Universidade Estadual de Santa Cruz – UESC

Projeto 4D

Discente: Levy Santiago

Matrícula: 201520138

Disciplina Compiladores.

Curso Ciência da Computação

Semestre 2018.2

Professor César Bravo

Ilhéus – BA
2018

Índice

1. Introdução

Neste relatório é descrita a implementação de um compilador simples, o qual visa transformar um código fonte em instruções p-code. Nesta seção serão apontados o objetivo de se implementar um compilador, também será detalhada a linguagem de programação aceita por este. Os módulos desenvolvidos para o funcionamento do compilador foram: Parser, o qual analisa uma programa de entrada e determina uma estrutura segundo uma gramática; ADA2ASA que transforma a árvore de análise descendente (ADA) em árvore de sintaxe abstrata (ASA); ASA2NP* que transforma a ASA em notação polonesa (NP) e/ou NP reversa; NP2GCI que transforma NP* em código intermediário (CI) e GCI2GCO que transforma o código intermediário em código objeto (CO). Todos os códigos fonte, arquivos de teste e saídas se encontram disponíveis no link apresentado na seção Referências.

1.1. Objetivo do Compilador

Oferecer a análise de programas, os quais seguem as regras da gramática apresentada no apêndice deste trabalho, e a transformação destes para o código objeto p-code, permitindo a execução desses programas em uma p-code-machine.

1.2. Linguagem de Programação

A estrutura da linguagem de programação para a qual o compilador está sendo desenvolvido é primeiramente a declaração de uma função, que pode ser a main (‘m’), uma outra função (‘h’ ou ‘g’), ou um conjunto de funções. A estrutura dessas funções é, por exemplo, T m () { STMT }, onde T é o tipo de dado de retorno (‘i’ para int ou ‘c’ para char) e STMT é uma concatenação de uma lista de declarações de variáveis, em seguida uma lista de atribuições, depois uma lista de controles e por fim a função de retorno.

A lista de declarações é uma sequência de declarações de variáveis. Cada declaração possui um tipo de dado seguido do nome de variável (‘j’, ‘k’, ‘l’, ‘n’, ‘p’, ‘q’, ‘s’, ‘t’, ‘u’, ‘v’, ‘x’, ‘y’ ou ‘z’) ou de uma conjunto de nomes de variáveis separados por vírgula, caso o programador queira declarar várias variáveis em uma mesma linha. O fim de todo conjunto de declarações é definido por um ponto-e-vírgula.

A lista de atribuições é uma sequência de atribuições, onde cada atribuição é iniciada com o nome de uma variável, em seguida um símbolo de igual (‘=’) e então uma expressão ou um caractere (‘a’, ‘b’, ‘d’ ou ‘e’). Uma expressão pode ser um número de 0 a 9, ou uma expressão aritmética organizada por parênteses (eg. (3 *

(2 + 1))) fazendo uso dos símbolos de operação binária: ‘+’, ‘-’, ‘*’ e ‘/’, e de operação unária: ‘~’. O fim de toda atribuição é separado por ponto-e-vírgula.

A lista de controles é a área onde são usadas estruturas de desvio e loops. Para representar a estrutura de desvio ‘IF’, é usado o símbolo ‘f’, e assim pode-se aplicar seguindo a estrutura f(B){LC}, onde ‘LC’ é uma lista de controles e ‘B’ é a condição que pode ser ‘T’ (para verdadeiro), ‘F’ (para falso) ou uma expressão lógica organizada com parênteses e usando símbolos lógicos como: ‘&’, ‘|’, ‘:’, ‘_’, ‘>’ ou ‘<’. Também pode ser usado o operador de negação ‘!’. A estrutura de repetição ‘FOR’ é representada pelo símbolo ‘o’, e para usá-la deve-se seguir a estrutura o(A ; B ; A) {LC}, onde ‘LC’ é uma lista de controles, ‘A’ é uma atribuição, ‘B’ é uma condição que segue as mesmas regras da condição do ‘IF’. A estrutura de repetição ‘WHILE’ é representada pelo símbolo ‘w’ e para fazer o uso desta, deve-se seguir a estrutura w(B){LC}, onde ‘LC’ é uma lista de controles e B é uma condição.

A função de retorno é representada por um ‘r’, e segue o padrão r(E);, onde ‘E’ é uma expressão.

2. Módulos do compilador

Esta seção apresenta uma descrição para cada módulo do compilador. Para aqueles que foram implementados é demonstrado como compilar o módulo individualmente e executá-lo dada uma entrada. Além disso, também é demonstrada uma forma de compilar todos os módulos de uma só vez.

2.1. Scanner (Analisador Léxico)

Este módulo realiza o processo de fazer uma varredura caractere por caractere do programa fonte submetido pelo usuário, traduzindo em símbolos léxicos chamados tokens. Neste módulo são reconhecidas todas as palavras reservadas, constantes, identificadores e qualquer outra palavra que pertença à linguagem adotada.

2.2. Parser (Analisador Sintático)

Este módulo realiza o processo de analisar uma cadeia de entrada para determinar sua estrutura gramatical segundo uma gramática formal. Basicamente o parser transforma um texto na entrada em uma estrutura de dados (eg. uma árvore). O parser está diretamente ligado ao scanner, o parser o ativa para obter os símbolos (tokens) e assim usar um conjunto de regras para construir a árvore sintática e a tabela de símbolos. Caso na análise sejam encontrados erros, a análise continua para verificar se existem outros e, por fim, são apresentadas mensagens de erro para o usuário. A saída deste módulo é a árvore de análise sintática em

formato de vetor com os símbolos e respectivos índices de cada símbolo, que representa os símbolos de entrada em uma estrutura de árvore, e a tabela de símbolos, que representa todos as variáveis com seus respectivos tipos de dados e valor.

Para compilar o módulo:

```
$ gcc pop.c isVariavel.c pushAda.c buscarProducao.c
adicionarTabSimb.c imprimirArvore.c quick_sort.c S.c parser.c
parsermain.c -o parser -w
```

para executar passando como entrada o programa a ser compilado:

```
$ ./parser < testeG1.txt
```

O código fonte da implementação deste módulo está no Apêndice B.

2.3. ADA2ASA (Árvore Sintática para Árvore Sintática Abstrata)

Este é o módulo responsável por transformar a árvore de análise descendente ou árvore sintática (ADA) na árvore sintática abstrata (ASA). São considerados todos os símbolos que são operandos e operadores, símbolos como parênteses e vírgulas são desconsiderados. A partir deste estágios, erros não são mais encontrados, pois todos os erros foram tratados no módulo anterior, portanto este módulo só precisa realizar a transformação. A saída deste módulo é a árvore sintática abstrata em forma de vetor com os símbolos e os respectivos índices de cada um, que representa todos os operandos e operadores em uma estrutura de dados que será submetida como entrada do próximo módulo.

Para compilar o módulo:

```
$ gcc pushAsa.c A.c indiceRealAda.c imprimirArvore.c
quick_sort.c ada2asa.c ada2asamain.c -o ada2asa -w
```

para executar passando como entrada a saída do módulo anterior:

```
$ ./ada2asa < ada.txt
```

O código fonte da implementação deste módulo está no Apêndice C.

2.4. ASA2NP* (Árvore Sintática Abstrata para Notação Polonesa)

Este módulo é responsável por transformar a árvore de sintaxe abstrata (ASA) em notação polonesa ou polonesa reversa (NP*). Convertendo a árvore para notação polonesa é realizada uma busca em pré-ordem na mesma. Caso a árvore seja transformada para notação polonesa reversa, ela é percorrida em pós-ordem.

Assim, a saída do módulo é a notação polonesa (reversa) representando a sequência de operações do programa a serem realizadas. As instruções de controle como IF e FOR são representadas em notação polonesa e as outras são representadas em notação polonesa reversa.

Para compilar o módulo:

```
$ gcc indiceRealAsa.c asaToPolonesaReversa.c asaToPolonesa.c
asa2np.c asa2npmain.c -o asa2np -w
```

para executar passando como entrada a saída do módulo anterior:

```
$ ./asa2np < asa.txt
```

O código fonte da implementação deste módulo está no Apêndice D.

2.5. NP2GCI (Gerador de Código Intermediário)

O GCI é o último módulo da camada de front end em que o código fonte é transitado. Este módulo traduz a saída do módulo anterior (notação polonesa padrão ou reversa) para um código intermediário. O uso de uma representação intermediária tem vantagens como reaproveitamento de código, o que facilita o transporte de um compilador para diversas plataformas de hardware, outra vantagem é que independente da máquina, um otimizador de código pode ser usado no código intermediário. A diferença básica entre o código intermediário e o código objeto é que no intermediário não são especificados detalhes da máquina alvo, como quais registradores estão sendo usados ou quais endereços de memória são referenciados. Assim, a saída deste módulo é o código intermediário reproduzindo em instruções as operações descritas na notação polonesa.

Para compilar o módulo:

```
$ gcc read.c write.c isOperador.c isOperadorLogicoBinario.c
geraPseudocodigo.c expressao.c expressaoLogica.c
leListaDeclaracoes.c leAtribuicao.c leListaAtribuicoes.c
controle.c np2gci.c np2gcimain.c -o np2gci -w
```

para executar passando como entrada a saída do módulo anterior:

```
$ ./np2gci < np.txt
```

O código fonte da implementação deste módulo está no Apêndice E.

2.6. GCI2GCO (Gerador de Código Objeto)

Este módulo é responsável por transformar o código intermediário em código objeto. O código objeto é gerado em p-code. Assim, este componente gera como saída, um programa semanticamente equivalente ao código fonte dado como entrada pelo usuário, mas escrito em p-code.

Para compilar o módulo:

```
$ gcc readInstruction.c setInstruction.c obterPosicaoTS.c
leExpressao.c leControle.c lerTabelaSimbolo.c gci2gco.c
gci2gcomain.c -o gci2gco -w
```

para executar passando como entrada a saída do módulo anterior:

```
$ ./gci2gco < ci.txt
```

O código fonte da implementação deste módulo está no Apêndice F.

2.7. OC (Otimizador de Código)

Este módulo é a etapa final de geração de código pelo compilador. Dependendo do programa dado como entrada no compilador, existirão várias situações em que sejam encontradas sequências de código ineficiente. O que o otimizador faz é aplicar um conjunto de heurísticas que detectem estas sequências para então removê-las e substituí-las por instruções mais eficientes ou que eliminem esta ineficiência.

2.8. Compilando todos os módulos

Assim como a compilação pode ser feita individualmente, existe a possibilidade de compilar e executar todos os módulos de uma só vez. Abaixo é apresentado um exemplo de comando para essa forma de compilação:

```
$ gcc pop.c isVariavel.c pushAda.c buscarProducao.c
adicionarTabSimb.c S.c imprimirArvore.c quick_sort.c
pushAsa.c A.c indiceRealAda.c indiceRealAsa.c
asaToPolonesaReversa.c asaToPolonesa.c read.c write.c
isOperador.c isOperadorLogicoBinario.c geraPseudocodigo.c
expressao.c expressaoLogica.c leListaDeclaracoes.c
leAtribuicao.c leListaAtribuicoes.c controle.c
readInstruction.c setInstruction.c obterPosicaoTS.c
leExpressao.c leControle.c lerTabelaSimbolo.c parser.c
ada2asa.c asa2np.c np2gci.c gci2gco.c main.c -o c0 -w
```

Para compilar um programa a partir do compilador gerado, pode-se seguir o exemplo e executar:

```
$ ./c0 < testeG1.txt
```

3. Descrição UML

Nesta seção serão detalhados os métodos e planos para a implementação do compilador. Para isso, será demonstrado como estão organizados cada um dos módulos a partir de um diagrama de sequência, da listagem de casos de uso e de um cronograma para expor qual foi o planejamento para o desenvolvimento de cada módulo.

3.1. Diagrama de Sequência

O Diagrama de Sequência para o projeto é apresentado abaixo. Como o módulo de otimizador de código não foi implementado, este não foi representado no diagrama.

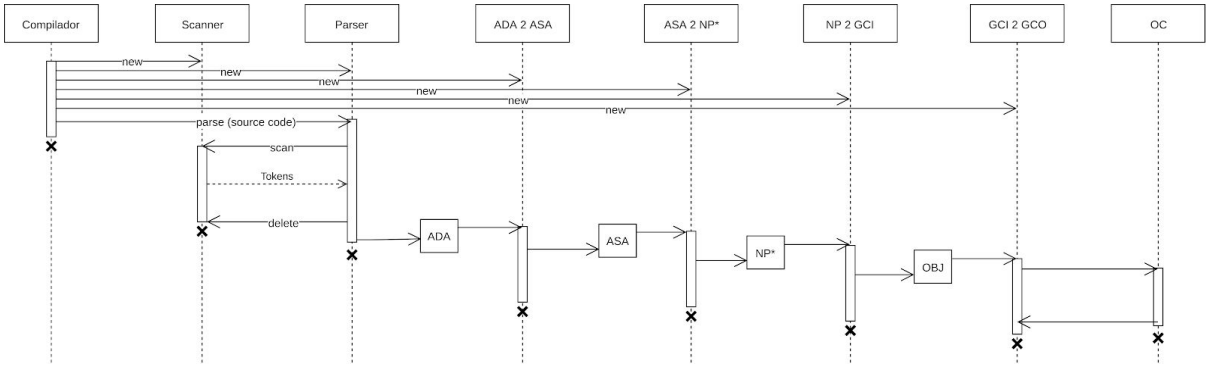

Figura 1. Diagrama de Sequência

3.2. Caso de Uso compilação bem sucedida

- Usuário entra com o código fonte.
- O compilador repassa o código fonte para o parser.
- O parser envia o código fonte para o scanner.
- O scanner realiza a análise de linhas de caracteres e gera a tabela de símbolos.
- O parser utiliza a tabela de símbolos para construção da árvore de análise sintática.
- A árvore de análise sintática então é convertida em árvore de análise sintática abstrata.
- A árvore de análise sintática abstrata por sua vez é transformada em notação polonesa e/ou polonesa reversa.
- A partir da notação polonesa é gerado o código intermediário.

- Baseado no código intermediário é gerado o código objeto.

3.3. Casos de Uso de erros sintáticos

3.3.1. Cadeia consumida e pilha não vazia

- Parser consome totalmente a cadeia
- Parser verifica que a pilha não está vazia
- Ainda faltam ser incluídos símbolos na cadeia de entrada para que esta seja aceita
- É gerado um erro

3.3.2. Cadeia não consumida e pilha vazia

- Parser verifica que a pilha está vazia
- Parser verifica que a cadeia não foi totalmente consumida
- Prefixo da cadeia de entrada é o suficiente para que esta seja aceita
- É gerado um erro

3.3.3. Lookahead diferente do topo da pilha

- Parser verifica que lookahead é diferente do topo da pilha
- Token recebido, o qual fez tal produção estar no topo da pilha, não é o esperado pelo parser.
- É gerado um erro

3.4. Cronograma

A Figura 2 mostra o cronograma de criação do compilador. Cada um dos projetos listados tiveram duração de 2 dias.

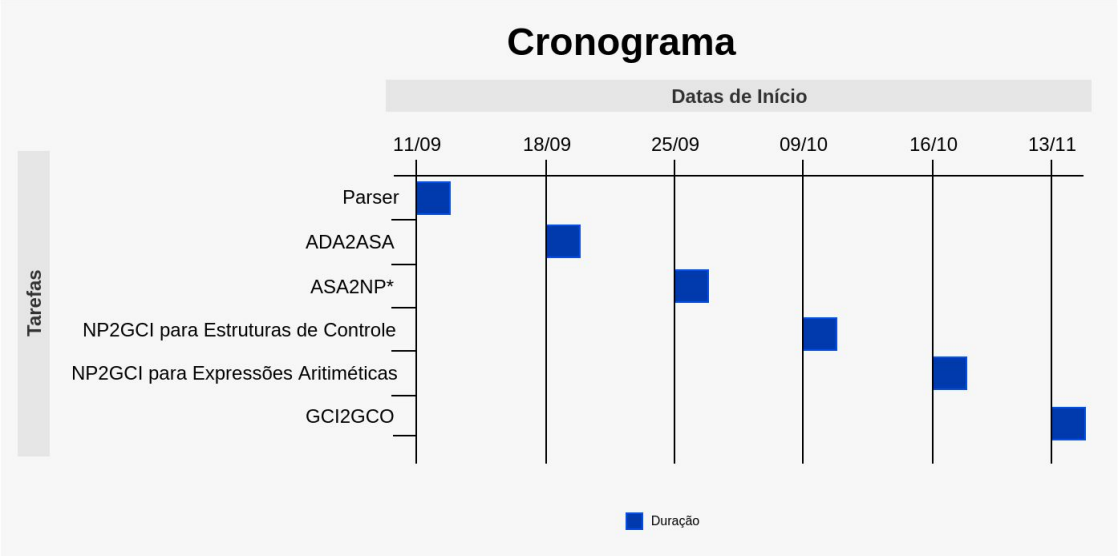

Figura 2. Cronograma do Projeto

4. Plano de Testes

Esta seção mostra alguns programas que foram submetidos como entrada para testar a implementação de cada módulo do compilador. Para cada teste é explicado o arquivo de entrada, os comandos para a execução do teste em cada módulo e o formato do arquivo de saída.

4.1. Teste com IF's sequenciais

4.1.1. Arquivo de Entrada

O arquivo de entrada para qualquer teste é o programa que será compilado, o qual neste caso possui dois if's sequenciais, ou seja, em algum trecho do programa deve existir f(B){ LC }f(B){ LC } (ver Apêndice G.1.).

4.1.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG1.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG1.txt
```

4.1.3. Formato do Arquivo de Saída

O arquivo de saída, evidentemente, será diferente para cada módulo conforme o programa vai sendo analisado e transformado para código objeto. Para o módulo Parser, o arquivo de saída será a representação da ADA em vetor com os devidos índices e valores em cada posição do vetor respectivamente. Da mesma forma é apresentado o arquivo de saída para o módulo ADA2ASA, que neste caso, a saída é a representação da ASA. A saída do módulo ASA2NP* é somente a NP* em uma linha. Para o módulo NP2GCI a saída é o código intermediário com operações que referenciam a tabela de símbolos e a pilha. A saída do módulo GCI2GCO é a tradução do código intermediário em código objeto que é a representação do programa submetido como entrada em p-code (ver Apêndice G.1.).

4.2. Teste com WHILE's sequenciais

4.2.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui dois while's sequenciais, ou seja, em algum trecho do programa deve existir w(B){ LC }w(B){ LC } (ver Apêndice G.2.).

4.2.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG2.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG2.txt
```

4.2.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.2.).

4.3. Teste com FOR's sequenciais

4.3.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui dois for's sequenciais, ou seja, em algum trecho do programa deve existir o(A;E;A){ LC }o(A;B;A){ LC } (ver Apêndice G.3.).

4.3.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG3.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG3.txt
```

4.3.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.3.).

4.4. Teste com IF's aninhados

4.4.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui dois if's aninhados, ou seja, em algum trecho do programa deve existir f(B){ f(B){ LC } } (ver Apêndice G.4.).

4.4.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG4.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG4.txt
```

4.4.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.4.).

4.5. Teste com WHILE's aninhados

4.5.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui dois while's aninhados, ou seja, em algum trecho do programa deve existir w(B){ w(B){ LC } } (ver Apêndice G.5.).

4.5.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG5.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG5.txt
```

4.5.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.5.).

4.6. Teste com FOR's aninhados

4.6.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui dois for's aninhados, ou seja, em algum trecho do programa deve existir o(A;B;A){ o(A;B;A){ LC } } (ver Apêndice G.6.).

4.6.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG6.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG6.txt
```

4.6.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.6.).

4.7. Teste contendo IF com WHILE aninhado

4.7.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui um if com um while aninhado, ou seja, em algum trecho do programa deve existir f(B){ w(B){ LC } } (ver Apêndice G.7.).

4.7.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG7.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG7.txt
```

4.7.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.7.).

4.8. Teste contendo IF com FOR aninhado

4.8.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui um if com um for aninhado, ou seja, em algum trecho do programa deve existir f(B){ o(A;B;A){ LC } } (ver Apêndice G.8.).

4.8.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG8.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG8.txt
```

4.8.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.8.).

4.9. Teste contendo WHILE com IF aninhado

4.9.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui um while com um if aninhado, ou seja, em algum trecho do programa deve existir w(B){ f(B){ LC } } (ver Apêndice G.9.).

4.9.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG9.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG9.txt
```

4.9.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.9.).

4.10. Teste contendo WHILE com FOR aninhado

4.10.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui um while com um for aninhado, ou seja, em algum trecho do programa deve existir w(B){ o(A;B;A){ LC } } (ver Apêndice G.10.).

4.10.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG10.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG10.txt
```

4.10.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.10.).

4.11. Teste contendo FOR com IF aninhado

4.11.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui um for com um if aninhado, ou seja, em algum trecho do programa deve existir o(A;B;A){ f(B){ LC } } (ver Apêndice G.11.).

4.11.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG11.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG11.txt
```

4.11.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.11.).

4.12. Teste contendo FOR com WHILE aninhado

4.12.1. Arquivo de Entrada

Analogamente ao teste anterior, no arquivo de entrada é incluído o programa a ser compilado, que neste caso possui um for com um while aninhado, ou seja, em algum trecho do programa deve existir o(A;B;A){ w(B){ LC } } (ver Apêndice G.12.).

4.12.2. Comando de Execução do Teste

Para executar o teste em cada módulo, basta compilá-los (ver Módulos do Compilador) e executá-los com as devidas entradas:

Parser

```
$ ./parser < testeG12.txt
```

ADA2ASA

```
$ ./ada2asa < ada.txt
```

ASA2NP*

```
$ ./asa2np < asa.txt
```

NP2GCI

```
$ ./np2gci < np.txt
```

GCI2GCO

```
$ ./gci2gco < ci.txt
```

Para executar o teste com todos os módulos compilados de uma só vez:

```
$ ./c0 < testeG12.txt
```

4.12.3. Formato do Arquivo de Saída

O arquivo de saída para todos os módulos é análogo ao teste anterior, somente alterando o conteúdo já que a entrada é diferente (ver Apêndice G.12.).

Apêndices

A. Gramática

Abaixo é apresentada a gramática livre de contexto da sintaxe da linguagem de programação para a qual o compilador está sendo desenvolvido.

```
Mim(){S}
ScX;S
SiX;S
SX=E;C
Sw(E){C}C
Sf(E){C}C
So(E;E;E){C}C
Sr(E);
E0
E1
E2
E3
E4
E5
E6
E7
E8
E9
EX
E-E
E(EBE)
B+
B-
B*
B/
B<
B>
CX=E;C
Cw(E){C}C
Cf(E){C}C
Co(A;E;A){C}C
Cr(E);
```

```
C.
Xh
Xk
Xs
Xx
Xy
Xz
AX=E
AE
```

B. Parser

Abaixo é apresentado o código fonte do módulo.

parseredd.h

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

#include "globalparser.h"
#include "pop.h"
#include "isVariavel.h"
#include "pushAda.h"
#include "buscarProducao.h"
#include "adicionarTabSimb.h"
#include "imprimirArvore.h"
#include "quick_sort.h"
#include "S.h"
#include "parser.h"
```

globalparser.h

```
typedef struct producao{
    char variavel[5];
    char simbolos[20];
} produc;

typedef struct{
    char tipo;
    char simbolo;
    char valor;
} simbTabela;

simbTabela tabelaSimbolos[100];
int qtdTabelaSimbolos;

char prod[512];
char pilha[1024];
char cadeia[512];
int fimArvore;
int topo;
int psize;
int nivel[50];
int tnivel;
```

```
char ada[512];
unsigned long topoAda;
unsigned long indicesAda[512];
unsigned long topoIndicesAda;

char variaveis[50];
int fimVariaveis;

produc producoes[500];
int qtdProducoes;
```

pop.h

```
char pop();
```

isVariavel.h

```
int isVariavel(char c);
```

pushAda.h

```
void pushAda(produc p, int n);
```

buscarProducao.h

```
int buscarProducao(char variavel, char token);
```

adicionarTabSimb.h

```
void adicionarTabSimb(char tipo, char simb, char valor);
```

imprimirArvore.h

```
void imprimirArvore(char *arvore, unsigned long *indices,
unsigned long topo, FILE *saida);
```

quick_sort.h

```
void quick_sort(char *arvore, unsigned long *mapa, int left,
int right);
```

S.h

```
void S();
```

parser.h

```
void parser();
```

pop.c

```
#include "parseredd.h"

char pop(){
```

```
    return pilha[topo--];
}
```

isVariavel.c

```
#include "parseredd.h"

int isVariavel(char c){
    for(size_t k = 0; k < fimVariaveis; k++)
    {
        if(c == variaveis[k])
            return 1;
    }
    return 0;
}
```

pushAda.c

```
#include "parseredd.h"

void pushAda(produc p, int n){
    int tam = strlen(p.simbolos);
    for(size_t k = tam - 1; k != -1; k--){
        pilha[++topo] = p.simbolos[k];
        if(isVariavel(p.simbolos[k]))
            nivel[tnivel++] = psize*n+k+1;
        ada[topoAda++] = p.simbolos[k];
        indicesAda[topoIndicesAda++] = psize*n+k+1;
    }
    if(fimArvore < psize*n+tam+1){
        fimArvore = psize*n+tam+1;
    }
}
```

buscarProducao.c

```
#include "parseredd.h"

int buscarProducao(char variavel, char token){
    for(size_t i = 0; i < qtdProducoes; i++){
        if(producoes[i].variavel[0] == variavel){
            if(isVariavel(producoes[i].simbolos[0])){
                int p =
buscarProducao(producoes[i].simbolos[0], token);
                if(p != -1)
                    return p;
            }
            else if(token == producoes[i].simbolos[0])
```

```
                return i;
            }
        }
        return -1;
}
```

adicionarTabSimb.c

```
#include "parseredd.h"

void adicionarTabSimb(char tipo, char simb, char valor){
    simbTabela newSimbolo;
    newSimbolo.tipo = tipo;
    newSimbolo.simbolo = simb;
    newSimbolo.valor = valor;

    tabelaSimbolos[qtdTabelaSimbolos++] = newSimbolo;
}
```

imprimirArvore.c

```
#include "parseredd.h"

void imprimirArvore(char *arvore, unsigned long *indices,
unsigned long topo, FILE *saida){
   for(size_t j = 0; j < topo; j++){
       fprintf(saida, "%u ", indices[j]);
    }
   fprintf(saida, "\n");

    for(size_t j = 0; j < topo; j++){
       fprintf(saida, "%c ", arvore[j]);
    }
   fprintf(saida, "\n");
}
```

quick_sort.c

```
#include "parseredd.h"

void quick_sort(char *arvore, unsigned long *mapa, int left,
int right) {
   int i, j, x, y;
    char a;

   i = left;
   j = right;
   x = mapa[(left + right) / 2];
```

```
    while(i <= j) {
        while(mapa[i] < x && i < right) {
            i++;
        }
        while(mapa[j] > x && j > left) {
            j--;
        }
        if(i <= j) {
            y = mapa[i];
            mapa[i] = mapa[j];
            mapa[j] = y;
             a = arvore[i];
            arvore[i] = arvore[j];
            arvore[j] = a;
            i++;
            j--;
        }
    }

    if(j > left) {
        quick_sort(arvore, mapa, left, j);
    }
    if(i < right) {
        quick_sort(arvore, mapa, i, right);
    }
}
```

S.c

```
#include "parseredd.h"

void S(){
    int j = 0, n = 0, flag = 0;
    pilha[topo] = variaveis[0];

    ada[topoAda++] = variaveis[0];
    indicesAda[topoIndicesAda++] = 0;

    nivel[tnivel++] = 0;
    while(cadeia[j] != '\0'){
        for(size_t i = 0; i < qtdProducoes; i++)
        {
            if(pilha[topo] == producoes[i].variavel[0]){
                if(isVariavel(producoes[i].simbolos[0])){

if(buscarProducao(producoes[i].simbolos[0], cadeia[j]) !=
```

```
-1)
                        flag = 1;
                }
                if(cadeia[j] == producoes[i].simbolos[0] ||
flag == 1){
                    if((producoes[i].simbolos[0] == 'i' ||
producoes[i].simbolos[0] == 'c') &&
producoes[i].simbolos[1] != 'm'){
                        adicionarTabSimb(cadeia[j],
cadeia[j+1], '0');
                    }
                    flag = 0;
                    pop();
                    tnivel--;
                    pushAda(producoes[i], nivel[tnivel]);
                    n = psize*n +
strlen(producoes[i].simbolos);
                }
            }
            else if(pilha[topo] == cadeia[j]){
                pop();
                j++;
                n++;
                break;
            }
        }
    }
}
```

parser.c

```
#include "parseredd.h"

void parser(){
    int i, j;
    FILE * tabSimbFile;
   FILE *saida, *glc_file;
    psize = 0;
    topoAda = 0;
    topoIndicesAda = 0;
   fimArvore = fimVariaveis = qtdProducoes =
qtdTabelaSimbolos = 0;

   glc_file = fopen("glc.txt", "r");
   if(glc_file == NULL){
       printf("Nao foi possivel abrir GLC.");
```

```
        exit(-1);
    }

    saida = fopen("ada.txt", "w");
    if(saida == NULL){
        printf("Nao foi possivel gerar ADA.");
        exit(-1);
    }

    while (fscanf(glc_file, " %[^\n]s", prod) != EOF){
         i = 0;
         j = 0;
         produc p;
         if(fimVariaveis == 0 || variaveis[fimVariaveis - 1]
!= prod[i])
             variaveis[fimVariaveis++] = prod[i];

         p.variavel[j++] = prod[i++];
         p.variavel[j] = '\0';

         j = 0;
         while(prod[i] != '\0'){
             p.simbolos[j++] = prod[i++];
         }
         p.simbolos[j] = '\0';
         if(strlen(p.simbolos) > psize)
             psize = strlen(p.simbolos);

         producoes[qtdProducoes++] = p;
     }

     scanf(" %[^\n]s", cadeia);
    topo = 0;
    tnivel = 0;

    S();
    quick_sort(ada, indicesAda, 0, topoIndicesAda-1);
    imprimirArvore(ada, indicesAda, topoIndicesAda, saida);

    tabSimbFile = fopen("TabelaSimbolos.txt", "w");
    for(size_t i = 0; i < qtdTabelaSimbolos; i++){
        fprintf(tabSimbFile, "%c %c %c\n",
tabelaSimbolos[i].tipo, tabelaSimbolos[i].simbolo,
tabelaSimbolos[i].valor);
    }
```

```
    for(size_t i = 0; i < 10; i++){
        fprintf(tabSimbFile, "s %d %d\n", i, i);
    }

    fclose(tabSimbFile);
    fclose(glc_file);
    fclose(saida);
}
```

parsermain.c

```
#include "parseredd.h"

int main(void){
    printf("\nSaida Gerada:\n");
    parser();
    printf (" \u2713\tADA (Arvode Sintatica
Descendente)\n");

    return 0;
}
```

C. ADA2ASA

Abaixo é apresentado o código fonte do módulo.

ada2asaedd.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "globalada2asa.h"
#include "pushAsa.h"
#include "A.h"
#include "indiceRealAda.h"
#include "imprimirArvore.h"
#include "quick_sort.h"
#include "ada2asa.h"
```

globalada2asa.h

```
int psize;

char ada[512];
unsigned long topoAda;
unsigned long indicesAda[512];
unsigned long topoIndicesAda;

char asa[512];
unsigned long topoAsa;
unsigned long indicesAsa[512];
unsigned long topoIndicesAsa;
```

pushAsa.h

```
void pushAsa(char c, int n);
```

A.h

```
void A(int n, int p);
```

indiceRealAda.h

```
int indiceRealAda(int valor);
```

ada2asa.h

```
void ada2asa();
```

pushAsa.c

```
#include "ada2asaedd.h"

void pushAsa(char c, int n){
```

```
    asa[topoAsa++] = c;
    indicesAsa[topoIndicesAsa++] = n;
}
```

A.c

```
#include "ada2asaedd.h"

void A(int n, int p){
    if (ada[indiceRealAda(p)] == 'i' &&
ada[indiceRealAda(p+1)] == 'm'){
        pushAsa(ada[indiceRealAda(p)], n);
        pushAsa(ada[indiceRealAda(p+1)], 3*(n)+1);
        A(3*(n)+2, (psize*(p+5))+1);
    }
    else if (ada[indiceRealAda(p)] == 'c' ||
ada[indiceRealAda(p)] == 'i'){
        pushAsa(ada[indiceRealAda(p)], n);
        pushAsa(ada[indiceRealAda(p+2)], n+1);
        A(3*(n)+1, (psize*(p+1))+1);
        A(3*(n+1)+1, (psize*(p+3))+1);
    }
    else if (ada[indiceRealAda(p+1)] == '=' &&
indiceRealAda(p+3) == -1){
        pushAsa(ada[indiceRealAda(p+1)], n);

        A(3*(n)+1, (psize*(p))+1);
        A(3*(n)+2, (psize*(p+2))+1);
    }
    else if (ada[indiceRealAda(p+1)] == '='){
        pushAsa(ada[indiceRealAda(p+1)], n);
        pushAsa(ada[indiceRealAda(p+3)], n+1);

        A(3*(n)+1, (psize*(p))+1);

        A(3*(n)+2, (psize*(p+2))+1);
        A(3*(n+1)+1, (psize*(p+4))+1);
    }
    else if (ada[indiceRealAda(p)] == '('){
        A(3*(n)+1, (psize*(p+1))+1);
        A(n, (psize*(p+2))+1);
        A(3*(n)+2, (psize*(p+3))+1);
    }
    else if (ada[indiceRealAda(p)] == 'r'){
        pushAsa(ada[indiceRealAda(p)], n);
        pushAsa(ada[indiceRealAda(p+4)], n+1);
```

```
        A(3*(n)+1, (psize*(p+2))+1);
    }
    else if (ada[indiceRealAda(p)] == 'f'){
        pushAsa(ada[indiceRealAda(p)], n);
        pushAsa('.', n+1);
        A(3*(n)+1, (psize*(p+2))+1);
        A(3*(n)+2, (psize*(p+5))+1);
        A(3*(n+1)+1, (psize*(p+7))+1);
    }
    else if (ada[indiceRealAda(p)] == 'w'){
        pushAsa(ada[indiceRealAda(p)], n);
        pushAsa('.', n+1);
        A(3*(n)+1, (psize*(p+2))+1);
        A(3*(n)+2, (psize*(p+5))+1);
        A(3*(n+1)+1, (psize*(p+7))+1);
    }
    else if (ada[indiceRealAda(p)] == 'o'){
        pushAsa(ada[indiceRealAda(p)], n);
        pushAsa('.', n+1);

        A(3*(n)+1, (psize*(p+2))+1);
        pushAsa(';', 3*(n)+2);
        A(3*(3*(n)+2)+1, (psize*(p+4))+1);
        pushAsa(';', 3*(3*(n)+2)+2);
        A(3*(3*(3*(n)+2)+2)+1, (psize*(p+9))+1);
        A(3*(n)+3, (psize*(p+6))+1);

        A(3*(n+1)+1, (psize*(p+11))+1);
    }
    else if (ada[indiceRealAda(p)] == '.'){
    }
    else if (ada[indiceRealAda(p)] == 'X'){
        A(n, (psize*(p))+1);
    }
    else{
        pushAsa(ada[indiceRealAda(p)], n);
    }
}
```

indiceRealAda.c

```
#include "ada2asaedd.h"

int indiceRealAda(int valor){
    int inf, sup, meio;
    inf = 0;
```

```
    sup = topoIndicesAda-1;
    while (inf <= sup)
    {
        meio = (inf + sup)/2;
        if (valor == indicesAda[meio])
            return meio;
        else if (valor < indicesAda[meio])
            sup = meio-1;
        else
            inf = meio+1;
    }
    return -1;
}
```

ada2asa.c

```
#include "ada2asaedd.h"

void ada2asa(){
    int i, j;
   FILE *saida;
   FILE *entrada;

   entrada = fopen("ada.txt", "r");
   if(entrada == NULL){
       printf("Nao foi possivel abrir arquivo de
entrada.");
       exit(-1);
   }

   saida = fopen("asa.txt", "w");
   if(saida == NULL){
       printf("Nao foi possivel gerar ASA.");
       exit(-1);
   }

    psize = 12;

    topoAda = 0;
    topoIndicesAda = 0;
    topoAsa = 0;
    topoIndicesAsa = 0;

    char linha[16384];
    char parte[32];
```

```
    fscanf(entrada, " %[^\n]s", linha);

     i = 0;
     while(linha[i] != '\0'){
         j = 0;
         while(linha[i] != ' '){
             parte[j++] = linha[i++];
         }

         i++;
         parte[j] = '\0';

         indicesAda[topoIndicesAda++] = atoi(parte);
     }

    fscanf(entrada, " %[^\n]s", linha);
     i = 0;
     j = 0;
     while(linha[i] != '\0'){
         ada[topoAda++] = linha[i];
         i+=2;

     }

     A(0,1);
     quick_sort(asa, indicesAsa, 0, topoIndicesAsa-1);
     imprimirArvore(asa, indicesAsa, topoIndicesAsa, saida);

    fclose(saida);
    fclose(entrada);
}
```

ada2asamain.c

```
#include "ada2asaedd.h"

int main(void){
    printf("\nSaida Gerada:\n");
    ada2asa();
    printf (" \u2713\tASA (Arvode Sintatica Abstrata)\n");

    return 0;
}
```

D. ASA2NPR

Abaixo é apresentado o código fonte do módulo.

asa2npedd.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "globalasa2np.h"
#include "indiceRealAsa.h"
#include "asaToPolonesaReversa.h"
#include "asaToPolonesa.h"
#include "asa2np.h"
```

globalasa2np.h

```
char prod[512];
char pilha[1024];
char cadeia[512];
int topo;
int psize;
int nivel[50];
int tnivel;

char asa[512];
unsigned long topoAsa;
unsigned long indicesAsa[512];
unsigned long topoIndicesAsa;

char variaveis[50];
char polonesaR[512];
int iP;
```

indiceRealAsa.h

```
int indiceRealAsa(int valor);
```

asaToPolonesaReversa.h

```
void asaToPolonesaReversa(int n);
```

asaToPolonesa.h

```
void asaToPolonesa(int n);
```

asa2np.h

```
void asa2np();
```

indiceRealAsa.c

```c
#include "asa2npedd.h"

int indiceRealAsa(int valor){
    int inf, sup, meio;
    inf = 0;
    sup = topoIndicesAsa-1;
    while (inf <= sup)
    {
        meio = (inf + sup)/2;
        if (valor == indicesAsa[meio])
            return meio;
        else if (valor < indicesAsa[meio])
            sup = meio-1;
        else
            inf = meio+1;
    }
    return -1;
}
```

asaToPolonesaReversa.c

```c
#include "asa2npedd.h"

void asaToPolonesaReversa(int n){
    if(indiceRealAsa(3*n+1) == -1 || 3*n+1 >
indicesAsa[topoIndicesAsa - 1]){
        polonesaR[iP++] = asa[indiceRealAsa(n)];
    }
    else{
        asaToPolonesaReversa(3*n+1);
        if(asa[indiceRealAsa(3*n+2)] != ' ' && 3*n+2 <=
indicesAsa[topoIndicesAsa - 1]){
            asaToPolonesaReversa(3*n+2);
        }
        polonesaR[iP++] = asa[indiceRealAsa(n)];
    }
}
```

asaToPolonesa.c

```c
#include "asa2npedd.h"

void asaToPolonesa(int n){
    if(indiceRealAsa(3*n+1) == -1 || 3*n+1 >
indicesAsa[topoIndicesAsa - 1]){
        polonesaR[iP++] = asa[indiceRealAsa(n)];
    }
```

```
    else{
        polonesaR[iP++] = asa[indiceRealAsa(n)];
        asaToPolonesa(3*n+1);

        for(size_t k = 2; indiceRealAsa(3*n+k) != -1 &&
3*n+k <= indicesAsa[topoIndicesAsa - 1] && k <= 3; k++)
        {
            if(asa[indiceRealAsa(n)] != '=')
                asaToPolonesa(3*n+k);
            else
                asaToPolonesaReversa(3*n+k);
        }
    }
}
```

asa2np.c

```
#include "asa2npedd.h"

void asa2np(){
    int i, j;
   FILE *saida;
   FILE *entrada;

   entrada = fopen("asa.txt", "r");
   if(entrada == NULL){
       printf("Nao foi possivel abrir arquivo de
entrada.");
       exit(-1);
   }

   saida = fopen("np.txt", "w");
   if(saida == NULL){
       printf("Nao foi possivel gerar NP.");
       exit(-1);
   }

    psize = 0;
    topoAsa = 0;
    topoIndicesAsa = 0;

    char linha[1024];
    char parte[32];
   fgets(linha, 1024, entrada);

    i = 0;
```

```
    while(linha[i] != '\n'){
        j = 0;
        while(linha[i] != ' '){
            parte[j++] = linha[i++];
        }

        i++;
        parte[j] = '\0';

        indicesAsa[topoIndicesAsa++] = atoi(parte);
    }

   fgets(linha, 1024, entrada);
    //scanf(" %[^\n]s", linha);
    i = 0;
    j = 0;
    while(linha[i] != '\0'){
        asa[topoAsa++] = linha[i];
        i+=2;
    }

    asaToPolonesa(0);
    polonesaR[iP] = '\0';
    //printf("%s\n", polonesaR);
   fprintf(saida, "%s\n", polonesaR);

   fclose(entrada);
   fclose(saida);
}
```

asa2npmain.c

```
#include "asa2npedd.h"

int main(void){
   printf("\nSaida Gerada:\n");
   asa2np();
   printf (" \u2713\tNP (Notacao Polonesa)\n");

   return 0;
}
```

E. NPR2GCI

Abaixo é apresentado o código fonte do módulo.

globalnp2gci.h

```
char notacaoPolonesa[100];
int count;
char token[1];
char aux[50];
char arquivos[200][50];
int arquivosPoint;
```

read.h

```
char read();
```

write.h

```
void write(char *cont, int nameCount);
```

isOperator.h

```
int isOperador(char c);
```

isOperadorLogicoBinario.h

```
int isOperadorLogicoBinario(char c);
```

geraPseudocodigo.h

```
void geraPseudocodigo();
```

expressao.h

```
int expressao(int inst, int nameCount);
```

expressaoLogica.h

```
int expressaoLogica(int inst, int nameCount);
```

leListaDeclaracoes.h

```
int leListaDeclaracoes(int inst, int nameCount);
```

leAtribuicao.h

```
int leAtribuicao(int inst, int nameCount);
```

leListaAtribuicoes.h

```
int leListaAtribuicoes(int inst, int nameCount);
```

controle.h

```
int controle(int nameCount, int inst);
```

np2gci.h

```
void np2gci();
```

read.c

```
#include "np2gciedd.h"

char read(){
    return notacaoPolonesa[count++];
}
```

write.c

```
#include "np2gciedd.h"

void write(char *cont, int nameCount){
    FILE *arqSaida;
    char name[50];

    sprintf(arquivos[arquivosPoint++], "%d.inst",
nameCount);
    sprintf(name, "%d.inst", nameCount);
    arqSaida = fopen(name, "w");
    fprintf(arqSaida, "%s", cont);
    fclose(arqSaida);
}
```

isOperador.c

```
#include "np2gciedd.h"

int isOperador(char c){
    if (c == '+' || c == '*' || c == '-' || c == '/' || c
== '~')
    {
        return 1;
    }
    return 0;
}
```

isOperadorLogicoBinario.c

```
#include "np2gciedd.h"

int isOperadorLogicoBinario(char c){
    if (c == '>' || c == '<' || c == '&' || c == '|' || c
== ':' || c == '_')
    {
        return 1;
    }
    return 0;
```

```
}
```

geraPseudocodigo.c

```
#include "np2gciedd.h"

#if defined(_WIN32) || defined(_WIN64)
const char *os = "dir /b *.inst";
#else
#ifdef __linux
const char *os = "ls *.inst";
#else
const char *os = "Unknown";
#endif
#endif

void geraPseudocodigo(){
    FILE *files;
    FILE *saida;
    char line[20];

    FILE *fp = popen(os, "r");
    if (fp == NULL)
    {
        printf("Failed to run command\n");
        exit(1);
    }

    saida = fopen("ci.txt", "a");
    if (saida == NULL)
    {
        printf("Failed to run command\n");
        exit(1);
    }

    for (int i = 0; i < arquivosPoint; i++)
    {
        files = fopen(arquivos[i], "r");
        if (files == NULL)
        {
            printf("Failed to run command\n");
            exit(1);
        }
        fscanf(files, "%[^\n]s", line);
        fprintf(saida, "%s\n", line);
        fclose(files);
```

```
    }

    fprintf(saida, "\n");

    fclose(saida);
    fclose(fp);
}
```

expressao.c

```
#include "np2gciedd.h"

int expressao(int inst, int nameCount){
   int cabecalhoOn = 0;

    *token = read();
   if(notacaoPolonesa[count] != ';'){
       write("EXPR:", nameCount + inst);
       inst++;
       cabecalhoOn = 1;
   }
    while(*token != ';' && *token != '.'){
        if(*token == '+'){
            write("ADD", nameCount+inst);
            inst++;
        }
        else if(*token == '-'){
            write("SUB", nameCount+inst);
            inst++;
        }
        else if(*token == '*'){
            write("MUL", nameCount+inst);
            inst++;
        }
        else if(*token == '/'){
            write("DIV", nameCount+inst);
            inst++;
        }
        else if(*token == '~'){
            write("NEG", nameCount+inst);
            inst++;
        }
        else{
            write(token, nameCount + inst);
            inst++;
            if(notacaoPolonesa[count] != ';'){
```

```
                sprintf(aux, "push(%c)", *token);
                write(aux, nameCount + inst);
                inst++;
            }
        }
        *token = read();
    }

   if(cabecalhoOn){
       write("ENDEXPR:", nameCount + inst);
       inst++;
   }

    return inst;
}
```

expressaoLogica.c

```
#include "np2gciedd.h"

int expressaoLogica(int inst, int nameCount){
   if(isOperadorLogicoBinario(*token)){
       sprintf(aux, "%c%c%c", notacaoPolonesa[count-1],
*token, notacaoPolonesa[++count]);
       write(aux, nameCount+inst);
       inst++;
       *token = read();
   }
    else{
        sprintf(aux, "%c", *token);
       write(aux, nameCount+inst);
       inst++;
    }
   *token = read();

    return inst;
}
```

leListaDeclaracoes.c

```
#include "np2gciedd.h"

int leListaDeclaracoes(int inst, int nameCount){
   *token = read();
   while (*token == 'i' || *token == 'c')
   {
       if (*token == 'i')
       {
```

```
            *token = read();
            while (*token != ';')
            {
                write("TABSIMB.addInt(", nameCount + inst);
                inst++;
                write(token, nameCount + inst);
                inst++;
                write(")", nameCount + inst);
                inst++;
                *token = read();
            }
        }
        else
        {
            *token = read();
            while (*token != ';')
            {
                write("TABSIMB.addChar(", nameCount + inst);
                inst++;
                write(token, nameCount + inst);
                inst++;
                write(")", nameCount + inst);
                inst++;
                *token = read();
            }
        }
        //Lê o ;
        *token = read();

    }
    return inst;
}
```

leAtribuicao.c

```
#include "np2gciedd.h"

int leAtribuicao(int inst, int nameCount){
    write("TABSIMB.update(", nameCount + inst);
    inst++;
    *token = read();
    sprintf(aux, "%c,", *token);
    write(aux, nameCount + inst);
    inst++;
    inst = expressao(inst, nameCount);
    write(")", nameCount + inst);
```

```
    inst++;
    *token = read();

    return inst;
}
```

leListaAtribuicoes.c

```
#include "np2gciedd.h"

int leListaAtribuicoes(int inst, int nameCount){
    while(*token == '='){
        *token = read();
        write("TABSIMB.update(", nameCount + inst);
        inst++;
        sprintf(aux, "%c,", *token);
        write(aux, nameCount + inst);
        inst++;
        inst = expressao(inst, nameCount);
        write(")", nameCount + inst);
         inst++;
        if(*token != '.'){
            *token = read();
        }
    }

    return inst;
}
```

controle.c

```
#include "np2gciedd.h"

int controle(int nameCount, int inst){
    char aux2[50];
     if (*token != '='){
         inst = 0;
         nameCount = nameCount * 100 + 1000;
     }

     while(*token != '\0'){

         if (*token == 'f'){
             write("IF:", nameCount + inst);
             inst++;
             *token = read();
             inst = expressaoLogica(inst, nameCount);
            inst++;
```

```
            sprintf(aux2, "if(!%s) goto ENDIF", aux);
            write(aux2, nameCount + inst);
            inst++;

            inst = controle(nameCount, inst);

            write("ENDIF:", nameCount + 99);

            //Verificando IF's sequenciais
            *token = read();
            while(*token == 'f' || *token == 'w' || *token
== 'o'){
                inst = controle(nameCount, inst);
            }
        }
        else if (*token == 'w'){
            write("WHILE:", nameCount + inst);
            inst++;
            *token = read();
            inst = expressaoLogica(inst, nameCount);
           inst++;
            sprintf(aux2, "if(!%s) goto ENDWHILE", aux);
            write(aux2, nameCount + inst);
            inst++;

            inst = controle(nameCount, inst);

            sprintf(aux, "goto WHILE:");
            write(aux, nameCount + inst);
            inst++;
            write("ENDWHILE:", nameCount + 98);
        }
        else if (*token == 'o'){
            *token = read();

            inst = leAtribuicao(inst, nameCount);

            write("FOR:", nameCount + inst);
            inst++;

           inst = expressaoLogica(inst, nameCount);

            sprintf(aux2, "if(!%s) goto ENDFOR", aux);
            write(aux2, nameCount + inst);
            inst++;
```

```
            *token = read();


            inst = controle(nameCount, inst);

            write("goto FOR", nameCount + inst);
            inst++;
            write("ENDFOR:", nameCount + 99);
        }
        else if (*token == '='){
            inst = leListaAtribuicoes(inst, nameCount);
           if(*token == 'f' || *token == 'w' || *token ==
'o'){
               inst = controle(nameCount, inst);
           }
           if(*token == '.'){
               return inst;
           }
        }
        else{
            break;
        }

        *token = read();
    }

    return inst;
}
```

np2gci.c

```
#include "np2gciedd.h"

#if defined(_WIN32) || defined(_WIN64)
const char *del = "del *.inst";
#else
#ifdef __linux
const char *del = "rm *.inst";
#else
const char *del = "Unknown";
#endif
#endif

void np2gci(){
    int nameCount = 1000;
```

```
     int inst = 0;
     FILE *arqSaida;
    FILE *entrada;

    entrada = fopen("np.txt", "r");
    if(entrada == NULL){
        printf("Nao foi possivel abrir arquivo de
entrada.");
        exit(-1);
    }

     arqSaida = fopen("ci.txt", "w");
     fclose(arqSaida);

    fgets(notacaoPolonesa, 100, entrada);
    arquivosPoint = 0;
    count = 0;
    inst = 0;
    nameCount = 1000;

    //Lendo tipo de retorno
    *token = read();
    //Lendo função
    *token = read();

    if(*token == 'm'){
        write("MAIN:", nameCount);

        nameCount = nameCount * 100 + nameCount;

        inst = leListaDeclaracoes(inst, nameCount);
        inst = leListaAtribuicoes(inst, nameCount);
        inst = controle(nameCount, inst);

        if(*token == 'r'){
             write("RETURN(", nameCount + inst);
             inst++;
             *token = read();
             sprintf(aux, "%c", *token);
             write(aux, nameCount + inst);
             inst++;
             write(")", nameCount + inst);
         }
    }
```

```
    write("END:", nameCount + 99);
    geraPseudocodigo();
    system(del);
}
```

np2gcimain.c

```
#include "np2gciedd.h"

int main(void){
    printf("\nSaida Gerada:\n");
    np2gci();
    printf (" \u2713\tCI (Codigo Intermediario)\n");

    return 0;
}
```

F. GCI2GCO

Abaixo é apresentado o código fonte do módulo.

gci2gcoedd.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "globalgci2gco.h"
#include "readInstruction.h"
#include "setInstruction.h"
#include "obterPosicaoTS.h"
#include "leExpressao.h"
#include "leControle.h"
#include "lerTabelaSimbolo.h"
#include "gci2gco.h"
```

globalgci2gco.h

```
#define stacksize 500

typedef enum fct
{
   LIT,
   OPR,
   LOD,
   STO,
   CAL,
   INT,
   JMP,
   JPC
} fct;

typedef struct instruction
{
   fct f;
   int l;
   int a;
} instruction;

instruction code[stacksize];
int cont;

typedef struct{
    char tipo;
```

```
    char simbolo;
    int valor;
} simbGcoTabela;

simbGcoTabela tabelaGcoSimbolos[100];
int qtdSimbolosTabela;
FILE *entrada;
```

readInstruction.h

```
void readInstruction(char * instrucao);
```

setInstruction.h

```
void setInstruction(fct func, int level, int arg);
```

obterPosicaoTS.h

```
int obterPosicaoTS(char simb);
```

leExpressao.h

```
void leExpressao();
```

leControle.h

```
void leControle();
```

lerTabelaSimbolo.h

```
void lerTabelaSimbolo();
```

gci2gco.h

```
void gci2gco();
```

readInstruction.c

```
#include "gci2gcoedd.h"

void readInstruction(char * instrucao){
   fscanf(entrada, " %[^\n]s", instrucao);
}
```

setInstruction.c

```
#include "gci2gcoedd.h"

void setInstruction(fct func, int level, int arg){
    code[cont].f = func;
    code[cont].l = level;
    code[cont++].a = arg;
}
```

obterPosicaoTS.c

```c
#include "gci2gcoedd.h"

int obterPosicaoTS(char simb){
    for(size_t i = 0; i < qtdSimbolosTabela; i++)
    {
        if(tabelaGcoSimbolos[i].simbolo == simb){
            if(tabelaGcoSimbolos[i].tipo == 's')
                return -1;
            return i + 2;
        }
    }
}
```

leExpressao.c

```c
#include "gci2gcoedd.h"

void leExpressao(){
    int posicao;
    char instrucao[32];
    char aux[5], aux2[5];

    readInstruction(instrucao);
    if(strcmp(instrucao, "EXPR:") == 0){
        readInstruction(instrucao);
        while (strcmp(instrucao, "ENDEXPR:") != 0)
        {
            if(instrucao[0] == 'p'){
                posicao = obterPosicaoTS(aux[0]);
                if(posicao == -1){
                    sprintf(aux2, "%c", aux[0]);
                    setInstruction(LIT, 0, atoi(aux2));
                }
                else
                    setInstruction(LOD, 0, posicao);
            }
            else if(strcmp(instrucao, "NEG") == 0){
                setInstruction(OPR, 0, 1);
            }
            else if(strcmp(instrucao, "ADD") == 0){
                setInstruction(OPR, 0, 2);
            }
            else if(strcmp(instrucao, "SUB") == 0){
                setInstruction(OPR, 0, 3);
            }
            else if(strcmp(instrucao, "MUL") == 0){
```

```
                setInstruction(OPR, 0, 4);
            }
            else if(strcmp(instrucao, "DIV") == 0){
                setInstruction(OPR, 0, 5);
            }
            else if(strcmp(instrucao, "IGUAL") == 0){
                setInstruction(OPR, 0, 8);
            }
            else if(strcmp(instrucao, "NAOIGUAL") == 0){
                setInstruction(OPR, 0, 9);
            }
            else if(strcmp(instrucao, "<") == 0){
                setInstruction(OPR, 0, 10);
            }
            else if(strcmp(instrucao, "MAIORIGUAL") == 0){
                setInstruction(OPR, 0, 11);
            }
            else if(strcmp(instrucao, ">") == 0){
                setInstruction(OPR, 0, 12);
            }
            else if(strcmp(instrucao, "MENORIGUAL") == 0){
                setInstruction(OPR, 0, 13);
            }
            else if(strcmp(instrucao, ".") == 0){
                continue;
            }
            else{
                strcpy(aux, instrucao);
            }
            readInstruction(instrucao);
        }
    }
    else{
        if(instrucao[1] == '<' || instrucao[1] == '>'){
            posicao = obterPosicaoTS(instrucao[0]);
            if(posicao == -1){
                sprintf(aux2, "%c", instrucao[0]);
                setInstruction(LIT, 0, atoi(aux2));
            }
            else{
                setInstruction(LOD, 0, posicao);

            posicao = obterPosicaoTS(instrucao[2]);
            if(posicao == -1){
                sprintf(aux2, "%c", instrucao[2]);
```

```
                setInstruction(LIT, 0, atoi(aux2));
            }
            else
                setInstruction(LOD, 0, posicao);
            }

            if(instrucao[1] == '<')
                setInstruction(OPR, 0, 10);
            else
                setInstruction(OPR, 0, 12);
        }
        else{
            posicao = obterPosicaoTS(instrucao[0]);
            if(posicao == -1){
                sprintf(aux2, "%c", instrucao[0]);
                setInstruction(LIT, 0, atoi(aux2));
            }
            else
                setInstruction(LOD, 0, posicao);
        }
    }
}
```

leControle.c

```
#include "gci2gcoedd.h"

void leControle(){
    int token;
    char instrucao[32];
    int pilha[32], indice = 0;

    while (fscanf(entrada, " %[^\n]s", instrucao) != EOF)
    {
        if(strcmp(instrucao, "MAIN:") == 0){
            setInstruction(INT, 0, 3);
        }
        else if(strcmp(instrucao, "END:") == 0){
            break;
        }
        else if(strcmp(instrucao, "TABSIMB.addChar(") == 0){
            setInstruction(INT, 0, 1);
        }
        else if(strcmp(instrucao, "TABSIMB.addInt(") == 0){
            setInstruction(INT, 0, 1);
        }
```

```
        else if(strcmp(instrucao, "TABSIMB.update(") == 0){
            readInstruction(instrucao);
            leExpressao();
            token = obterPosicaoTS(instrucao[0]);
            setInstruction(STO, 0, token);
            readInstruction(instrucao);
        }
        else if(strcmp(instrucao, "IF:") == 0){
            leExpressao();
            setInstruction(LIT, 0, 0);
            setInstruction(OPR, 0, 8);
            readInstruction(instrucao);
            if(instrucao[0] == 'i'){
                setInstruction(JPC, 0, 0);
                pilha[indice++] = cont - 1;
            }
        }
        else if(strcmp(instrucao, "ENDIF:") == 0){
            code[pilha[--indice]].a = cont;
        }
        else if(strcmp(instrucao, "WHILE:") == 0){
            pilha[indice++] = cont;
            leExpressao();
            setInstruction(LIT, 0, 0);
            setInstruction(OPR, 0, 8);
            readInstruction(instrucao);
            if(instrucao[0] == 'i'){
                setInstruction(JPC, 0, 0);
                pilha[indice++] = cont - 1;
            }
        }
        else if(strcmp(instrucao, "ENDWHILE:") == 0){
            code[pilha[--indice]].a = cont + 1;
            setInstruction(JMP, 0, pilha[--indice]);
        }
        else if(strcmp(instrucao, "FOR:") == 0){
            pilha[indice++] = cont;
            leExpressao();
            setInstruction(LIT, 0, 0);
            setInstruction(OPR, 0, 8);
            readInstruction(instrucao);
            if(instrucao[0] == 'i'){
                setInstruction(JPC, 0, 0);
                pilha[indice++] = cont - 1;
            }
```

```
        }
        else if(strcmp(instrucao, "ENDFOR:") == 0){
            code[pilha[--indice]].a = cont + 1;
            setInstruction(JMP, 0, pilha[--indice]);
        }
        else{
            continue;
        }
    }
}
```

lerTabelaSimbolo.c

```
#include "gci2gcoedd.h"

void lerTabelaSimbolo(){
    FILE * tabSimbFile;
    simbGcoTabela newSimbolo;
    char linha[5];

    tabSimbFile = fopen("TabelaSimbolos.txt", "r");
    if (!tabSimbFile) {
        printf("Erro ao tentar abrir tabela de simbolos.");
        return ;
    }

    while (!feof(tabSimbFile))
    {
        fscanf(tabSimbFile, " %[^\n]s", linha);
        newSimbolo.tipo = linha[0];
        newSimbolo.simbolo = linha[2];
        newSimbolo.valor = linha[4];

        tabelaGcoSimbolos[qtdSimbolosTabela++] =
newSimbolo;
    }

    fclose(tabSimbFile);
}
```

gci2gco.c

```
#include "gci2gcoedd.h"

char * instructionString[] = { "LIT", "OPR", "LOD", "STO",
"CAL", "INT", "JMP", "JPC", "WRT" };

void gci2gco(){
```

```
    FILE *saida;

    entrada = fopen("ci.txt", "r");
    if(entrada == NULL){
        printf("Nao foi possivel abrir arquivo de
entrada.");
        exit(-1);
    }
    saida = fopen("co.txt", "w");
    if(saida == NULL){
        printf("Nao foi possivel gerar CO.");
        exit(-1);
    }

     lerTabelaSimbolo();

     leControle();
     setInstruction(OPR, 0, 0);

     for(size_t i = 0; i < cont; i++){
        fprintf(saida, "%s %d %d\n",
instructionString[code[i].f], code[i].l, code[i].a);
     }

    fclose(entrada);
    fclose(saida);
}
```

gci2gcomain.c

```
#include "gci2gcoedd.h"

int main(void){
    printf("\nSaida Gerada:\n");
    gci2gco();
    printf (" \u2713\tCO (Codigo Objeto)\n\n");

    return 0;
}
```

G. Programas Teste

G.1. Programa com if's sequenciais

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;is;s=1;x=1;f((s<9)){s=(s+1);.}f((x<9)){x=(x+1);.}r(0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 10969 10993 10994
10995 10996 10997 131917 131941 131965 131966 131967 131968
131969 1583581 1583605 1583629 1583630 1583631 1583632 1583633
1583634 1583635 1583636 19003573 19003574 19003575 19003576
19003577 19003609 19003610 19003611 19003612 19003613 19003633
19003634 19003635 19003636 19003637 19003638 19003639 19003640
228042889 228042901 228042913 228043309 228043333 228043334
228043335 228043336 228043337 228043357 228043621 228043622
228043623 228043624 228043625 228043657 228043658 228043659
228043660 228043661 228043681 228043682 228043683 228043684
228043685 2736514669 2736520009 2736520021 2736520033 2736523465
2736523477 2736523489 2736523885 2736523909 2736523910 2736523911
2736523912 2736523913 2736523933 2736524197 2773469037 2773510509
2773515849 2773515861 2773515873 3217419117
M i m ( ) { S } i X ; S x i X ; S s X = E ; C s 1 X = E ; C x 1 f
( E ) { C } C ( E B E ) X = E ; C f ( E ) { C } C X < 9 s ( E B E
) . ( E B E ) X = E ; C r ( E ) ; s X + 1 X < 9 x ( E B E ) . 0 s
x X + 1 x

Árvore ASA:
0 1 2 3 7 10 11 31 34 35 103 104 106 107 319 320 322 323 967 968
969 970 971 2902 2903 2905 2906 2911 2912 2913 2914 2915 8719
8720 8734 8735 8737 8738 8743 26215 26216
i m i ; x i ; s = ; s 1 = ; x 1 f . < = ; f . s 9 s + < = ; r ; s
1 x 9 x + 0 x 1

Notação Polonesa:
imix;is;=s1;=x1;f<s9=ss1+;.f<x9=xx1+;.r0;

Código Intermediário:
MAIN:
```

```
TABSIMB.addInt(
x
)
TABSIMB.addInt(
s
)
TABSIMB.update(
s,
1
)
TABSIMB.update(
x,
1
)
IF:
s<9
if(!s<9) goto ENDIF
TABSIMB.update(
s,
EXPR:
s
push(s)
1
push(1)
ADD
ENDEXPR:
)
ENDIF:
IF:
x<9
if(!x<9) goto ENDIF
TABSIMB.update(
x,
EXPR:
x
push(x)
1
push(1)
ADD
ENDEXPR:
)
ENDIF:
END:
```

Código Objeto:

```
INT 0 3
INT 0 1
INT 0 1
LIT 0 1
STO 0 3
LIT 0 1
STO 0 2
LOD 0 3
LIT 0 9
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 17
LOD 0 3
LIT 0 1
OPR 0 2
STO 0 3
LOD 0 2
LIT 0 9
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 27
LOD 0 2
LIT 0 1
OPR 0 2
STO 0 2
OPR 0 0
```

G.2. Programa com while’s sequenciais

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;is;s=1;x=1;w((s<9)){s=(s+1);.}w((x<9)){x=(x+1);.}r(0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 10969 10993 10994
10995 10996 10997 131917 131941 131965 131966 131967 131968
131969 1583581 1583605 1583629 1583630 1583631 1583632 1583633
1583634 1583635 1583636 19003573 19003574 19003575 19003576
```

```
19003577 19003609 19003610 19003611 19003612 19003613 19003633
19003634 19003635 19003636 19003637 19003638 19003639 19003640
228042889 228042901 228042913 228043309 228043333 228043334
228043335 228043336 228043337 228043357 228043621 228043622
228043623 228043624 228043625 228043657 228043658 228043659
228043660 228043661 228043681 228043682 228043683 228043684
228043685 2736514669 2736520009 2736520021 2736520033 2736523465
2736523477 2736523489 2736523885 2736523909 2736523910 2736523911
2736523912 2736523913 2736523933 2736524197 2773469037 2773510509
2773515849 2773515861 2773515873 3217419117
M i m ( ) { S } i X ; S x i X ; S s X = E ; C s 1 X = E ; C x 1 w
( E ) { C } C ( E B E ) X = E ; C w ( E ) { C } C X < 9 s ( E B E
) . ( E B E ) X = E ; C r ( E ) ; s X + 1 X < 9 x ( E B E ) . 0 s
x X + 1 x

Árvore ASA:
0 1 2 3 7 10 11 31 34 35 103 104 106 107 319 320 322 323 967 968
969 970 971 2902 2903 2905 2906 2911 2912 2913 2914 2915 8719
8720 8734 8735 8737 8738 8743 26215 26216
i m i ; x i ; s = ; s 1 = ; x 1 w . < = ; w . s 9 s + < = ; r ; s
1 x 9 x + 0 x 1

Notação Polonesa:
imix;is;=s1;=x1;w<s9=ss1+;.w<x9=xx1+;.r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
)
TABSIMB.addInt(
s
)
TABSIMB.update(
s,
1
)
TABSIMB.update(
x,
1
)
WHILE:
s<9
if(!s<9) goto ENDWHILE
TABSIMB.update(
```

```
s,
EXPR:
s
push(s)
1
push(1)
ADD
ENDEXPR:
)
goto WHILE:
ENDWHILE:
WHILE:
x<9
if(!x<9) goto ENDWHILE
TABSIMB.update(
x,
EXPR:
x
push(x)
1
push(1)
ADD
ENDEXPR:
)
goto WHILE:
ENDWHILE:
RETURN(
0
)
END:

Código Objeto:
INT 0 3
INT 0 1
INT 0 1
LIT 0 1
STO 0 3
LIT 0 1
STO 0 2
LOD 0 3
LIT 0 9
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 18
```

```
LOD 0 3
LIT 0 1
OPR 0 2
STO 0 3
JMP 0 7
LOD 0 2
LIT 0 9
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 29
LOD 0 2
LIT 0 1
OPR 0 2
STO 0 2
JMP 0 18
OPR 0 0
```

G.3. Programa com for's sequenciais

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;is;x=3;o(x=0;(s<6);x=(x+1)){s=(s+2);.}o(x=0;(s<6);x
=(x+1)){s=(s+2);.}r(0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 10969 10993
10994 10995 10996 10997 131917 131941 131965 131966 131967
131968 131969 131970 131971 131972 131973 131974 131975
131976 1583605 1583606 1583607 1583629 1583630 1583631
1583632 1583633 1583653 1583654 1583655 1583689 1583690
1583691 1583692 1583693 1583713 1583714 1583715 1583716
1583717 1583718 1583719 1583720 1583721 1583722 1583723
1583724 19003261 19003285 19003561 19003573 19003585
19003837 19003861 19003862 19003863 19003864 19003865
19004269 19004293 19004294 19004295 19004296 19004297
19004317 19004581 19004582 19004583 19004605 19004606
19004607 19004608 19004609 19004629 19004630 19004631
19004665 19004666 19004667 19004668 19004669 19004689
19004690 19004691 19004692 19004693 228042733 228046345
228046357 228046369 228051529 228051541 228051553 228054973
```

```
228054997 228055273 228055285 228055297 228055549 228055573
228055574 228055575 228055576 228055577 228055981 228056005
228056006 228056007 228056008 228056009 228056029 228056293
2736556141 2736618349 2736663277 2736666889 2736666901
2736666913 2736672073 2736672085 2736672097 2775231597
2775293805
M i m ( ) { S } i X ; S x i X ; S s X = E ; C x 3 o ( A ; E
; A ) { C } C X = E ( E B E ) X = E X = E ; C o ( A ; E ; A
) { C } C x 0 X < 6 x ( E B E ) s ( E B E ) . X = E ( E B E
) X = E X = E ; C r ( E ) ; s X + 1 X + 2 x 0 X < 6 x ( E B
E ) s ( E B E ) . 0 x s s X + 1 X + 2 x s

Árvore ASA:
0 1 2 3 7 10 11 31 34 35 103 104 106 107 319 320 321 322
323 958 959 961 962 964 965 967 968 969 970 971 2884 2885
2887 2888 2896 2897 2902 2903 2905 2906 2908 2909 2911 8662
8663 8716 8717 8719 8720 8728 8729 25990 25991 26158 26159
78478 78479
i m i ; x i ; s = ; x 3 o . = ; = o . x 0 < ; x + = ; = r ;
s 6 = ; x 1 x 0 < ; x + 0 s + s 6 = ; x 1 s 2 s + s 2

Notação Polonesa:
imix;is;=x3;o=x0;<s6;=ss2+;=xx1+.o=x0;<s6;=ss2+;=xx1+.r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
)
TABSIMB.addInt(
s
)
TABSIMB.update(
x,
3
)
TABSIMB.update(
x,
0
)
FOR:
s<6
if(!s<6) goto ENDFOR
TABSIMB.update(
s,
```

```
EXPR:
s
push(s)
2
push(2)
ADD
ENDEXPR:
)
TABSIMB.update(
x,
EXPR:
x
push(x)
1
push(1)
ADD
ENDEXPR:
)
goto FOR
ENDFOR:
TABSIMB.update(
x,
0
)
FOR:
s<6
if(!s<6) goto ENDFOR
TABSIMB.update(
s,
EXPR:
s
push(s)
2
push(2)
ADD
ENDEXPR:
)
TABSIMB.update(
x,
EXPR:
x
push(x)
1
push(1)
ADD
```

```
ENDEXPR:
)
goto FOR
ENDFOR:
RETURN(
0
)
END:

Código Objeto:
INT 0 3
INT 0 1
INT 0 1
LIT 0 3
STO 0 2
LIT 0 0
STO 0 2
LOD 0 3
LIT 0 6
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 22
LOD 0 3
LIT 0 2
OPR 0 2
STO 0 3
LOD 0 2
LIT 0 1
OPR 0 2
STO 0 2
JMP 0 7
LIT 0 0
STO 0 2
LOD 0 3
LIT 0 6
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 39
LOD 0 3
LIT 0 2
OPR 0 2
STO 0 3
LOD 0 2
```

```
LIT 0 1
OPR 0 2
STO 0 2
JMP 0 24
OPR 0 0
```

G.4. Programa com if's aninhados

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;is;x=3;s=2;f((x>2)){f((s<3)){x=(5+2);s=(3+2);.}.}r(
0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 10969 10993
10994 10995 10996 10997 131917 131941 131965 131966 131967
131968 131969 1583581 1583605 1583629 1583630 1583631
1583632 1583633 1583634 1583635 1583636 19003573 19003574
19003575 19003576 19003577 19003609 19003610 19003611
19003612 19003613 19003614 19003615 19003616 19003633
19003634 19003635 19003636 19003637 228042889 228042901
228042913 228043333 228043334 228043335 228043336 228043337
228043369 228043370 228043371 228043372 228043373 228043393
228043621 2736514669 2736520009 2736520021 2736520033
2736520429 2736520453 2736520454 2736520455 2736520456
2736520457 2736520477 2736520478 2736520479 2736520480
2736520481 2773469037 2773474377 2773474389 2773474401
2773474653 2773474677 2773474678 2773474679 2773474680
2773474681 2773474701 3216925065 3216925077 3216925089
M i m ( ) { S } i X ; S x i X ; S s X = E ; C x 3 X = E ; C
s 2 f ( E ) { C } C ( E B E ) f ( E ) { C } C r ( E ) ; X >
2 ( E B E ) X = E ; C . 0 x X < 3 x ( E B E ) X = E ; C s 5
+ 2 s ( E B E ) . 3 + 2

Árvore ASA:
0 1 2 3 7 10 11 31 34 35 103 104 106 107 319 320 322 323
967 968 969 970 971 2902 2903 2905 2906 2907 2911 8716 8717
8719 8720 8722 8723 26161 26162 26167 26168 78505 78506
i m i ; x i ; s = ; x 3 = ; s 2 f . > f . r ; x 2 < = ; 0 s
3 x + = ; 5 2 s + 3 2
```

```
Notação Polonesa:
imix;is;=x3;=s2;f>x2f<s3=x52+;=s32+;..r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
)
TABSIMB.addInt(
s
)
TABSIMB.update(
x,
3
)
TABSIMB.update(
s,
2
)
IF:
x>2
if(!x>2) goto ENDIF
IF:
s<3
if(!s<3) goto ENDIF
TABSIMB.update(
x,
EXPR:
5
push(5)
2
push(2)
ADD
ENDEXPR:
)
TABSIMB.update(
s,
EXPR:
3
push(3)
2
push(2)
ADD
ENDEXPR:
)
```

```
ENDIF:
ENDIF:
END:

Código Objeto:
INT 0 3
INT 0 1
INT 0 1
LIT 0 3
STO 0 2
LIT 0 2
STO 0 3
LOD 0 2
LIT 0 2
OPR 0 12
LIT 0 0
OPR 0 8
JPC 0 27
LOD 0 3
LIT 0 3
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 27
LIT 0 5
LIT 0 2
OPR 0 2
STO 0 2
LIT 0 3
LIT 0 2
OPR 0 2
STO 0 3
OPR 0 0
```

G.5. Programa com while’s aninhados

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;is;x=1;s=1;w((x<5)){w((s<5)){s=(s+1);.}s=1;x=(x+1);
.}r(0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 10969 10993
10994 10995 10996 10997 131917 131941 131965 131966 131967
131968 131969 1583581 1583605 1583629 1583630 1583631
1583632 1583633 1583634 1583635 1583636 19003573 19003574
19003575 19003576 19003577 19003609 19003610 19003611
19003612 19003613 19003614 19003615 19003616 19003633
19003634 19003635 19003636 19003637 228042889 228042901
228042913 228043333 228043334 228043335 228043336 228043337
228043369 228043370 228043371 228043372 228043373 228043393
228043394 228043395 228043396 228043397 228043621
2736514669 2736520009 2736520021 2736520033 2736520429
2736520453 2736520454 2736520455 2736520456 2736520457
2736520477 2736520717 2736520741 2736520765 2736520766
2736520767 2736520768 2736520769 2773469037 2773474377
2773474389 2773474401 2773478109 2773478133 2773478134
2773478135 2773478136 2773478137 2773478157 3216921453
3216966537 3216966549 3216966561 4243860077
M i m ( ) { S } i X ; S x i X ; S s X = E ; C x 1 X = E ; C
s 1 w ( E ) { C } C ( E B E ) w ( E ) { C } C r ( E ) ; X <
5 ( E B E ) X = E ; C X = E ; C 0 x X < 5 s ( E B E ) . s 1
X = E ; C s X + 1 x ( E B E ) . s X + 1 x

Árvore ASA:
0 1 2 3 7 10 11 31 34 35 103 104 106 107 319 320 322 323
967 968 969 970 971 2902 2903 2905 2906 2907 2908 2909 2911
8716 8717 8719 8720 8725 8726 8728 8729 26161 26162 26185
26186 78559 78560
i m i ; x i ; s = ; x 1 = ; s 1 w . < w . r ; x 5 < = ; = ;
0 s 5 s + s 1 = ; s 1 x + x 1

Notação Polonesa:
imix;is;=x1;=s1;w<x5w<s5=ss1+;.=s1;=xx1+;.r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
)
TABSIMB.addInt(
s
)
TABSIMB.update(
x,
1
```

```
)
TABSIMB.update(
s,
1
)
WHILE:
x<5
if(!x<5) goto ENDWHILE
WHILE:
s<5
if(!s<5) goto ENDWHILE
TABSIMB.update(
s,
EXPR:
s
push(s)
1
push(1)
ADD
ENDEXPR:
)
goto WHILE:
ENDWHILE:
TABSIMB.update(
s,
1
)
TABSIMB.update(
x,
EXPR:
x
push(x)
1
push(1)
ADD
ENDEXPR:
)
goto WHILE:
ENDWHILE:
RETURN(
0
)
END:
```

Código Objeto:

```
INT 0 3
INT 0 1
INT 0 1
LIT 0 1
STO 0 2
LIT 0 1
STO 0 3
LOD 0 2
LIT 0 5
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 31
LOD 0 3
LIT 0 5
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 24
LOD 0 3
LIT 0 1
OPR 0 2
STO 0 3
JMP 0 13
LIT 0 1
STO 0 3
LOD 0 2
LIT 0 1
OPR 0 2
STO 0 2
JMP 0 7
OPR 0 0
```

G.6. Programa com for’s aninhados

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;is;x=3;o(x=0;(s<6);x=(x+1)){o(x=0;(s<6);x=(x+1)){s=
(s+2);.}.}r(0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
```

```
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 10969 10993
10994 10995 10996 10997 131917 131941 131965 131966 131967
131968 131969 131970 131971 131972 131973 131974 131975
131976 1583605 1583606 1583607 1583629 1583630 1583631
1583632 1583633 1583653 1583654 1583655 1583689 1583690
1583691 1583692 1583693 1583694 1583695 1583696 1583697
1583698 1583699 1583700 1583713 1583714 1583715 1583716
1583717 19003261 19003285 19003561 19003573 19003585
19003837 19003861 19003862 19003863 19003864 19003865
19004293 19004294 19004295 19004317 19004318 19004319
19004320 19004321 19004341 19004342 19004343 19004377
19004378 19004379 19004380 19004381 19004401 19004581
228042733 228046345 228046357 228046369 228051517 228051541
228051817 228051829 228051841 228052093 228052117 228052118
228052119 228052120 228052121 228052525 228052549 228052550
228052551 228052552 228052553 228052573 2736556141
2736621805 2736625417 2736625429 2736625441 2736630601
2736630613 2736630625 2774733933 2774796141
M i m ( ) { S } i X ; S x i X ; S s X = E ; C x 3 o ( A ; E
; A ) { C } C X = E ( E B E ) X = E o ( A ; E ; A ) { C } C
r ( E ) ; x 0 X < 6 x ( E B E ) X = E ( E B E ) X = E X = E
; C . 0 s X + 1 x 0 X < 6 x ( E B E ) s ( E B E ) . x s X +
1 X + 2 x s

Árvore ASA:
0 1 2 3 7 10 11 31 34 35 103 104 106 107 319 320 321 322
323 958 959 961 962 964 965 967 2884 2885 2887 2888 2896
2897 8662 8663 8664 25987 25988 25990 25991 25993 25994
77971 77972 77974 77975 77983 77984 233923 233924 701773
701774
i m i ; x i ; s = ; x 3 o . = ; = r ; x 0 < ; x + 0 s 6 o .
x 1 = ; = x 0 < ; x + s 6 = ; x 1 s + s 2

Notação Polonesa:
imix;is;=x3;o=x0;<s6;o=x0;<s6;=ss2+;=xx1+.=xx1+.r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
)
TABSIMB.addInt(
s
)
TABSIMB.update(
```

```
x,
3
)
TABSIMB.update(
x,
0
)
FOR:
s<6
if(!s<6) goto ENDFOR
TABSIMB.update(
x,
0
)
FOR:
s<6
if(!s<6) goto ENDFOR
TABSIMB.update(
s,
EXPR:
s
push(s)
2
push(2)
ADD
ENDEXPR:
)
TABSIMB.update(
x,
EXPR:
x
push(x)
1
push(1)
ADD
ENDEXPR:
)
goto FOR
ENDFOR:
TABSIMB.update(
x,
EXPR:
x
push(x)
1
```

```
push(1)
ADD
ENDEXPR:
)
goto FOR
ENDFOR:
RETURN(
0
)
END:

Código Objeto:
INT 0 3
INT 0 1
INT 0 1
LIT 0 3
STO 0 2
LIT 0 0
STO 0 2
LOD 0 3
LIT 0 6
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 35
LIT 0 0
STO 0 2
LOD 0 3
LIT 0 6
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 30
LOD 0 3
LIT 0 2
OPR 0 2
STO 0 3
LOD 0 2
LIT 0 1
OPR 0 2
STO 0 2
JMP 0 15
LOD 0 2
LIT 0 1
OPR 0 2
```

```
STO 0 2
JMP 0 7
OPR 0 0
```

G.7. Programa contendo if com while aninhado

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;x=3;f((x<8)){w((x<8)){x=(x+1);.}.}r(0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 917 10957
10981 11005 11006 11007 11008 11009 11010 11011 11012
132085 132086 132087 132088 132089 132121 132122 132123
132124 132125 132126 132127 132128 132145 132146 132147
132148 132149 1585033 1585045 1585057 1585477 1585478
1585479 1585480 1585481 1585513 1585514 1585515 1585516
1585517 1585537 1585765 19020397 19025737 19025749 19025761
19026157 19026181 19026182 19026183 19026184 19026185
19026205 228308845 228314185 228314197 228314209 2739770221
M i m ( ) { S } i X ; S x X = E ; C x 3 f ( E ) { C } C ( E
B E ) w ( E ) { C } C r ( E ) ; X < 8 ( E B E ) X = E ; C .
0 x X < 8 x ( E B E ) . x X + 1 x

Árvore ASA:
0 1 2 3 7 10 11 31 32 34 35 103 104 105 106 107 310 311 313
314 315 319 940 941 943 944 2833 2834
i m i ; x = ; x 3 f . < w . r ; x 8 < = ; 0 x 8 x + x 1

Notação Polonesa:
imix;=x3;f<x8w<x8=xx1+;..r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
)
TABSIMB.update(
x,
3
)
```

```
IF:
x<8
if(!x<8) goto ENDIF
WHILE:
x<8
if(!x<8) goto ENDWHILE
TABSIMB.update(
x,
EXPR:
x
push(x)
1
push(1)
ADD
ENDEXPR:
)
goto WHILE:
ENDWHILE:
ENDIF:
END:

Código Objeto:
INT 0 3
INT 0 1
LIT 0 3
STO 0 2
LOD 0 2
LIT 0 8
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 21
LOD 0 2
LIT 0 8
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 21
LOD 0 2
LIT 0 1
OPR 0 2
STO 0 2
JMP 0 10
OPR 0 0
```

G.8. Programa contendo if com for aninhado

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;is;f((x<5)){o(x=0;(x<5);x=(x+1)){s=(s+2);.}.}r(0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 10969 10993
10994 10995 10996 10997 10998 10999 11000 131941 131942
131943 131944 131945 131977 131978 131979 131980 131981
131982 131983 131984 131985 131986 131987 131988 132001
132002 132003 132004 132005 1583305 1583317 1583329 1583749
1583750 1583751 1583773 1583774 1583775 1583776 1583777
1583797 1583798 1583799 1583833 1583834 1583835 1583836
1583837 1583857 1584037 18999661 19004989 19005013 19005289
19005301 19005313 19005565 19005589 19005590 19005591
19005592 19005593 19005997 19006021 19006022 19006023
19006024 19006025 19006045 228063469 228067081 228067093
228067105 228072265 228072277 228072289 2736804973
2736867181
M i m ( ) { S } i X ; S x i X ; S s f ( E ) { C } C ( E B E
) o ( A ; E ; A ) { C } C r ( E ) ; X < 5 X = E ( E B E ) X
= E X = E ; C . 0 x x 0 X < 5 x ( E B E ) s ( E B E ) . x X
+ 1 X + 2 x s

Árvore ASA:
0 1 2 3 7 10 11 31 34 35 103 104 105 106 107 310 311 313
314 315 319 940 941 943 944 946 947 2830 2831 2833 2834
2842 2843 8500 8501 25504 25505
i m i ; x i ; s f . < o . r ; x 5 = ; = 0 x 0 < ; x + x 5 =
; x 1 s + s 2

Notação Polonesa:
imix;is;f<x5o=x0;<x5;=ss2+;=xx1+..r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
```

```
)
TABSIMB.addInt(
s
)
IF:
x<5
if(!x<5) goto ENDIF
TABSIMB.update(
x,
0
)
FOR:
x<5
if(!x<5) goto ENDFOR
TABSIMB.update(
s,
EXPR:
s
push(s)
2
push(2)
ADD
ENDEXPR:
)
TABSIMB.update(
x,
EXPR:
x
push(x)
1
push(1)
ADD
ENDEXPR:
)
goto FOR
ENDFOR:
ENDIF:
END:

Código Objeto:
INT 0 3
INT 0 1
INT 0 1
LOD 0 2
LIT 0 5
```

```
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 26
LIT 0 0
STO 0 2
LOD 0 2
LIT 0 5
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 26
LOD 0 3
LIT 0 2
OPR 0 2
STO 0 3
LOD 0 2
LIT 0 1
OPR 0 2
STO 0 2
JMP 0 11
OPR 0 0
```

G.9. Programa contendo while com if aninhado

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;x=3;w((x<8)){f((x<5)){x=(x+1);.}.}r(0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 917 10957
10981 11005 11006 11007 11008 11009 11010 11011 11012
132085 132086 132087 132088 132089 132121 132122 132123
132124 132125 132126 132127 132128 132145 132146 132147
132148 132149 1585033 1585045 1585057 1585477 1585478
1585479 1585480 1585481 1585513 1585514 1585515 1585516
1585517 1585537 1585765 19020397 19025737 19025749 19025761
19026157 19026181 19026182 19026183 19026184 19026185
19026205 228308845 228314185 228314197 228314209 2739770221
M i m ( ) { S } i X ; S x X = E ; C x 3 w ( E ) { C } C ( E
B E ) f ( E ) { C } C r ( E ) ; X < 8 ( E B E ) X = E ; C .
```

```
0 x X < 5 x ( E B E ) . x X + 1 x

Árvore ASA:
0 1 2 3 7 10 11 31 32 34 35 103 104 105 106 107 310 311 313
314 315 319 940 941 943 944 2833 2834
i m i ; x = ; x 3 w . < f . r ; x 8 < = ; 0 x 5 x + x 1

Notação Polonesa:
imix;=x3;w<x8f<x5=xx1+;..r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
)
TABSIMB.update(
x,
3
)
WHILE:
x<8
if(!x<8) goto ENDWHILE
IF:
x<5
if(!x<5) goto ENDIF
TABSIMB.update(
x,
EXPR:
x
push(x)
1
push(1)
ADD
ENDEXPR:
)
ENDIF:
goto WHILE:
ENDWHILE:
END:

Código Objeto:
INT 0 3
INT 0 1
LIT 0 3
STO 0 2
```

```
LOD 0 2
LIT 0 8
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 21
LOD 0 2
LIT 0 5
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 20
LOD 0 2
LIT 0 1
OPR 0 2
STO 0 2
JMP 0 4
OPR 0 0
```

G.10. Programa contendo while com for aninhado

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;iy;is;x=0;w((x<3)){o(y=0;(y<3);y=(y+1)){s=(s+3);.}x
=(x+1);.}r(0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 10969 10993
10994 10995 10996 131929 131953 131954 131955 131956 131957
1583437 1583461 1583485 1583486 1583487 1583488 1583489
1583490 1583491 1583492 19001845 19001846 19001847 19001848
19001849 19001881 19001882 19001883 19001884 19001885
19001886 19001887 19001888 19001889 19001890 19001891
19001892 19001905 19001906 19001907 19001908 19001909
228022153 228022165 228022177 228022597 228022598 228022599
228022621 228022622 228022623 228022624 228022625 228022645
228022646 228022647 228022681 228022682 228022683 228022684
228022685 228022705 228022706 228022707 228022708 228022709
228022885 2736265837 2736271165 2736271189 2736271465
2736271477 2736271489 2736271741 2736271765 2736271766
2736271767 2736271768 2736271769 2736272173 2736272197
```

```
2736272198 2736272199 2736272200 2736272201 2736272221
2736272461 2736272485 2736272486 2736272487 2736272488
2736272489 2736272509 2770486509 2770490121 2770490133
2770490145 2770495305 2770495317 2770495329 2770498761
2770498773 2770498785 3181110381 3181172589 3181214061
M i m ( ) { S } i X ; S x i X ; S y i X ; S s X = E ; C x 0
w ( E ) { C } C ( E B E ) o ( A ; E ; A ) { C } C r ( E ) ;
X < 3 X = E ( E B E ) X = E X = E ; C X = E ; C 0 x y 0 X <
3 y ( E B E ) s ( E B E ) . x ( E B E ) . y X + 1 X + 3 X +
1 y s x

Árvore ASA:
0 1 2 3 7 10 11 31 34 35 103 106 107 319 320 322 323 967
968 969 970 971 2902 2903 2905 2906 2907 2908 2909 2911
8716 8717 8719 8720 8722 8723 8725 8726 26158 26159 26161
26162 26170 26171 26179 26180 78484 78485 235456 235457
i m i ; x i ; y i ; s = ; x 0 w . < o . r ; x 3 = ; = = ; 0
y 0 < ; y + x + y 3 = ; y 1 x 1 s + s 3

Notação Polonesa:
imix;iy;is;=x0;w<x3o=y0;<y3;=ss3+;=yy1+.=xx1+;.r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
)
TABSIMB.addInt(
y
)
TABSIMB.addInt(
s
)
TABSIMB.update(
x,
0
)
WHILE:
x<3
if(!x<3) goto ENDWHILE
TABSIMB.update(
y,
0
)
FOR:
```

```
y<3
if(!y<3) goto ENDFOR
TABSIMB.update(
s,
EXPR:
s
push(s)
3
push(3)
ADD
ENDEXPR:
)
TABSIMB.update(
y,
EXPR:
y
push(y)
1
push(1)
ADD
ENDEXPR:
)
goto FOR
ENDFOR:
TABSIMB.update(
x,
EXPR:
x
push(x)
1
push(1)
ADD
ENDEXPR:
)
goto WHILE:
ENDWHILE:
RETURN(
0
)
END:

Código Objeto:
INT 0 3
INT 0 1
INT 0 1
```

```
INT 0 1
LIT 0 0
STO 0 2
LOD 0 2
LIT 0 3
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 34
LIT 0 0
STO 0 3
LOD 0 3
LIT 0 3
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 29
LOD 0 4
LIT 0 3
OPR 0 2
STO 0 4
LOD 0 3
LIT 0 1
OPR 0 2
STO 0 3
JMP 0 14
LOD 0 2
LIT 0 1
OPR 0 2
STO 0 2
JMP 0 6
OPR 0 0
```

G.11. Programa contendo for com if aninhado

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;iy;y=0;o(x=0;(x<8);x=(x+1)){f((x<5)){y=(y+1);.}.}r(
0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
```

```
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 10969 10993
10994 10995 10996 10997 131917 131941 131965 131966 131967
131968 131969 131970 131971 131972 131973 131974 131975
131976 1583605 1583606 1583607 1583629 1583630 1583631
1583632 1583633 1583653 1583654 1583655 1583689 1583690
1583691 1583692 1583693 1583694 1583695 1583696 1583713
1583714 1583715 1583716 1583717 19003261 19003285 19003561
19003573 19003585 19003837 19003861 19003862 19003863
19003864 19003865 19004293 19004294 19004295 19004296
19004297 19004329 19004330 19004331 19004332 19004333
19004353 19004581 228042733 228046345 228046357 228046369
228051529 228051541 228051553 228051949 228051973 228051974
228051975 228051976 228051977 228051997 2736556141
2736618349 2736623689 2736623701 2736623713 2774713197
M i m ( ) { S } i X ; S x i X ; S y X = E ; C y 0 o ( A ; E
; A ) { C } C X = E ( E B E ) X = E f ( E ) { C } C r ( E )
; x 0 X < 8 x ( E B E ) ( E B E ) X = E ; C . 0 x X + 1 X <
5 y ( E B E ) . x x X + 1 y

Árvore ASA:
0 1 2 3 7 10 11 31 34 35 103 104 106 107 319 320 321 322
323 958 959 961 962 964 965 967 2884 2885 2887 2888 2896
2897 8662 8663 8664 25987 25988 25990 25991 77974 77975
i m i ; x i ; y = ; y 0 o . = ; = r ; x 0 < ; x + 0 x 8 f .
x 1 < = ; x 5 y + y 1

Notação Polonesa:
imix;iy;=y0;o=x0;<x8;f<x5=yy1+;.=xx1+.r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
)
TABSIMB.addInt(
y
)
TABSIMB.update(
y,
0
)
TABSIMB.update(
x,
0
)
```

```
FOR:
x<8
if(!x<8) goto ENDFOR
IF:
x<5
if(!x<5) goto ENDIF
TABSIMB.update(
y,
EXPR:
y
push(y)
1
push(1)
ADD
ENDEXPR:
)
ENDIF:
goto FOR
ENDFOR:
END:

Código Objeto:
INT 0 3
INT 0 1
INT 0 1
LIT 0 0
STO 0 3
LIT 0 0
STO 0 2
LOD 0 2
LIT 0 8
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 24
LOD 0 2
LIT 0 5
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 23
LOD 0 3
LIT 0 1
OPR 0 2
STO 0 3
```

```
JMP 0 7
OPR 0 0
```

G.12. Programa contendo for com while aninhado

Conteúdo do arquivo de entrada para a realização do teste:

```
im(){ix;iy;y=0;o(x=0;(x<3);x=(x+1)){y=0;w((y<5)){y=(y+1);.}
.}r(0);}
```

Abaixo são apresentadas as saídas de cada módulo desenvolvido para a entrada acima.

```
Árvore ADA:
0 1 2 3 4 5 6 7 73 74 75 76 889 913 914 915 916 10969 10993
10994 10995 10996 10997 131917 131941 131965 131966 131967
131968 131969 131970 131971 131972 131973 131974 131975
131976 1583605 1583606 1583607 1583629 1583630 1583631
1583632 1583633 1583653 1583654 1583655 1583689 1583690
1583691 1583692 1583693 1583713 1583714 1583715 1583716
1583717 19003261 19003285 19003561 19003573 19003585
19003837 19003861 19003862 19003863 19003864 19003865
19004269 19004293 19004317 19004318 19004319 19004320
19004321 19004322 19004323 19004324 19004581 228042733
228046345 228046357 228046369 228051829 228051830 228051831
228051832 228051833 228051865 228051866 228051867 228051868
228051869 228051889 2736556141 2736621961 2736621973
2736621985 2736622381 2736622405 2736622406 2736622407
2736622408 2736622409 2736622429 2774692461 2774697801
2774697813 2774697825 3231602541
M i m ( ) { S } i X ; S x i X ; S y X = E ; C y 0 o ( A ; E
; A ) { C } C X = E ( E B E ) X = E X = E ; C r ( E ) ; x 0
X < 3 x ( E B E ) y 0 w ( E ) { C } C 0 x X + 1 ( E B E ) X
= E ; C . x X < 5 y ( E B E ) . y X + 1 y

Árvore ASA:
0 1 2 3 7 10 11 31 34 35 103 104 106 107 319 320 321 322
323 958 959 961 962 964 965 967 2884 2885 2887 2888 2896
2897 8662 8663 8665 8666 25996 25997 25998 77989 77990
77992 77993 233980 233981
i m i ; x i ; y = ; y 0 o . = ; = r ; x 0 < ; x + 0 x 3 = ;
x 1 y 0 w . < = ; y 5 y + y 1

Notação Polonesa:
```

```
imix;iy;=y0;o=x0;<x3;=y0;w<y5=yy1+;.=xx1+.r0;

Código Intermediário:
MAIN:
TABSIMB.addInt(
x
)
TABSIMB.addInt(
y
)
TABSIMB.update(
y,
0
)
TABSIMB.update(
x,
0
)
FOR:
x<3
if(!x<3) goto ENDFOR
TABSIMB.update(
y,
0
)
WHILE:
y<5
if(!y<5) goto ENDWHILE
TABSIMB.update(
y,
EXPR:
y
push(y)
1
push(1)
ADD
ENDEXPR:
)
goto WHILE:
ENDWHILE:
TABSIMB.update(
x,
EXPR:
x
push(x)
```

```
1
push(1)
ADD
ENDEXPR:
)
goto FOR
ENDFOR:
RETURN(
0
)
END:

Código Objeto:
INT 0 3
INT 0 1
INT 0 1
LIT 0 0
STO 0 3
LIT 0 0
STO 0 2
LOD 0 2
LIT 0 3
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 31
LIT 0 0
STO 0 3
LOD 0 3
LIT 0 5
OPR 0 10
LIT 0 0
OPR 0 8
JPC 0 26
LOD 0 3
LIT 0 1
OPR 0 2
STO 0 3
JMP 0 15
LOD 0 2
LIT 0 1
OPR 0 2
STO 0 2
JMP 0 7
OPR 0 0
```

Referências

1. SETHI, Ravi; ULLMAN, Jeffrey D.; MONICA S. LAM. **Compiladores: princípios, técnicas e ferramentas.** Pearson Addison Wesley, 2008.
2. **Construção de Compiladores - Wikilivros** <https://pt.wikibooks.org> Acessado em 24/10/18
3. **Geração de Código Objeto** <http://www.ybadoo.com.br/tutoriais/cmp/08/CMP08.pdf> Acessado em 24/10/18
4. Link para arquivos: https://drive.google.com/open?id=1pxLrVYbr4HVeNnB9MlBQCkAMMbkexomP