**M S Ramaiah Institute of Technology**

**(Autonomous Institute Affiliated to VTU)**

**Department of Information Science and Engineering**

**Course File**

**(Applicable for Academic Year 2016 - 2017)**

**Semester: V I**

**Course Code: IS631L**

**Course Name: System Software Lab**

**Course Credits: 0:0:1**

**Course Faculty:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl#** | **Section** | **Course Faculty Name** | **Sign** | **Date** |
| **1.** | **A** | **Savita K Shetty** |  |  |
| **2.** | **B** | **Sumana M** |  |  |
| **3.** | **C** | **Myna A N** |  |  |

**Course Coordinator (Name,Sign,Date): Sumana M**

**3rdYear Coordinator (Name, Sign, Date):**

**Dept.Program Coordinator (Name,Sign,Date):** Dr.Mydhili K Nair

**Head of Department (Sign & Date)**

**COURSE DESIGN, DELIVERY AND ASSESMENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Course code and Title : IS631L, System Software Lab | | | Course Credits :0:0:1 | |
| CIE : **50 Marks** | | | SEE : **50 Marks** | |
| Total No of Theory / Tutorial / Lab Hours : 24 hours | | | | |
| Prepared by : Savita K Shetty | | | Date : | |
| Reviewed by : | | | Date : | |
| **Prerequisites**   |  | | --- | | **Prerequisite Courses with codes** | | **NIL** | |  | |

**Course objectives:**

1. The lab focuses on the experimental design, implementation, and evaluation of systems software technologies
2. To acquire basic skills and knowledge in constructing two-pass assemblers.
3. To analyze the concept of absolute and relocating loaders.
4. To understand and study of lex and yacc tools which helps write programs whose control flow is directed by instances of regular expressions in the input stream
5. Realize the role of macroprocessors in system software.

**Syllabus**

**PART-A**

**Implement the following using C/C++ for the SIC Machine:**

1. Develop pass-1 of two-pass assembler.
2. Develop pass-2 of two-pass assembler.
3. Develop absolute loader.
4. Develop relocating loader using bit mask.
5. Develop one-pass macroprocessor.

**PART-B**

LEX Exercises.

1. Develop a lex program to count the number of characters, words, spaces and lines in a given input file.
2. Develop a lex program to count the number of comment lines in a given C program.
3. Develop a lex program to recognize a valid arithmetic expression and identify the identifiers and operators present.
4. Develop a lex Program to recognize and count the number of identifiers in a given input file.

YACC Exercises.

1. Develop a yacc program to recognize a valid arithmetic expression that uses operators + ,- ,\* and /.
2. Develop a yacc program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.
3. Develop a yacc program to evaluate an arithmetic expression involving operators +,-,\* and /.
4. Develop a yacc program to recognize the strings of the form ambn, where m>=0,n>=0.

**Text Book:**

1. Leland.L.Beck, System Software, 3rd Edition, Addison-Wesley, 1997.

2. John.R.Levine, Tony Mason and Doug Brown, Lex and Yacc, O'Reilly, SPD, 1999.

3. Alfred V Aho, Monica S. Lam, Ravi Sethi, Jeffrey D Ullman, Compilers- Principles, Techniques and Tools, Second Edition, Addison-Wesley, 2007.

**Reference Books**:

1. D.M.Dhamdhere, System Programming and Operating Systems, 2nd Edition, Tata McGraw Hill, 1999.

**Concept map**

System Software Lab

S

1. Two pass assemblers

2. Implementing Loaders

3. Implementing Macroprocessor

4. Lex programs

5. YACC programs

4.1. count the number of characters, words, spaces and lines in a given input file.

4.2 number of comment lines

4.3 recognize a valid arithmetic expression

4.4 count the number of identifiers.

3.1 One pass macroprocessors

2.1 Absolute loaders

2.2 Relocating loaders with bit mask.

1.1. Introduction to SIC programs

1.2. Pass 1 of two pass assemblers

1.3. Pass 2 of two pass assemblers.

5.1 a valid arithmetic expression

5.2 Evaluation of expressions

5.3 Valid variables

5.4 recognize strings

Involves

Deals with

Involves

Deals with

Involves

**Course Contents and Lecture Schedule**

| **Lesson No/Session No** | **Topics** | **No. of hours** |
| --- | --- | --- |
| 1. | Develop pass-1 of two-pass assembler. | 2 |
| 2. | Develop pass-2 of two-pass assembler. | 2 |
| 3. | Develop pass-2 of two-pass assembler. | 2 |
| 4. | Develop absolute loader. | 2 |
| 5. | Develop relocating loader using bit mask. | 2 |
| 6. | Develop one-pass macroprocessor | 2 |
|  | **LEX and YACC programs** |  |
| 7 | Develop a lex program to count the number of characters, words, spaces and lines in a given input file.  Develop a lex program to count the number of comment lines in a given C program | 2 |
| 8 | Develop a lex program to recognize a valid arithmetic expression and identify the identifiers and operators present.  Develop a lex Program to recognize and count the number of identifiers in a given input file. | 2 |
| 9 | Develop a yacc program to recognize a valid arithmetic expression that uses operators + ,- ,\* and /. | 2 |
| 10 | Develop a yacc program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits. | 2 |
| 11 | Develop a yacc program to evaluate an arithmetic expression involving operators +,-,\* and /. | 2 |
| 12 | Develop a yacc program to recognize the strings of the form ambn, where m>=0,n>=0. | 2 |

**Course outcomes:**

**CO1:** Design and implement 2 pass assemblers for SIC machine.

**CO2:** Design and implement loaders.

**CO3:** Design and implement macroprocessors.

**CO4:** Design and implement lex programs.

**CO5:** Design and implement YACC programs

**Note:**

In SEE, student has to execute one program from PART-A and one LEX / one YACC program from PART-B.

**Mapping Course Outcomes with Program Outcomes:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Course Outcomes** | | **PO-a** | **PO-b** | **PO-c** | **PO-d** | **PO-e** | **PO-f** | **PO-g** | **PO-h** | **PO-i** | **PO-j** | **PO-k** | **PO-l** |
| 1. | Design and implement 2 pass assemblers for SIC machine. | X |  |  | X |  |  |  |  |  | X |  |  |
| 2. | Design and implement loaders. | X |  |  | X |  |  |  |  |  | X |  |  |
| 3. | Design and implement macroprocessors. | X |  |  | X |  |  |  |  |  | X |  |  |
| 4. | Design and implement lex program. |  |  |  | X |  |  |  |  |  | X |  |  |
| 5. | Design and implement YACC program. |  |  |  | X |  |  |  |  |  | X |  |  |

**Course Assessment and Evaluation:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **What** | | **To whom** | **When/ Where (Frequency in the course)** | **Max marks** | **Evidence collected** | **Contributing to Course Outcomes** |
| **Direct Assessment Methods** | CIE | Assessment of experimental Skills / Weekly Evaluation | Students | Weekly | 10 | Record Book | CO1,CO2,CO3,CO4,CO5 |
| Mid-Sem Internals | Once | 20 | Data Sheets | CO1,CO2,CO3 |
| Final Internals | Once | 20 | Data Sheets | CO1,CO2,CO3,CO4,CO5 |
| SEE | Standard examination | End of course | 50 | Answer scripts | CO1,CO2,CO3,CO4,CO5 |
| **Indirect Assessment Methods** | Students feedback | | Students | Middle of the course | - | Feedback forms | CO1,CO2,CO3 |
| End of course survey | | End of course | - | Questionnaire | CO1,CO2,CO3,CO4,CO5 |

Questions for CIE and SEE will be designed to evaluate the various educational components (Bloom’s taxonomy) such as:

CIE and SEE evaluation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Bloom’s Category | Test 1 | Test 2 | Semester-End Examination |
| 1 | Remember | 0 | 0 | 0 |
| 2 | Understand | 20 | 20 | 20 |
| 3 | Apply | 60 | 60 | 70 |
| 4 | Analyze | 10 | 10 | 10 |
| 5 | Evaluate | 20 | 20 | 0 |
| 6 | Create | 0 | 0 | 0 |

|  |  |  |
| --- | --- | --- |
|  | Name | Signature with Date |
| Prepared by |  |  |
| Reviewed by |  |  |

**Lab Manual**

Experiment No-1: **Develop pass-1 of two-pass assembler**

ALGORITHM

1. Open the files fp1 and fp4 in read mode and fp2 and fp3 in write mode.
2. Read the source program.
3. If the opcode read in the source program is START, the variable location counter is initialized with the operand value.
4. Else the location counter is initialized to 0.
5. The source program is read line by line until the reach of opcode END.
6. Check whether the opcode read is present in the operation code table.
7. If the opcode is present, then the location counter is incremented by 3.
8. If the opcode read is WORD, the location counter is incremented by3.
9. If the opcode read is RESW, the operand value is multiplied by 3 and then the location counter is incremented.
10. If the opcode read is RESB, the location counter value is incremented by operand value.
11. If the opcode read is BYTE, the location counter is auto incremented.

The length of the source program is found using the location counter value.

Experiment No-2: **Develop pass-2 of two-pass assembler**

ALGORITHM

1. Start the program

2. Initialize all the variables

3. Open file to

i. generate assembly listing.

ii. to generate an refer to symbol table.

iii. to generate intermediate file.

iv. to maintain opcode of the SIC instructions.

4. Read the content of the intermediate file

5. If opcode is BYTE Then

Write hex equivalent to output file

else if opcode is WORD then

Write hex equivalent to output file

Else if opcode is RESB or RESW

Ignore and do not copy to output file

If symbol then place the symbol value from symbol table

Else if opcode then refer optable and generate the opcode

Place the opcode and the symbol value to output file.

6. Repeat step 5 till end of intermediate file

7. Finally terminate the pass two of two pass assembler.

Possible input for experiment 1 and 2

**Three Input Files:**

***1) INPUT.txt***

COPY START 2000

LDA FIVE

STA ALPHA

LDCH CHARZ

STCH C1

ALPHA RESW 1

FIVE WORD 5

CHARZ BYTE C’Z’

C1 RESB 1

END

***2) OPTAB.txt***

LDA 00

STA 0C

LDCH 50

STCH 54

***3) SYMTAB.txt***

ALPHA 200C

FIVE 200F

CHARZ 2012

C1 2013

**One Output File:**

***OUTPUT.txt***

START 2000

2000 LDA FIVE 00200F

2003 STA ALPHA 0C200C

2006 LDCH CHARZ,X 50A012

2009 STCH C1 542013

200C ALPHA RESW 1

200F FIVE WORD 5 000005

2012 CHARZ BYTE C’A’ 41

2013 C1 RESB 1

2014 END

Experiment No-3: **Absolute Loader Implementation**

ALGORITHM:

1. Start the program

2. Assign the required variable

3. Open the files

fp1=fopen("input.dat","r");

fp2=fopen("output.dat","w");

4. Read the content

5. Using while loop perform the loop until character is not equal to E

**Possible input and expected output for Experiment 3**

**Input File:**

H COPY 001000 000013

T 001000 09 142033 483039 102036 220044

T 002000 0A 298300 230000 F1 302015

E 001000

**Output File:**

1000 14

1001 20

1002 33

1003 48

1004 30

1005 39

1006 10

1007 20

1008 36

1009 22

100A 00

100B 44

2000 29

2001 83

2002 00

2003 23

2004 00

2005 00

2006 F1

2007 30

2008 20

2009 15

Experiment No-4: **Relocating Loader using using bit-mask**

1. Start the program

2. Include the necessary header file and variable

3. Open the two file for

fp1= relinput.dat and give read

fp2= reloutput.dat and give write

4. Read the content

5. Using while loop perform the loop until character is not equal to E

while(strcmp(input,"E")!=0)

If the character is H

Get the variable add, length, and input

Else if the character is T

Get the variable address and bitmask

And perform the for loop for starting zero to up to len

Get the opcode ,addr and assign relocbit to bitmask

If relocabit is zero

Then

actualadd=addr;

else

Add the addr and star value

6. Finally terminate the program.

**Possible input and expected output for Experiment 4**

**Input File:**

H COPY 000000 00

T 000000 C80 141033 481039 901776 921765 571765

T 002011 E00 232838 432979 892060 662849

E 000000

**Output File:**

Enter the actual starting address : 4000

4000 145033

4003 485039

4006 901776

4009 921765

400C 575765

6011 236838

6014 436979

6017 896060

601A 662849

Experiment No-5:**Implement one-pass macroprocessor**

**ALGORITHM**

1.Start the macro processor program

2. Include the necessary header files and variable

3. Open the three files

f1=macin.dat with read privilege

f2=macout.dat with write privilege

f3= deftab.dat with write privilege

4. Get the variable form f1 file macin.dat for label,opcode,operand

5. Read the variable until the opcode is not is equal to zero

Then check if the opcode is equal to Macro if Macro

Then

6. Copy macroname=label

7. Get the variable label ,opcode ,operand

8. In these if condition perform the while loop until opcode is not equal to MEND

9. Copy the variable

d[lines].lab=label

d[lines].opc=opcode

d[lines].oper=operand

and increase lines++;

close while loop and if condtion

else if opcode is equal to macro name

10. Perform the for loop from 0 to length

fprint for d[i].lab,d[i].opc,d[i].oper

else if it is not match

fprintf(f2,"%s\t%s\t%s\n",label,opcode,operand);

11. Finally terminate the program

**Possible input and expected output for Experiment 5**

**INPUT FILE :**

**MACIN.DAT**

CALC START 1000

SUM MACRO \*\*

\*\* LDA #5

\*\* ADD #10

\*\* STA 2000

\*\* MEND \*\*

\*\* LDA LENGTH

\*\* COMP ZERO

\*\* JEQ LOOP

\*\* SUM \*\*

LENGTH WORD 5

ZERO WORD 0

LOOP SUM \*\*

\*\* END \*\*

**OUTPUT FILES :**

**MACOUT.DAT**

CALC START 1000

\*\* LDA LENGTH

\*\* COMP ZERO

\*\* JEQ LOOP

\*\* LDA #5

\*\* ADD #10

\*\* STA 2000

LENGTH WORD 5

ZERO WORD 0

\*\* LDA #5

\*\* ADD #10

\*\* STA 2000

\*\* END \*\*

**DEFTAB.DAT**

\*\* LDA #5

\*\* ADD #10

\*\* STA 2000

**Lex Programs**

1. Develop a lex program to count the number of characters, words, spaces and lines in a given input file.

BLANK [ ]

%%

{WORD} {w++; c=c+yyleng;}

{BLANK} {s++;}

{EOL} {l++;}

**.** {c++;}

%%

int yywrap()

{

return 1;

}

main(int argc, char \*argv[])

{

If(argc!=2)

{

printf(“Usage: <./a.out> <sourcefile>\n”);

exit(0);

}

yyin=fopen(argv[1],”r”);

yylex();

printf(“No of characters=%d\nNo of words=%d\nNo of spaces=%d\n No of lines=%d”,c,w,s,l);

}

2. Program to count the no of comment line in a given C program. Also eliminate them and copy that program into separate file

%{

#include<stdio.h>

int com=0;

%}

%s COMMENT

%%

“/\*”[**.**]\*”\*/” {com++;}

“/\*” {BEGIN COMMENT ;}

<COMMENT>”\*/” {BEGIN 0; com++ ;}

<COMMENT>\n {com++ ;}

<COMMENT>**.** {;}

**.**|\n {fprintf(yyout,”%s”,yytext);

%%

int yywrap()

{

return 1;

}

main(int argc, char \*argv[])

{

If(argc!=2)

{

printf(“Usage: <./a.out> <sourcefile> <destn file>\n”);

exit(0);

}

yyin=fopen(argv[1],”r”);

yyout=fopen(argv[2],”w”);

yylex();

printf(“No of comment lines=%d\n”,com);

}

3. Program to recognize a valid arithmetic expression and identify the identifiers and operators present. Print them separately.

%{

#include<stdio.h>

#include<string.h>

int noprt=0, nopnd=0, valid=1, top=-1, m, l=0, j=0;

char opnd[10][10], oprt[10][10], a[100];

%}

%%

“(“ { top++; a[top]=’(‘ ; }

“{“ { top++; a[top]=’{‘ ; }

“[“ { top++; a[top]=’[‘ ; }

“)” { if(a[top]!=’(‘)

{

valid=0; return;

}

else

top--;

}

“}” { if(a[top]!=’{‘)

{

valid=0; return;

}

else

top--;

}

“]” { if(a[top]!=’[‘)

{

valid=0; return;

}

else

top--;

}

“+”|”-“|”\*”|”/” { noprt++;

strcpy(oprt[l], yytext);

l++;

}

[0-9]+|[a-zA-Z][a-zA-Z0-9\_]\* {nopnd++;

strcpy(opnd[j],yytext);

j++;

}

%%

int yywrap()

{

return 1;

}

main()

{

int k;

printf(“Enter the expression.. at end press ^d\n”);

yylex();

if(valid==1 && i==-1 && (nopnd-noprt)==1)

{

printf(“The expression is valid\n”);

printf(“The operators are\n”);

for(k=0;k<l;k++)

Printf(“%s\n”,oprt[k]);

for(k=0;k<l;k++)

Printf(“%s\n”,opnd[k]);

}

else

Printf(“The expression is invalid”);

}

4. Program to recognize and count the number of identifiers in a given input file.

%{

#include<stdio.h>

int id=0;

%}

%%

[a-zA-Z][a-zA-Z0-9\_]\* { id++ ; ECHO; printf(“\n”);}

**.**+ { ;}

\n { ;}

%%

int yywrap()

{

return 1;

}

main (int argc, char \*argv[])

{

if(argc!=2)

{

printf(“Usage: <./a.out> <sourcefile>\n”);

exit(0);

}

yyin=fopen(argv[1],”r”);

printf(“Valid identifires are\n”);

yylex();

printf(“No of identifiers = %d\n”,id);

}

**YACC Programs**

1. Develop a yacc program to recognize a valid arithmetic expression that uses operators + ,- ,\* and /.

Yacc Part

%token NUMBER ID NL

%left ‘+’ ‘-‘

%left ‘\*’ ‘/’

%%

stmt : exp NL { printf(“Valid Expression”); exit(0);}

;

exp : exp ‘+’ exp

| exp ‘-‘ exp

| exp ‘\*’ exp

| exp ‘/’ exp

| ‘(‘ exp ‘)’

| ID

| NUMBER

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

main ()

{

printf(“Enter the expression\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

[0-9]+ { return DIGIT; }

[a-zA-Z][a-zA-Z0-9\_]\* { return ID; }

\n { return NL ;}

**.** { return yytext[0]; }

%%

2. Develop a yacc program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.

Yacc Part

%token DIGIT LETTER NL UND

%%

stmt : variable NL { printf(“Valid Identifiers\n”); exit(0);}

;

variable : LETTER alphanumeric

;

alphanumeric: LETTER alphanumeric

| DIGIT alphanumeric

| UND alphanumeric

| LETTER

| DIGIT

| UND

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

main ()

{

printf(“Enter the variable name\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

[a-zA-Z] { return LETTER ;}

[0-9] { return DIGIT ; }

[\n] { return NL ;}

[\_] { return UND; }

**.** { return yytext[0]; }

%%

3. Develop a yacc program to evaluate an arithmetic expression involving operators +,-,\* and /.

Yacc Part

%token NUMBER ID NL

%left ‘+’ ‘-‘

%left ‘\*’ ‘/’

%%

stmt : exp NL { printf(“Value = %d\n”,$1); exit(0);}

;

exp : exp ‘+’ exp { $$=$1+$3; }

| exp ‘-‘ exp { $$=$1-$3; }

| exp ‘\*’ exp { $$=$1\*$3; }

| exp ‘/’ exp { if($3==0)

{

printf(“Cannot divide by 0”);

exit(0);

}

else

$$=$1/$3;

}

| ‘(‘ exp ‘)’ { $$=$2; }

| ID { $$=$1; }

| NUMBER { $$=$1; }

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

main ()

{

printf(“Enter the expression\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

extern int yylval;

%}

%%

[0-9]+ { yylval=atoi(yytext); return NUMBER; }

\n { return NL ;}

**.** { return yytext[0]; }

%%

4. Develop a yacc program to recognize the strings of the form ambn, where m>=0,n>=0.

Yacc Part

%token A B NL

%%

stmt : s NL { printf(“Valid String\n”); exit(0) ;}

;

s : A s B

|

;

%%

int yyerror(char \*msg)

{

printf(“Invalid String\n”);

exit(0);

}

main ()

{

printf(“Enter the String\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

[aA] { return A; }

[bB] { return B; }

\n { return NL ;}

**.** { return yytext[0]; }

%%

Steps to Execute Lex Program:

lex <pgm name>

cc lex.yy.c –ll

./a.out

Steps to execute YACC program:

yacc –d <yacc\_pgm name>

lex <lex\_pgm\_name>

cc y.tab.c lex.yy.c –ly –ll

./a.out

**M.S. Ramaiah Institute of Technology**

**(An Autonomous Institute affiliated to VTU)**

**Department of Information Science and Engineering**

**Course Code: IS513L Course Name: System Software Lab**

**Semester: 5 Max Marks:50**

**Rubrics for Assessment of Student**

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Scoring Criteria** | **Marks** | **Score**  **(50)** |
| **Evaluation (20)**  **Lab Test1** | **VIVA:** |  |  |
| Has not answered any question | 0 |  |
| Has answered 1 question | 1 |  |
| Has answered 2 questions | 2 |  |
| Has answered 3 questions | 3 |  |
| Has answered 4 questions | 4 |  |
| Has answered 5 questions | 5 |  |
| **WRITEUP:** |  |  |
| Entirely Wrong Program Code | 0 |  |
| Written code not clear | 2 |  |
| Writeup clear and appropriate | 5 |  |
| **Execution:** |  |  |
| Errors / output totally irrelevant | 0 |  |
| Partial Output | 5 |  |
| Appropriate and clear Output | 10 |  |
| **Evaluation (20)**  **Lab Test** | **VIVA:** |  |  |
| Has not answered any question | 0 |  |
| Has answered 1 question | 1 |  |
| Has answered 2 questions | 2 |  |
| Has answered 3 questions | 3 |  |
| Has answered 4 questions | 4 |  |
| Has answered 5 questions | 5 |  |
| **WRITEUP:** |  |  |
| Entirely Wrong Program Code | 0 |  |
| Written code not clear | 2 |  |
| Writeup clear and appropriate | 5 |  |
| **Execution:** |  |  |
| Errors / output totally irrelevant | 0 |  |
| Partial Output | 5 |  |
| Appropriate and clear Output | 10 |  |
| No Output/ Partial output but right program code. | 5 |  |
| Correct Output but wrong program code and no knowledge / conclusions of the program. | 8 |
| Correct Output and program code with proper conclusions of the program derived. | 15 |
| **(Regular Evaluation)**  **(Execution – 5 M)** | Student has attended less than 50% of the lab sessions and unaware of the experiment performed. The results are entirely meaningless. | 0 |  |
| Student is regular to lab but unaware of the experiment performed. | 1 |  |
| Student is regular to lab and performed the experiment. However the procedure is not well designed. | 2 |  |
| Student is regular to lab and performed the experiment. The procedure is well designed. However Data presented is irrelevant and provides no understanding. Error analysis is not included. The results are entirely meaningless. | 3 |  |
| Student is regular to lab performed the experiment. The procedure is well designed. Data presented is relevant and provides understanding. However Error analysis is not included. The results are entirely meaningless. | 4 |  |
| Student is regular and meticulously working on the lab experiments. Questions and hypothesis are stated clearly. The procedure is well designed. Data are presented in ways that best facilitate understanding and interpretation. Error analysis is included. The results are fully interpreted. | 5 |  |
| **(Regular Evaluation)**  **(Record Book – 5 M)** | Student has attended less than 50% of the lab sessions and not submitted record. | 0 |  |
| Student has attended less than 50% of the lab sessions and acquired 50% of the record submission marks. | 1 |  |
| Student has attended regularly but acquired less than 50% of the record submission marks. | 2 |  |
| Student has attended regularly but acquired 50-70% of the record submission marks. | 3 |  |
| Student has attended regularly but acquired 70-90% of the record submission marks. | 4 |  |
| Student has attended regularly and acquired 90-100% of the record submission marks. | 5 |  |