**Alunos:**
**Tiago Schelp Lopes**
**Renan Martins Zomignani Mendes**

**Parte 1:**

**Analisador Lexico:**

O analisador lexico está implementado nos arquivos lex.h e lex.c.

Em lex.h é definida a estrutura do token, com dois valores: class é o código do tipo do token e repr é o caracter.
Obs.: Nessa linguagem, os lexemas possuem apenas um caracter.

Em Lex.c a função get_next_token lê um caracter digitado e verifica se é EOF, algarismo(entre 0 e 9) ou algo diferente. Descartando os caracteres espaço(` `), tab(`/t`) e enter(/n).

**Analisador Sintático:**

Implementado nos arquivos:
Parser.h
Parser.i
Parserbody.i
Parser.c

Parser.h
É definida a estrutura Expression que é a representação de uma árvore sintática abstrata (em inglês, AST, Abstract Syntax Tree). Todo nó possui referências para um elemento à esquerda(left), direita (right) e um operador (oper).

Parser.i
A função Parse_program verifica se a expressão de entrada está correta, utilizando para essa verificação a função descrita no Parsebody.h

Parserbody.h
A função Parse_expression verifica a estrutura de entrada, cada expressão deve iniciar com parênteses '(' seguido de DIGIT, operador, DIGIT e terminar com parênteses ')' . Representando DIGIT por 'D' e os operadores por 'O', a expressão regular aceita pelo compilador é a seguinte:

D = ( D O D )

**Tradutor**

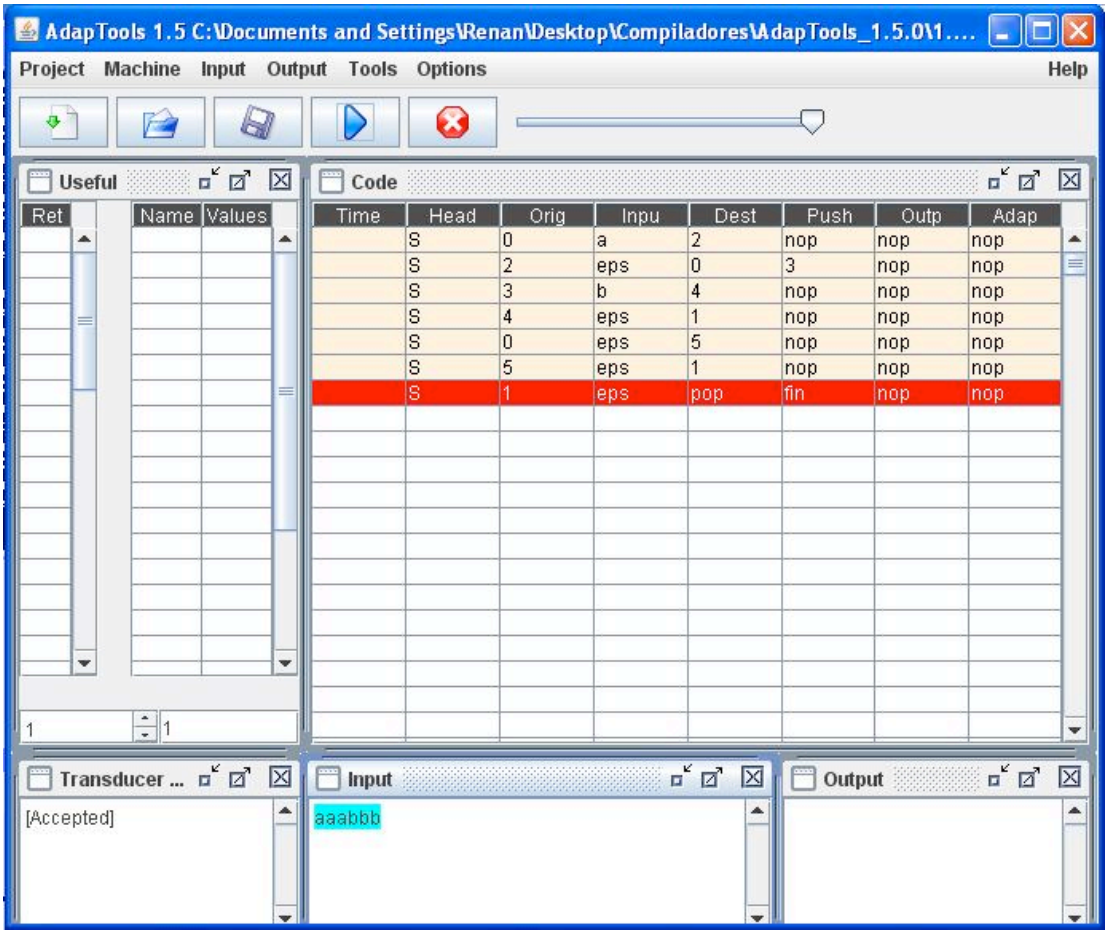
Em main.c é chamada a função Process.
Essa função é definida em codegen.c, interpret.c e em itinter.c

Em cada dessas descrições a transcrição da expressão gera uma saída em formato diferente.
Exemplo:
Para a entrada: (2*2)
Saída para codegen.c:
PUSH 2
PUSH 2
MULT

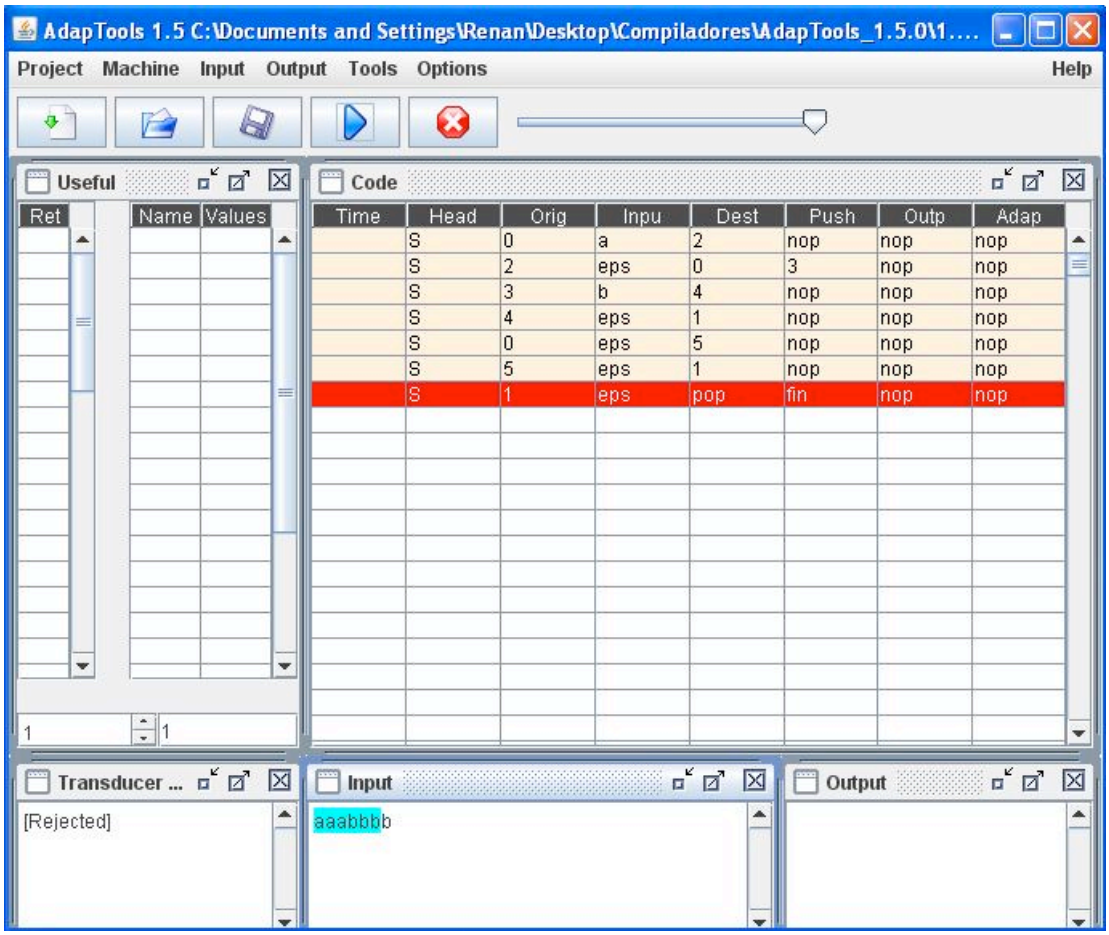
Saída para interor.c:
2*2

Saída para itinter.c:
Push( 2 * 2 )

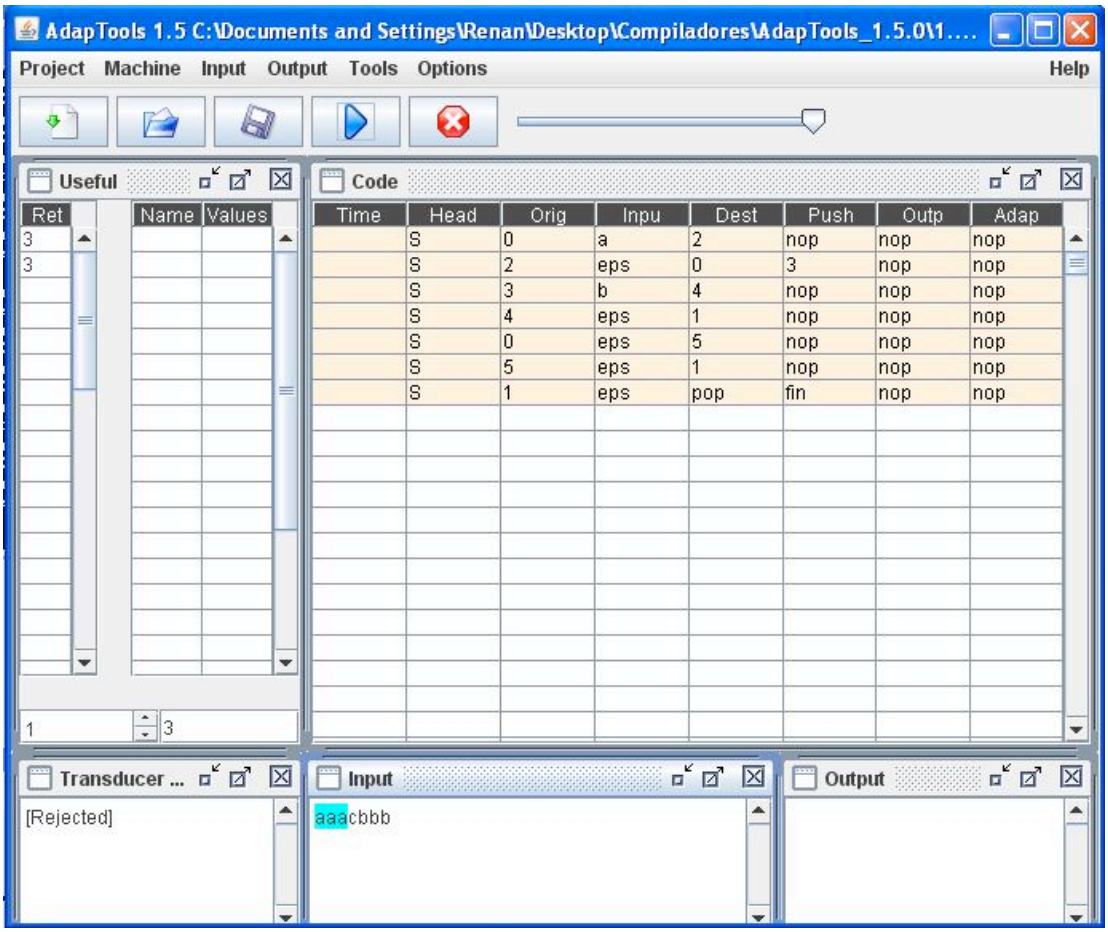
**Parte 2 e 3:**
Utilizamos o projeto Sintatico.prj com o input na forma $a^n b^n$, o resultado é apresentado abaixo:



Resultado para teste com “aaabbb“ - Resultado: “Accepted“



Resultado para “aaabbbb” - Resultado: “Rejected“



Resultado para “aaacbbb” – Resultado: “Rejected“