

# INF1022 – ANALISADORES LÉXICOS E SINTÁTICOS

## PROJETO FINAL

Professor: Edward Haeusler

Monitor: Bernardo Alkmim

Integrantes: Daniel Stuberg Huf (1920468) e Eduardo Sardenberg Tavares (1911328)

### 1. Objetivo

O projeto final da disciplina consistiu em usar o conjunto de ferramentas Flex/Bison (Flex/Yacc) para geração de compiladores com o objetivo de gerar código de uma linguagem imperativa, denominada Provol-One, em uma linguagem da escolha dos integrantes da dupla. No caso, foi escolhida a linguagem C, ou seja, o compilador de Provol-One irá gerar código objeto em C.

### 2. Desenvolvimento

Inicialmente, dada a gramática para a linguagem Provol-One exibida no enunciado do projeto, foram realizadas pequenas modificações para adequar melhor a elaboração do código sobre a gramática. Em seguida, foram incluídos novos comandos na gramática com o intuito de deixá-la mais rebuscada. As alterações estão descritas a seguir.

- Para a primeira regra original da gramática, foi adicionado após o último símbolo não terminal um novo último símbolo não terminal denominado *NEWLINE*, que irá ler a quebra de linha, de modo que o input do código na linguagem Provol-One deverá ser seguido de um pressionamento da tecla enter no teclado. O motivo por essa escolha vem da própria definição do que o padrão POSIX entende como linha: “A *sequence of zero or more non-  
<newline> characters plus a terminating <newline> character.*”
- Como o código gerado está na linguagem C, faz sentido pensar em único retorno de variável da função. Portanto, o segundo *varlist* da primeira regra foi substituído por um novo símbolo não terminal *ret*. Dessa forma, *varlist* pode derivar múltiplas entradas da função, enquanto que *ret* deriva apenas uma saída.
- Para a segunda e quinta regras da gramática, a recursão à direita foi substituída pela recursão à esquerda. Além de economizar espaço na pilha de execução, a troca da recursão facilitou a construção do código atributo associado ao símbolo não terminal *varlist* (segunda regra) e *cmds* (quinta regra).
- Foram incluídos comandos de seleção (if-then e if-then-else) e comando de repetição definida (Faça < *cmds* > X vezes). Para tais comandos, foi acrescentado ao final de cada uma de suas expressões o símbolo não terminal *FIM* com o intuito de não gerar ambiguidade.

Assim, a gramática final ficou da seguinte forma:

```
program → ENTRADA varlist SAIDA ret cmds END NEWLINE  
varlist → varlist id | id  
ret → id  
cmds → cmds cmd | cmd  
cmd → ENQUANTO id FACA cmds FIM  
cmd → id = id | INC(id) | ZERA(id)  
cmd → SE id ENTAO cmds FIM | SE id ENTAO cmds SENAO cmds FIM  
cmd → FACA id VEZES cmds FIM
```

### 3. Implementação

Primeiramente, foi desenvolvida a ferramenta do **lex**, encarregada de realizar a análise sintática. A ferramenta do lex necessita da especificação da linguagem na forma de expressão regular e de um conjunto de funções adicionais para a manipulação dos tokens gerados, em particular de funções que trabalhem com a tabela de símbolos. No preâmbulo do lex, foram incluídas apenas as bibliotecas necessárias para execução do código. No corpo, foram especificadas as expressões regulares para a linguagem e também as ações que decorrem do reconhecimento de cada token que foi definido. Ao final do arquivo, não foi incluída nenhuma função adicional.

Em seguida, foi desenvolvida a ferramenta do **yacc**, responsável pela geração do parser e por trabalhar sobre as funções geradas pelo lex para ler o input a ser compilado. Na parte de definição, foram incluídas as bibliotecas necessárias para execução do código, a função `yylex`, que reconhece os tokens do input e os retorna para o parser, a variável global `nParams`, que conta o número de parâmetros passados e, por fim, uma função para tratamento de erros sintáticos. No corpo do yacc, foi definida uma union, que alterna os tipos para o `yylval`, e foram declarados os tokens e símbolos não terminais a serem lidos, bem como seus respectivos tipos.

Também no corpo, está presente a gramática de atributos para o yacc, responsável pela manipulação semântica de contexto. Para cada regra da gramática, está associada a ela uma sequência de comandos, que se resume a declarar uma string e alocar dinamicamente para ela o tamanho equivalente ao código em C que será gerado a partir da regra, enviar o output do código em C para a string criada através da função `sprintf`, e por fim fazer o símbolo não terminal mais à esquerda da regra receber a mesma string, agora já inicializada.

O código yacc foi construído de maneira que, a partir do input lido em Provol-One, seja gerada uma função nos moldes da linguagem C. Portanto, o que a main do yacc faz é simplesmente completar o código em C para que ele fique 100% funcional.

Assim, primeiro é printado (ou melhor, redirecionado) no arquivo de saída a inclusão das bibliotecas, depois ocorre a análise sintática e consequente criação da função, e finalmente é printada a main do arquivo de saída, que printa o retorno da função criada pelo parser e que tem como argumentos os valores passados no terminal quando o executável é chamado.

#### 4. Execução

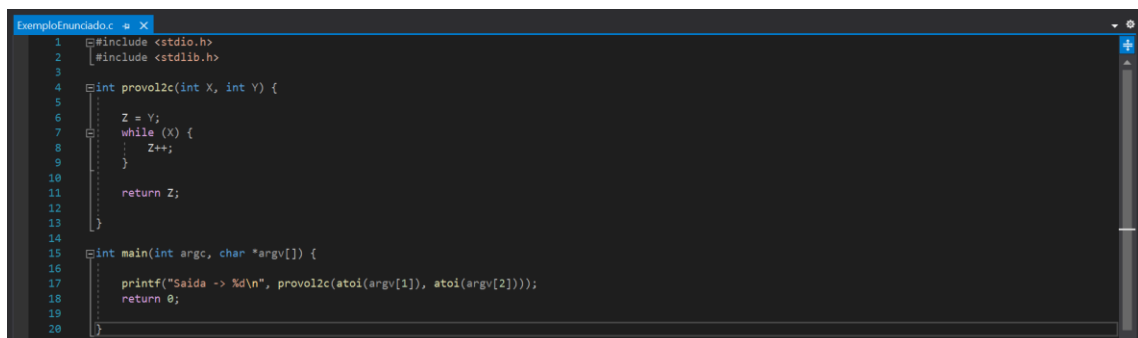
Para operar corretamente o parser e gerar código objeto em C, deve-se descompactar o arquivo zip e extrair os arquivos contidos nele para um mesmo diretório de sua escolha. Em seguida, execute a seguinte sequência de comandos no terminal do Linux:

```
yacc -d ProvolScanner.y
lex ProvolScanner.l
gcc -c lex.yy.c y.tab.c
gcc -o parser lex.yy.o y.tab.o -ll
./parser < xxxx.provol > yyyy.c
gcc -Wall -o zzzz yyyy.c
./zzzz arg1 arg2 argN
```

Onde **xxxx** é o nome do arquivo que contém o código em Provol-One, previamente criado pelo usuário, a ser lido pelo parser, **yyyy** é o nome do arquivo que contém o código em C a ser gerado pelo parser, **zzzz** é o arquivo executável que conterá o programa, e **arg** é cada argumento inteiro não negativo que o usuário desejar passar para a função, quantas vezes tiver sido definido um parâmetro de entrada para a mesma no código em Provol-One.

#### 5. Exemplos

No arquivo zip que contém o projeto, foram deixados alguns arquivos .provol contendo programas de exemplo, que podem ser utilizados no passo a passo anterior para gerarem código objeto em C. Os códigos gerados por tais arquivos estão exibidos a seguir.



```
ExemploGerado.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int provol2c(int X, int Y) {
5
6     Z = Y;
7     while (X) {
8         Z++;
9     }
10
11     return Z;
12 }
13
14
15 int main(int argc, char *argv[]) {
16
17     printf("Saída -> %d\n", provol2c(atoi(argv[1]), atoi(argv[2])));
18     return 0;
19 }
20
```

```
Incrementalvezes.c  X
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int provol2c(int A, int B) {
5     for (int i=0; i<B; i++) {
6         A++;
7     }
8     return A;
9 }
10
11
12
13
14 int main(int argc, char *argv[]) {
15     printf("Saida -> %d\n", provol2c(atoi(argv[1]), atoi(argv[2])));
16     return 0;
17 }
18
19 }
```

```
GuardaZeraRetorna.c  X
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int provol2c(int A, int B) {
5     if (A) {
6         B = A;
7         A = 0;
8     }
9     return B;
10 }
11
12
13
14
15 int main(int argc, char *argv[]) {
16     printf("Saida -> %d\n", provol2c(atoi(argv[1]), atoi(argv[2])));
17     return 0;
18 }
19
20 }
```

```
Soma3args.c  X
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int provol2c(int A, int B, int C) {
5     for (int i=0; i<B; i++) {
6         A++;
7     }
8     for (int i=0; i<C; i++) {
9         A++;
10     }
11     return A;
12 }
13
14
15
16
17 int main(int argc, char *argv[]) {
18     printf("Saida -> %d\n", provol2c(atoi(argv[1]), atoi(argv[2]), atoi(argv[3])));
19     return 0;
20 }
21
22 }
```

```
Soma3argsPositivos.c  X
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int provol2c(int A, int B, int C) {
5     if (A) {
6         if (B) {
7             if (C) {
8                 for (int i=0; i<A; i++) {
9                     C++;
10                 }
11             }
12             for (int i=0; i<B; i++) {
13                 C++;
14             }
15         }
16         else {
17             C = 0;
18         }
19     }
20     else {
21         C = 0;
22     }
23 }
24
25
26
27
28 return C;
29 }
30
31
32 int main(int argc, char *argv[]) {
33     printf("Saida -> %d\n", provol2c(atoi(argv[1]), atoi(argv[2]), atoi(argv[3])));
34     return 0;
35 }
36
37 }
```