:**

PrimaryMission(){

unit Sum (int A, int B){

Sum = A + B;

}

}