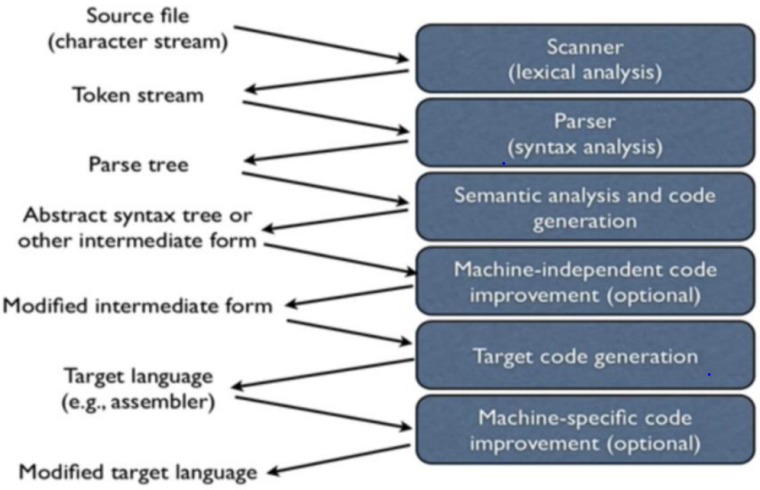
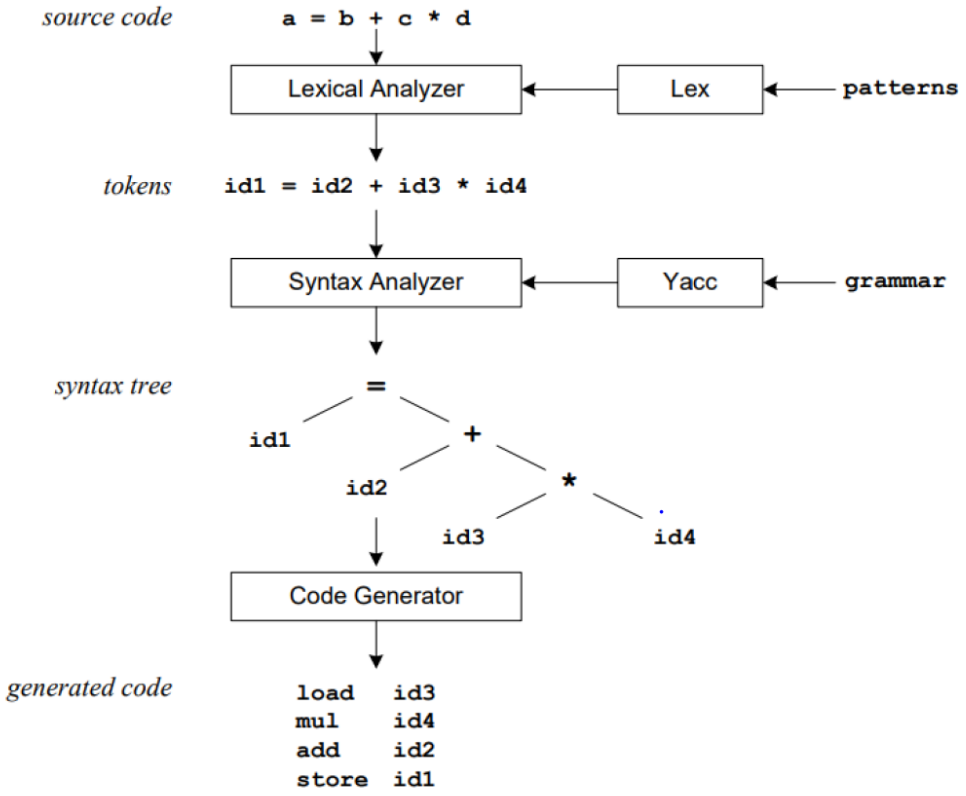
Language Processing:



Yacc: Yet Another Compiler Compiler

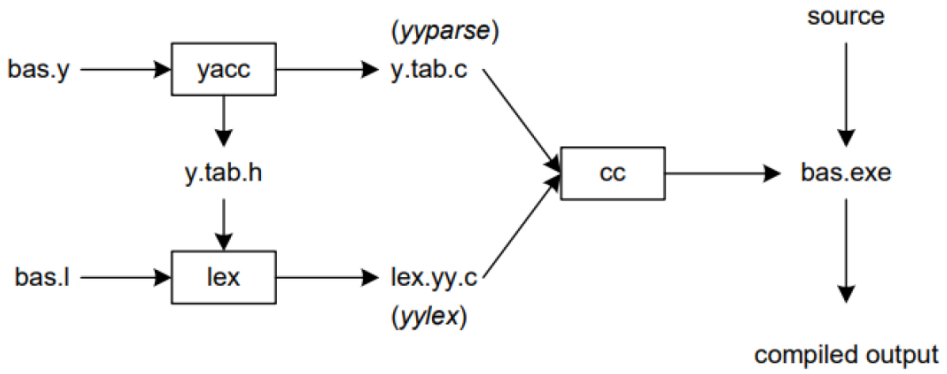
Parses and does semantic processing on stream of tokens produced by lex.



Installation:

1) `$ sudo apt-get install bison`
2) `$ sudo apt-get upgrade`

Using Yacc & Lex:



1) `$ yacc -d fileName.y` `Will create y.tab.h & y.tab.c`
2) `$ lex file.l` `Will create lex.yy.c`
3) `$ gcc lex.yy.c y.tab.c -ll`
4) `$ ./a.out`

–d causes yacc to generate definitions for tokens and place them in file **y.tab.h**.

Structure of Yacc file:

Part 1: Declarations:

- C declarations enclosed in %{ %}. Example:

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

- Yacc definitions:
  - **%start** StartingSymbol. First rule in the grammer.
  - **%union { char*** type1**; typeEnum** type2**; };**
  - **%token** terminalName or **%token <type1>** terminalName
  - **%type <type2>** nonTerminalName

Part 2: Grammar:

- Productions. Example:

  statement: **id** plusOperator **id** { **$$** = **$1** + **$3**; printf("Addition\n"); }
- Every symbol has a value associated with it (terminals and non-terminals).
  $$ is the value of the **left**.
  $1, **$2** refer to values associated with **symbols**.
- Handling error:

  statement: **error** SEMICOLON {printf("Syntax Error at line %d",**yylineno**); }

Note: Need to have the following at the top of the lex file to use yylineno (Line number counter): %option yylineno

Part 3: Programs:

- ```
  void yyerror(char *s) {
    printf("Parsing Error  ");
  }
  ```
- ```
  int main (void)
  {
     yyparse( );
     return 0;
  }
  ```

Code Sample 1:

Write a **compiler** that acts as a **calculator** (calculates the final result). **Associativity** and **precedence** must be handled. Example:

- 2 * (5 + 3)

test1.l:

```
%{
 #include <stdio.h>
 #include <stdlib.h>
 #include <stdarg.h>
 #include <string.h>
 void yyerror(char *);
 #include "y.tab.h"
%}

%%

[0-9]+  {yylval = atoi(yytext); return VALUE_INT;} //default return of yylex is int

\+      return OPERATOR_PLUS;
\-      return OPERATOR_MINUS;
\*      return OPERATOR_MULTIPLY;
\/      return OPERATOR_DIVIDE;
\(      return ARGUMENT_OPENBRACKET;
\)      return ARGUMENT_CLOSEBRACKET;

[ \t]   ;
\n      ;

.       yyerror("invalid character");

%%

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

test1.y

```
%{
 #include <stdio.h>
 #include <stdlib.h>
 #include <stdarg.h>
 #include <string.h>
 FILE * yyin;
 void yyerror(char *);
 int yylex(void);
%}

%token VALUE_INT OPERATOR_PLUS OPERATOR_MINUS OPERATOR_MULTIPLY OPERATOR_DIVIDE ARGUMENT_OPENBRACKET ARGUMENT_CLOSEBRACKET
//Precedence and left Associativity
%left OPERATOR_PLUS OPERATOR_MINUS
%left OPERATOR_MULTIPLY OPERATOR_DIVIDE

%%

program : program statement {printf("program: program statement\n\n");}
        |                   {printf("program: Epsilon\n\n");}
        ;

statement : expr            {printf("statement: expr(%d)\n", $1);}
          ;

expr:   VALUE_INT                     {printf("expr: VALUE_INT(%d)\n",$1); $$=$1;}
        | expr OPERATOR_PLUS expr     {printf("expr: expr(%d) OPERATOR_PLUS expr(%d)\n",$1,$3); $$=$1+$3;}
        | expr OPERATOR_MINUS expr    {printf("expr: expr(%d) OPERATOR_MINUS expr(%d)\n",$1,$3); $$=$1-$3;}
        | expr OPERATOR_MULTIPLY expr {printf("expr: expr(%d) OPERATOR_MULTIPLY expr(%d)\n",$1,$3); $$=$1*$3;}
        | expr OPERATOR_DIVIDE expr   {printf("expr: expr(%d) OPERATOR_DIVIDE expr(%d)\n",$1,$3); $$=$1/$3;}
        | ARGUMENT_OPENBRACKET expr ARGUMENT_CLOSEBRACKET
            {printf("expr: ARGUMENT_OPENBRACKET expr(%d) ARGUMENT_CLOSEBRACKET\n",$2); $$=$2;}
        ;

%%

void yyerror(char *s) {
 printf("%s\n", s);
}

int main (void)
{
        yyin  = fopen("testfile.txt","r+");
        if(yyin ==NULL)
        {
                printf("File Not Found\n");
        }
        else
        {
                printf(">>>>  Test File  <<<<\n\n");
                FILE *testFile; char ch;
                testFile = fopen("testfile.txt", "r");
                while((ch = fgetc(testFile)) != EOF)
                        printf("%c", ch);

                printf("\n\n\n>>>>  Parsing  <<<<\n\n");
                yyparse();
        }
        fclose(yyin);
        return 0;
}
```