S. B. JAIN INSTITUTE OF TECHNOLOGY,

MANAGEMENT & RESEARCH, NAGPUR.

Post Lab

AIM: Develop a C++ (small subset implementing using class keyword) to C preprocessor using LEX and YACC tools.

Name of Student: Shrutika Pradeep Bagdi

Roll No.: CS22130

Semester/Year: 6th Semester/3rd Year

Academic Session: 2024-25

Date of Performance:

Date of Submission:

AIM: Develop a C++ (small subset implementing using class keyword) to C preprocessor using LEX and YACC tools.

OBJECTIVE / EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- Understand the basics of lexical analysis and syntax analysis.
- Learn how to use LEX and YACC tools for compiler design.
- Convert a subset of C++ (using the `class` keyword) into equivalent C code.
- Gain hands-on experience in implementing a basic preprocessor.

HARDWARE AND SOFTWARE REQUIRMENTS:

Hardware Requirement:

- Processor: Dual Core
- RAM: 1GB
- Hard Disk Drive: > 80 GB

THEORY:

A preprocessor is a tool that translates high-level language constructs into lower-level representations before compilation. The goal of this practical is to implement a simple preprocessor that converts a small subset of C++ (mainly handling `class` declarations) into equivalent C code.
Key Concepts:

1. **Role of LEX and YACC:**
   - LEX is used for tokenizing input text based on patterns.
   - YACC is used for parsing and translating the input text according to grammar rules.
2. **Handling C++ Classes:**
   - Convert class definitions into C-style structures.
   - Convert member functions into equivalent function definitions.
   - Manage object instantiation in C format.

ALGORITHM / PROCEDURE:

1. Define LEX rules to identify C++ keywords (`class`, `public`, `private`, etc.), identifiers, and punctuation.
2. Define YACC grammar to parse class definitions and generate corresponding C code.
3. Implement actions in YACC to transform class-based C++ code into struct-based C code.
4. Compile and test the preprocessor on sample C++ programs.

CODE:

vi PostLab.l

csc15@linux-p2-1272il: ~/CS22130

```
%{
#include "y.tab.h"
#include <stdio.h>
#include <string.h>
%}

%%

"class"             return CLASS;
"public"            return PUBLIC;
"private"           return PRIVATE;
"protected"         return PROTECTED;
"int"               return INT;
"float"             return FLOAT;
"char"              return CHAR;
[a-zA-Z_][a-zA-Z0-9_]*    { yylval.str = strdup(yytext); return IDENTIFIER; }
"{"                 return LBRACE;
"}"                 return RBRACE;
";"                 return SEMICOLON;
[ \t\n]             ;     // Ignore whitespaces and new lines
.                   return yytext[0];

%%

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

vi PostLab.y

csc15@linux-p2-1272il: ~/CS22130

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void yyerror(const char *s);
extern int yylex();
%}

%union {
    char* str;
}

%token <str> CLASS PUBLIC PRIVATE PROTECTED IDENTIFIER INT FLOAT CHAR LBRACE RBRACE SEMICOLON
%type <str> class_decl members member type

%%

program:
    class_decl
    ;

class_decl:
    CLASS IDENTIFIER LBRACE members RBRACE SEMICOLON {
        printf("/* Converted C struct */\n");
        printf("typedef struct %s {\n", $2);
        printf("%s} %s;\n", $4, $2);
    }
    ;
```

```
members:
    /* empty */ { $$ = strdup(""); }
    | members member {
        char *temp = malloc(strlen($1) + strlen($2) + 10);
        sprintf(temp, "%s%s", $1, $2);
        $$ = temp;
    }
    ;

member:
    type IDENTIFIER SEMICOLON {
        char *temp = malloc(strlen($1) + strlen($2) + 10);
        sprintf(temp, "    %s %s;\n", $1, $2);
        $$ = temp;
    }
    ;

type:
    INT    { $$ = "int"; }
    | FLOAT { $$ = "float"; }
    | CHAR  { $$ = "char"; }
    ;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main() {
    printf("Enter C++ class definition:\n");
    yyparse();
    return 0;
}
```

OUTPUT:

```
csc15@linux-p2-1272il:~/CS22130$ vi PostLab.l
csc15@linux-p2-1272il:~/CS22130$ vi PostLab.y
csc15@linux-p2-1272il:~/CS22130$ yacc -d PostLab.y
csc15@linux-p2-1272il:~/CS22130$ lex PostLab.l
csc15@linux-p2-1272il:~/CS22130$ cc lex.yy.c y.tab.c
csc15@linux-p2-1272il:~/CS22130$ ./a.out
Enter C++ class definition:
class Car {
    int speed;
    float mileage;
    char model;
};
/* Converted C struct */
typedef struct Car {
    int speed;
    float mileage;
    char model;
} Car;
```

CONCLUSION: I have successfully implemented a basic C++ to C preprocessor using LEX and YACC.

DISCUSSION AND VIVA VOCE:

1. What is the role of LEX and YACC in compiler design?
2. How does LEX handle pattern matching?
3. What are the limitations of this approach in handling full C++ programs?

REFERENCE:

- Compiler Design by O.G. Kakde, Laxmi Publications, 2006.
- Lab Manual of Compiler Design (Institute of Aeronautical Engineering, Dundigal, Hyderabad).