|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Course**  **Code** | **Course Name** | **Teaching Scheme**  **(Hrs/week)** | | | **Credits Assigned** | | | |
| **L** | **T** | **P** | **L** | **T** | **P** | **Total** |
| CPCL601 | **System Programming and**  **Compiler Construction Lab** | **--** | **--** | **2** | **--** | **--** | **1** | **1** |
| **Examination Scheme** | | | | | | |
| **ISE** | | **ESE** | | | | **Total** |
| **Practical** | | **Oral** | |
| **40** | | **20** | | **-** | | **60** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Pre-requisite Course Codes** | | CPC601 (System Programming and Compiler Construction) | |
| At end of successful completion of this course, student will be able to | | | |
| **Course Outcomes** | CO1 | | To use Flex or similar tools to create a lexical analyzer and  Yacc/Bison tools to create a parser. |
| CO2 | | To implement different types of handwritten parsers. |
| CO3 | | To implement the working of assembler and Macros. |
| CO4 | | To demonstrate linkers and loaders. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Exp. No.** | **Experiment Details** | **Ref.** | **Marks** |
| **1** | Lex program to generate tokens (identifiers, keywords, operators,  delimiters, etc.) for Java Programming language. Program should generate at least 50 tokens and it should recognize unique identifiers. | 1 | **5** |
| **2** | Program to remove left recursion for the given grammar. Program  should accept the grammar from user, detect left recursion and eliminate it by generating a new non-terminal. | 3,4 | **5** |
| **3** | Implement Predictive Parser for the given grammar. | 3,4 | **5** |
| **4** | Program to find first and follow sets for the given grammar. Program  should accept the grammar from user and output the first and follow sets for each of the grammar symbol. | 3.4 | **5** |
| **5** | Program to generate Quadruple table for the given postfix String | 3,4 | **5** |
| **6** | Implement two pass Assembler for IBM 360/370 machine. The input is  a source consisting of syntactically correct IBM 360/370 statement.  The output should display all tables and their values. The final output is an object file of small subset of instructions. | 2 | **5** |
| **7** | Implement MACRO Assembler for IBM 360/370 for the feature  “Conditional MACRO expansion”. | 2 | **5** |
| **8** | Create user defined library in open source environment and use it for |  | **5** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | particular functions. |  |  |
| **Total Marks** | | | **40** |

**References:**

[1] John R. Levine, Tony Mason & Doug Brown O‟Reilly, “*lex & yacc*”, 2nd Edition

[2] J. J Donovan, “Systems Programming” Tata McGraw Hill Publications.

[3] A. V. Aho, R. Shethi and J.D. Ulman, “Compilers Principles, Techniques and Tools” ,

Pearson Education

[4] A. V. Aho, R. Shethi, Monica Lam , J.D. Ulman, “Compilers Principles, Techniques and

Tools”, Pearson Education, Second Edition.

Experiment – I

|  |  |  |  |
| --- | --- | --- | --- |
| **Aim** | Lex program to generate tokens (identifiers, keywords, operators,  delimiters, etc.) for Java Programming language. Program should generate at least 50 tokens and it should recognize unique identifiers. | No of turns = 2 | **Marks5** |

**Theory:**

What is LEX ?

General Structure of LEX program ?

How to compile ? Explain with example.

Draw supportive diagrams.

Write about different category of tokens available in java ?

**Implementation:**

1. **Write .l ( lexx file )**
2. **Display output.**

**Conclusion:**

**Reference Material:**

**[1] John R. Levine, Tony Mason & Doug Brown O‟Reilly, “*lex & yacc*”, 2nd Edition**

Experiment – II

|  |  |  |  |
| --- | --- | --- | --- |
| **Aim** | Program to remove left recursion for the given grammar. Program  should accept the grammar from user, detect left recursion and eliminate it by generating a new non-terminal. | No of turns = 2 | **Marks5** |

**Theory:** A grammar is left recursive if for a non-terminal A, there is a derivation A⇒+ Aα

There are three types of left recursion:

direct (A → A x)

indirect (A → B C, B → A )

hidden (A → B A, B →ε)

To eliminate direct left recursion replace

A → Aα1 | Aα2 | ... | Aαm | β1 | β2 | ... | βn

with

A →β1B | β2B | ... | βnB

B →α1B | α2B | ... | αmB | ε

**Implementation :**

**1.Input : Production rules.**

**2.Output: Set of production rules from which recursion is removed.**

**Reference:**

**[3] A. V. Aho, R. Shethi and J.D. Ulman, “Compilers Principles, Techniques and Tools”,**

**Pearson Education.**

Indirect Recursion ellimination

S → E  
E → E+T

E → T

T → E-T

T →id

Algorithm for eliminating indirect recursion

List the nonterminals in some order A1, A2, ...,An

for i=1 to n

{ for j=1 to i-1

if there is a production A**i**→A**jα,**

replace A**j**with its rhs

}

eliminate any direct left recursion on A**i**

**Implementation:**

1. **Write .l ( lexx file )**
2. **Display output.**

**Conclusion:**

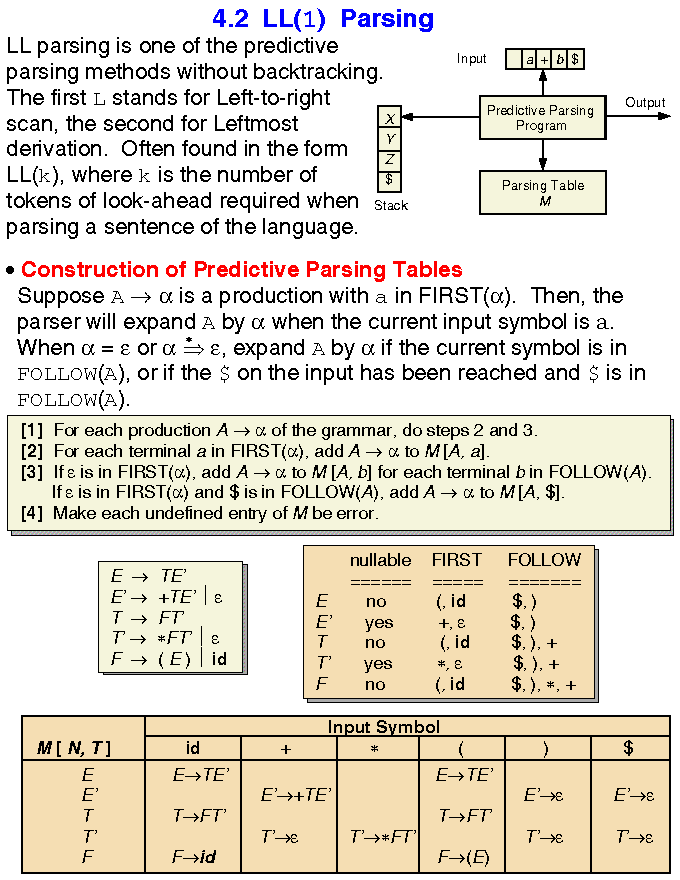
**Reference Material:**

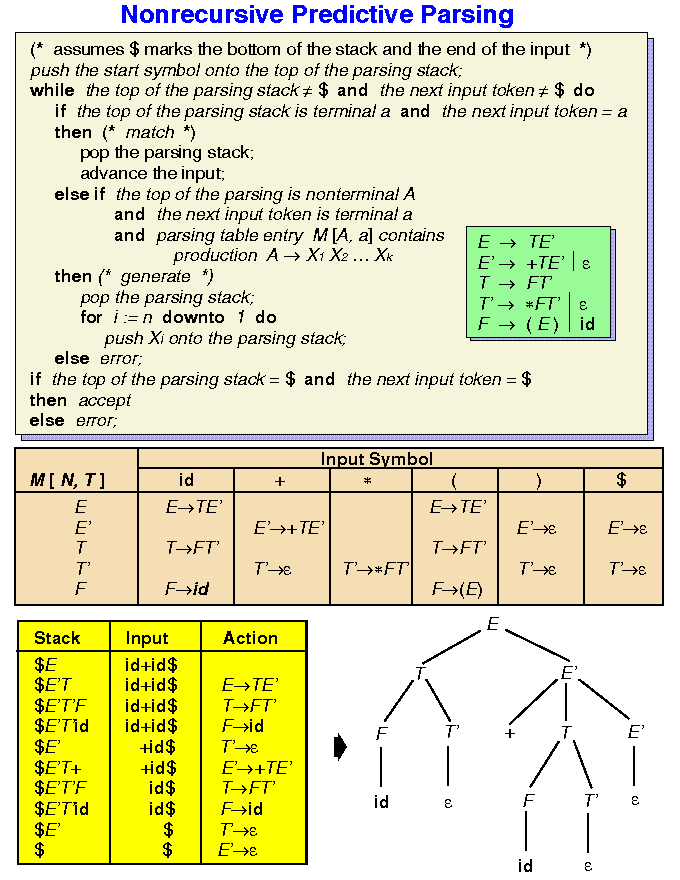
**[1] John R. Levine, Tony Mason & Doug Brown O‟Reilly, “*lex & yacc*”, 2nd Edition**

Experiment – III

|  |  |  |  |
| --- | --- | --- | --- |
| **Aim** | Implement Predictive Parser for the given grammar. | No of turns = 2 | **Marks5** |

**Theory:**





**Implementation: 1. Fixed the input grammar**

**2. Fixed the parser table.**

**3. Parse the given string.**

**Conclusion:**

**Reference:**

**[3] A. V. Aho, R. Shethi and J.D. Ulman, “Compilers Principles, Techniques and Tools”,**

**Pearson Education.**

Experiment – IV

|  |  |  |  |
| --- | --- | --- | --- |
| **Aim** | Program to find first and follow sets for the given grammar. Program should accept the grammar from user and output the first and follow sets for each of the grammar symbol. | No of turns = 2 | **Marks5** |

**Theory:**

## Rules for FIRST Set

1. If X is a terminal **then** First(X) is just X!
2. If there is a Production X → ε **then** add ε to first(X)
3. If there is a Production X → Y1Y2..Yk **then** add first(Y1Y2..Yk) to first(X)
4. First(Y1Y2..Yk) is **either**
   1. First(Y1) (if First(Y1) doesn't contain ε)
   2. **OR** (if First(Y1) does contain ε) then First (Y1Y2..Yk) is everything in First(Y1) <except for ε > as well as everything in First(Y2..Yk)
   3. If First(Y1) First(Y2)..First(Yk) all contain ε **then** add ε to First(Y1Y2..Yk) as well.

***Rules for FOLLOW Set***

1. First put $ (the end of input marker) in Follow(S) (S is the start symbol)
2. If there is a production A → aBb, (where a can be a whole string) **then** everything in FIRST(b) except for ε is placed in FOLLOW(B).
3. If there is a production A → aB, **then** everything in FOLLOW(A) is in FOLLOW(B)
4. If there is a production A → aBb, where FIRST(b) contains ε, **then** everything in FOLLOW(A) is in FOLLOW(B).

**IMPLEMENTATION:**

1. Program should accept Set of Production rules as input.
2. Capital letters should be treated as Non terminals.
3. Small letters should be treated as terminals.
4. ε should be represented by ‘n’.
5. O/p should be in the form of Table with two columns. – Symbol and FIRSTSET. And Symbol and FOLLOWSET.

**Conclusion:**

**Reference: [3] A. V. Aho, R. Shethi and J.D. Ulman, “Compilers Principles, Techniques and Tools”, Pearson Education.**

Experiment – V

|  |  |  |  |
| --- | --- | --- | --- |
| **Aim** | Program to generate Quadruple table for the given postfix String | No of turns = 2 | **Marks5** |

**Theory:**

What are the different phases of compiler ? Along with example.

State importance of ICG. and different formats.

Draw supportive diagrams.

**Implementation:**

**Conclusion:**

Experiment – VI

|  |  |  |  |
| --- | --- | --- | --- |
| **Aim** | Implement two pass Assembler for IBM 360/370 machine. The input is a source consisting of syntactically correct IBM 360/370 statement. The output should display all tables and their values.  The final output is an object file of small subset of instructions. | No of turns = 2 | **Marks5** |

**Theory:**

Assembler is a program that accepts as input an assembly language program and produces equivalent along with information to loader. An assembly language should support features like macro-processing, file inclusion etc.

It uses two passes :

Pass 1 : Defines the Symbols

Pass 2: Generates the machine code.

Two passes are required because of forward reference problem.

**Implementation:**

**Conclusion:**

Experiment – VII

|  |  |  |  |
| --- | --- | --- | --- |
| **Aim** | Implement MACRO Assembler for IBM 360/370 | No of turns = 2 | **Marks5** |

**Theory:**

Assembler is a program that accepts as input an assembly language program and produces equivalent along with information to loader. An assembly language should support features like macro-processing, file inclusion etc.

It uses two passes :

Pass 1 : Defines the Symbols

Pass 2: Generates the machine code.

Two passes are required because of forward reference problem.

**Implementation:**

1. **Assume the program containing MACRO is stored in ‘MACRO.ASM’**
2. **process MACRO , display MNT , MDT , ALA and**

**Expanded Source code.**

**Conclusion:**

**Reference Material:** [**2] J. J Donovan, “Systems Programming” Tata McGraw Hill Publications.**

Experiment – VIII

|  |  |  |  |
| --- | --- | --- | --- |
| **Aim** | Create user defined library in open source environment and use it for particular functions. | No of turns = 2 | **Marks5** |

**Theory:**

**Implementation:**

**Conclusion:**