Compilers Lab Assignment 2

-	Chinmoy Jyoti Kathar	150101019
-	Chinmoy Kachari	150101020
-	Nayan Jyoti Kakati	150101041
-	Pulak Kuli	150101050

Context Free Grammar For Language For Emulator Generator:

```
OP        ->  ==  |  <  |  >  |  !=  |  <=    |  >=
TYPE      ->  int  |  str
```

Context Free Grammar For Language For Emulator Generator:

```
RELATION      -> EXPR OP EXPR | EXPR
EXPR          -> TERM EXPR'
EXPR'         -> + TERM EXPR' | - TERM EXPR' |      ε
TERM          ->  FACTOR TERM'
TERM'         -> * FACTOR TERM' | / FACTOR TERM' |    ε
FACTOR        ->    ID  |    integer  |decimal  | ID | ( EXPR ) | ! FACTOR
                 | ! ( EXPR ) | ID.call_class_or_func
                 | call_class_or_func
```

Context Free Grammar For Language For Emulator Generator:

```
S               -> HEADER pgm
HEADER          -> HASH_INCLUDE < ID.h > HEADER |   ε
pgm             -> var_dec; pgm | f_dec pgm |  class_dec; pgm |   ε
var_dec         -> TYPE ID = EXPR
f_dec           -> TYPE ID ( dec_param_list ) { stmt_list }
class_dec       -> class ID ( param_list ) { dec_list }
dec_list        -> f_dec; dec_list | var_dec; dec_list |    ε
dec_param_list  -> TYPE ID, dec_param_list |    ε
```

S is the start symbol , **HEADER** is the production for defining the headers and the program , pgm is the production for the program lines of the file

Context Free Grammar For Language For Emulator Generator:

```
stmt_list               -> stmt; stmt_list |    ε
stmt                    ->   TYPE ID | TYPE ID = RHS |ID = RHS |   RHS      |
                             ID [ EXPR ] = RHS | while ( RELATION ) stmt     |
                             IF_ELSE |{ stmts }  | for ( stmt ; RELATION ; stmt)
                             stmt |   ε
RHS                      -> ID.call_class_or_func | call_class_or_func
                             | ID | EXPR
call_class_or_func       -> ID ( param_list )
```

stmt_list is a list of statements.

RHS is the production for function calls , calling a method of a class or ID or expression.

Context Free Grammar For Language For Emulator Generator:

```
param_list   -> param, param_list |    ε
param        -> EXPR | ID = ID | ID = EXPR | ID
                      | ID = call_class_or_func
                      | call_class_or_func | [ ID_LIST ]
                      | ID = [ ID_LIST ]
                      | ID = STRING | STRING
ID_LIST      -> ID , ID_LIST | EXPR , ID_LIST | ID | EXPR
```

param_list is the list of parameters passed in a function or method. Where parameter can be an ID , expression , string , an array , the output of another function , etc.

Context Free Grammar For Language For Emulator Generator:

```
IF_ELSE   ->   MIF      |     UIF
MIF       ->   if ( RELATION ) MIF else MIF | stmt
UIF       ->   if ( RELATION ) MIF else UIF
                   | if ( RELATION ) stmt
```

This handles the dangling if and else problem.

Context Free Grammar For Language For Emulator Generator:

Test Case:

```
#include <myheader.h>
int a = 3;

int caller(int a,int b){
    a = b*2 + a;
    printf();
}

int main(){
    int myvar = caller();
```

Context Free Grammar For Language For Emulator Generator:

```
int a ;
a = b;
b = 3*4 + b;
x = process();
int x = func();
b = x* func();
job_1 = Job(  job_id=1, flops_required = 100, deadline = 200,
              mem_required = 1024,affinity = [0.2,0.5,1,2]  )

mem1 = Memory(memory_type= ‘cache’, mem_size=1)
```

Context Free Grammar For Language For Emulator Generator:

```
ram = Memory(memory_type= ‘primary’, mem_size = 2048, name =
                   “ram1”)

proc_1 = Processor(isa = ‘ARM’, clock_speed = 40, l1_memory = mem1)

link_1 = Link(start_point = “proc_1”, end_point= “ram1”, 40, 50)
```

Context Free Grammar For Language For Emulator Generator:

```
while( ! Ram.get_available_memory( ) ){
        wait(1)
        }
    if ( job_1.get_memory( ) <= ram.get_available_memory( ) ) {
    proc_1.submit_jobs( job_1 )
    }
    else{
        discard_job( job_1 )
    }


    return 0;
}
```

Extensions to the language/compiler:

Scheduler Class:

Parameters:

1. List of Jobs
2. List of Processor
3. Algorithm Option for scheduling (eg. Round Robin, FCFS)

Scheduling will be based on the available memory required for a job, affinity of the job for each type of processor, job's flops required, each processor's allocated memory and it's clock speed.

Instructions to run parser:

Run these in terminal

```
lex tokenx.l
gcc parser.c lex.yy.c

./a.out < <input_file> (e.g. ./a.out < x.x)
```

Lexer File Program: (tokenx.l)

```
%{
#include "tokenx.h"
%}


%%
```

```
[ \n\t] {
        ;
}
[0-9]+ {
        return INT;
}
[0-9]+\.[0-9] {
        return FLOAT;
}

\"[^\"]*\" {
        return STRING;
}
\'[^\']*\' {
        return STRING;
}

"!" {
        return NOT;
}

"(" {
        return OPEN_ROUND;
}

")" {
        return CLOSE_ROUND;
}

"{" {
        return OPEN_CURLY;
}

"}" {
        return CLOSE_CURLY;
}

"[" {
        return OPEN_SQUARE;
}

"]" {
        return CLOSE_SQUARE;
}
"." {
        return DOT;
}

";" {
        return SEMI;
}
"++" {
        return INC;
}
"+" {
        return PLUS;
}

"*" {
        return TIMES;
}

"--" {
        return DEC;
}

"-" {
        return MINUS;
}

"/" {
        return DIV;
}

"=" {
        return ASSGN;
}

"," {
        return COMMA;
}

":" {
        return COLON;
}
```

```
"==" {
        return EQUAL;
}

"!=" {
        return NOT_EQUAL;
}

"<=" {
        return LESS_EQUAL;
}

">=" {
        return GREATER_EQUAL;
}

"<" {
        return LESS;
}
">" {
        return GREATER;
}

"for" {
        return FOR;
}

"while" {
        return WHILE;
}

"if" {
        return IF;
}

"else" {
        return ELSE;
}

"int" {
        return DTYPE_INT;
}
```

```
"float" {
	return DTYPE_FLOAT;
}

"string" {
	return DTYPE_STRING;
}

"#include" {
	return HASH_INCLUDE;
}

"class" {
	return CLASS_IDF;
}

[a-zA-Z][a-zA-Z0-9_]* {
	return ID;
}
. {
	printf("Unexpected Character : %c\n",*yytext);
}


%%


int yywrap(void){
	return 1;
}
```

Header File For Lexer(tokenx.h)

```
#define INT			1
#define FLOAT		2
#define STRING		3

#define NOT				4	/* ! */
```

```
#define OPEN_ROUND            5     /* ( */
#define CLOSE_ROUND           6     /* ) */
#define OPEN_CURLY            7     /* { */
#define CLOSE_CURLY           8     /* } */
#define OPEN_SQUARE           9     /* [ */
#define CLOSE_SQUARE      10    /* ] */
#define DOT               11    /* . */

#define SEMI              12    /* ; */

#define INC                   13    /* ++ */
#define PLUS              14    /* + */
#define TIMES             15    /* * */
#define DEC                   16    /* --  */
#define MINUS             17    /* - */
#define DIV                   18    /* / */
#define ASSGN             19    /* = */
#define COMMA             20    /* , */
#define COLON             21    /* : */

#define EQUAL             22    /* == */
#define NOT_EQUAL         23    /* != */
#define LESS_EQUAL        24    /* <= */
#define GREATER_EQUAL     25    /* >= */
#define LESS              26    /* < */
#define GREATER           27    /* > */

#define FOR                   28
#define WHILE             29
#define IF                30
#define ELSE              31

#define DTYPE_INT         32    /* int */
```

```
#define DTYPE_FLOAT                     33     /* float */
#define DTYPE_STRING        34     /* string */

#define HASH_INCLUDE        35     /* #include */
#define CLASS_IDF           36

#define ID                  37
```

Parser File for Output (parser.c)

```
#include <stdio.h>

extern int yylex();
extern int yylneno;
extern char *yytext;

int main(){
        int ntoken, vtoken;

        FILE *f = fopen("output.txt","w");


        while(ntoken = yylex()){
                fprintf(f,"Token ( %s, %d)\n",yytext, ntoken);
        }
}
```

Sample Code for parser: (x.x)

```
#include <myheader.h>
int a = 3;
int caller(int a,int b){
    a = b*2 + a;
    printf();
}

int main(){
    int myvar = caller();
    int a ;
    a = b;
    b = 3*4 + b;
    x = process();
    func();
    b = x* func();

    job_1 = Job(job_id=1, flops_required = 100, deadline = 200,
mem_required = 1024,affinity = [0.2,0.5,1,2])
    mem1 = Memory(memory_type= 'cache', mem_size=1)
    ram = Memory(memory_type= 'primary', mem_size = 2048,, name =
"ram1")
    proc_1 = Processor(isa = 'ARM', clock_speed = 40, l1_memory = mem1)
    link_1 = Link(start_point = "proc_1", end_point= "ram1", 40, 50)
    while(!==Ram.get_available_memory()){
        wait(1)
        }
    if ( job_1.get_memory() <= ram.get_available_memory() ) {
    proc_1.submit_jobs(job_1)
    }
    else{
        discard_job(job_1)
    }


    return 0;
}
```