Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

SCC0206 - Introdução à Compilação
Professora - Sandra Maria Aluísio

FRANKIE

Implementação de uma linguagem de programação híbrida entre Pascal e C

Parte II

Parte I Revisada e Implementação do Analisador Sintático

*Projeto de Curso Desenvolvido pelos Alunos (**GRUPO 7**)*

Ubiratan F. Soares (5634292)
Humberto Yagi (5634420)
Ulisses F. Soares (5377365)

São Carlos, 25 de abril de 2011

Notas sobre a Atualização desse Documento

Foi decidido pelos autores desse trabalho que o desenvolvimento da documentação referente à implementação do projeto da disciplina de Compiladores seria feito de maneira incremental, de maneira que cada nova entrega estenda e atualize anterior.

No intuito de facilitar a percepção em relação às adições a esse texto e facilitar o trabalho de correção, segue um registro das últimas modificações :

Adição do Índice Remissivo Completo (pg. 2)
Correções e Adaptações Menores na Seção 3 (pgs x a y)
Comentários sobre a transição da FRANKIE para LL1 e resolução de problemas
Atualização da Seção 5, considerando as modificações para FRANKIE em gramática LL1
Correções na Seção 6, de acordo com as observações da correção da entrega anterior
Seção 7 reescrita
Adicionadas Seções 8 e 9

1. Introdução

Frankie é uma linguagem de programação híbrida entre Pascal e C, proposta como projeto prático da disciplina de Introdução à Compilação. Trata-se, portanto, de uma linguagem procedimental, dotada de um conjunto mínimo de funcionalidades que torne possível a sua utilização enquanto linguagem de alto nível e viável a sua implementação em um curso de um semestre.

A caráter de tornar a implementação de cada grupo participante da disciplina única, foram propostas extensões individuais por grupo, de maneira que esse trabalho contempla a Frankie estendida com a estrutura de dados composta **UNION**, sintática e semanticamente herdada da linguagem C. Demais extensões não estão prevista ao longo desse trabalho.

2. Visão Geral da Linguagem

A linguagem Frankie possui em sua forma mínima a definição de programa e de bloco, além da gramática de expressões herdadas da linguagem Pascal. Além disso, também vêm do Pascal a definição de procedimentos, bem como a passagem e lista de parâmetros dos mesmos. Não há o suporte há funções na proposta minimalista da linguagem.

Por outro lado, a forma como tipos nativos dados são declarados é herdada da linguagem C. Na versão mínima, Frankie suporta somente os tipos inteiro (***int***) e booleano (***boolean***). Em nossa implementação, suportará também o tipo composto Union. Além disso, o controle de fluxo e repetição admite duas declarações também herdadas da linguagem C, nas instruções ***if*** e ***while***.

Demais aspectos da linguagem foram decididos pelo grupo e serão apresentados na sequência.

3. Especificações Gerais do Compilador

Tratamos a seguir as especificações gerais do Compilador da linguagem Frankie. Essa especificação está de acordo com a proposta da docente, sendo subdividida nos 10 itens que se seguem.

3.1. Sobre a Finalidade do Compilador

O Compilador a ser desenvolvido se destina a compilar programas escrito na linguagem **Frankie** a ser implementada em suas diversas etapas ao longo da disciplina de Introdução à Compilação. Assim, o compilador será desenvolvido com a finalidade didática, para de modo a apresentar aos alunos os conceitos da área de Compiladores e ensinar técnicas que abordem as principais fases processo de tradução.

3.2. Sobre os Requisitos de Velocidade do Compilador

A eficiência do processo de compilação não é o foco da implementação em questão. Dessa forma, heurísticas de *parsing* eficiente, uso de estruturas de dados auxiliares otimizadas, algoritmos para geração de representações intermediárias eficientes e outros não serão explorados nesse trabalho.

A finalidade do projeto proposto com a Frankie é de construir um software de intuito didático, com tempo de desenvolvimento adequado. Assim, serão utilizadas ferramentas para automatizar partes do processo, ficando em segundo plano o compromisso com desempenho. Além disso, a intenção é que o compilador gere código transportável, uma vez que a linguagem-objeto será gerada para ser interpretada por uma máquina hipotética (ver item 3.5).

Finalmente, deve-se destacar que o número de passos que será utilizado no processo de tradução é um item que de certa forma agrega eficiência ao processo a ser implementado. (ver item 3.9)

3.3 Sobre os Requisitos de Velocidade do Código-Objeto

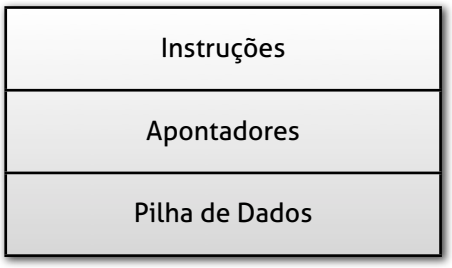
Sendo esse um projeto de caráter didático, não pretende-se alcançar requisitos de desempenho do código-objeto gerado pelo Compilador Frankie. Assim, a geração de código se dará para a linguagem de máquina de uma máquina hipotética, sem compromissos de desempenho para execução. (ver item 3.5)

3.4. Sobre a Máquina Hospedeira do Compilador

O compilador a ser implementado para essa linguagem é do tipo cruzado (*Cross Compiler*), de maneira que o hardware de execução representado pela máquina virtual onde residirá o código-objeto é distinto do ambiente onde reside o tradutor. O segmento de memória onde residirá o código-objeto é formado por três regiões, como ilustrado a seguir :

3.5. Sobre a Máquina-Objeto para Execução de Programas

A máquina-objeto para a qual os programas em Frankie serão traduzidos é a **MEPA** (***M****áquina de* ***E****xecução para* ***PA****scal*). Essa máquina hipotética, como descrita em [2], consiste de uma máquina à pilha dotada de um conjunto limitado de registradores e um conjunto adequado de instruções de máquina.



- ***Região de Instruções*** : contém as instruções geradas pelo Compilador da linguagem Frankie.
- ***Pilha de Dados*** : contém os valores manipulados pelas instruções. Supõe-se que contém valores inteiros ou não definidos.
- ***Apontadores*** : contém endereços dos dados da região da Pilha ou valores indefinidos.

A máquina MEPA possui 2 registrados especiais, com funções bem definidas : ***contador de programa*** e ***topo de pilha***. Esses registradores são responsáveis pelo funcionamento adequado da MEPA enquanto hardware hipotético.

3.6. Sobre o Software de Sistema para a Execução de Programas

O ambiente de sistema para a execução do compilador Frankie é multi-plataforma. A implementação usará, dentre outras ferramentas, o *compiler compiler* **JavaCC**, que executa em ambiente **Java** e gera código nessa mesma linguagem. Dessa forma, qualquer sistema que possua compatibilidade com a versão ***1.6_x*** do ***JRE (Java Runtime Engine)*** ou outra versão mais atual poderá compilar programas na linguagem Frankie.

Além disso, assume-se que esse Compilador estará acessível por interface de linha de comando e aceitará como entrada arquivos de dados na extensão ***.frankie***, que identificarão os códigos-fonte e produzirá o código-objeto apropriado.

3.7. Sobre o Ambiente de Execução da Máquina-Objeto

Sendo a MEPA uma máquina hipotética, há diversas maneiras de tratá-la em um ambiente de execução real. Em [2], são propostas diversas estratégias associadas ao ambiente do Compilador para o problema em questão. Uma delas consiste no uso de um macro-montador.

Segundo essa abordagem, cada instrução da MEPA será tratada como uma ***macro-instrução***. A implementação da MEPA, portanto, consiste na implementação dessas macros, que nada mais são do rotinas de tradução das instruções MEPA conhecidas para uma linguagem de montagem da máquina hospedeira.

A vantagem da estratégia da macro-montagem é que, embora utilize um passo adicional do ponto de vista da tradução, é didaticamente atraente, uma vez que não é preciso conhecer a linguagem de montagem para qual as instruções serão traduzidas e depois montadas.

3.8. Sobre o Grau de Complexidade da Linguagem-Fonte

A linguagem Frankie é simples do ponto de vista da complexidade. Além de suportar somente dois tipos nativos de dados, duas instruções de fluxo de controle, não suportar ponteiros e funções, ela é completamente procedimental, como as linguagens-mãe Pascal e C. Além disso, as rotinas de entrada e saída são tratadas como nativas da linguagem, de maneira que não é necessário a tradução de um comando de ES a ser interceptado por uma biblioteca ou rotina do sistema.

Por esses e outros aspectos, a Frankie pode ser considerada de complexidade simples.

3.9. Sobre o Número de Passos no Processo de Compilação

O compilador será implementado com um tradutor de um passo. Dessa forma, o código-objeto será gerado conforme o código-fonte em Frankie for processado pela compilador, sem retrocessos para otimizações. Dessa forma, o processo de tradução ganha em eficiência, ainda que não sejam introduzidos ganhos de performance com otimizações globais ou locais, uma vez que as fases intermediárias são entrelaçadas. Essa abordagem será dirigida por sintaxe[1].

3.10. Sobre as Técnicas de Implementação

Para auxiliar no desenvolvimento desse trabalho e conforme comentado no item 3.6, foi adotada nesse trabalho a ferramenta de *compiler compiler* **JavaCC**. Essa ferramenta oferece suporte ao menos a duas fases do processo de compilação : a análise léxica (*tokenizer*) e a análise sintática (*parser*).

Uma vez que a ferramenta JavaCC gera código para executar tais funções em linguagem Java, o restante do compilador FRANKIE também será implementado nessa mesma linguagem.

A especificação léxica se dá segundo expressões regulares estendidas, de acordo com a entrada padrão JavaCC. Por outro lado, a Análise Sintática será feita de forma descendente recursiva, de forma que a gramática da linguagem Frankie descrita em EBNF foi adaptada para ser LL1, especificação aceitável por esse *compiler compiler*. Maiores detalhes são fornecidos na próxima seção.

Critérios para a implementação dos demais passos do processo de tradução (Análise Semântica, Geração de Código e afins) serão decididos ao longo do desenvolvimento desse trabalho.

4. Especificação Formal da Linguagem Frankie

A especificação formal em notação EBNF da linguagem Frankie, com a extensão de tipo de dados composto UNION é fornecida no **Apêndice A** que acompanha esse documento. As regras da gramática foram reescritas de forma a tornar os nomes das produções mais padronizados e práticos para o grupo.

5. Comentários sobre a Extensão para a Linguagem

A especificação formal para a extensão da Frankie proposta para o grupo foi o tipo de dados composto UNION, com formação herdada da linguagem C. Dessa forma, foi consultada a forma EBNF da gramática da linguagem C, fornecida em documento impresso durante as aulas. Dessa forma, traçamos um comparativo entre as regras da gramática da Frankie envolvidas na extensão a seguir.

Frankie Básica

```
<TYPE-SPECIFIER> := ("boolean" | "int" | UNION-SPECIFIER)
```

Gramática da linguagem C

```
1.    <TYPE-SPECIFIER> := ("void" | "char" | "short" |  ... | <STRUCT-OR-UNION_SPECIFIER>)
2.
3.    <STRUCT-OR- UNION-SPECIFIER> :=
4.                          ("struct" | "union") (
5.                          IDENTIFIER? "{" <STRUCT-DECLARATION>+ "}" | <IDENTIFIER>  )
6.
7.    <STRUCT-DECLARATION> := ( <TYPE-SPECIFIER> | <TYPE-QUALIFIER>)+ <STRUCT-DECLARATOR>?
8.
9.    <TYPE-QUALIFIER> :=  ("const" | "volatile")
10.
11.   <STRUC-DECLARATOR> := <DECLARATOR> | <DECLARATOR>? ":" <CONSTANT-EXPRESSION>
12.
13.   <DECLARATOR> :=
14.             <POINTER>? (IDENTIFIER | "(" <DECLARATOR ")" )  (
15.                         "[" <CONSTANT-EXPRESSION>? "]" |
16.                         "(" <PARAMETER-TYPE-LIST ")" |
17.                         "(" <IDENTIFIER> ")"
18.             )*
19.
20.   // Outras declarações
```

Frankie Estendida

```
<TYPE-SPECIFIER> := ("boolean" | "int" | UNION-SPECIFIER)

<UNION-SPECIFIER> := "union" <IDENTIFIER> "begin" <UNION-DECLARATION> "end" ";"

<UNION-DECLARATION> := {<IDENTIFIER> : <TYPE-SPECIFIER> ";" }
```

As regras fornecidas pela gramática de C foram adaptadas segundo os seguintes critérios:

- "**STRUCT-OR-UNION"** refere-se somente a **"UNION"** *(linhas 1 e 3)*
- Frankie não suporta o segundo identificador pós-declaração *(linha 5)*
- As construções de bloco seguem Pascal; **"{"** e **"}"** foram trocados por **"begin"** e **"end"** *(linha 5)*
- Frankie não suporta qualificadores de tipo *(linha 9)*
- Frankie não suporta expressões constantes *(linha 10)*
- Frankie não suporta ponteiros *(linha 14)*
- PARAMETER-TYPE-LIST é convertido em **{<IDENTIFIER> : <TYPE-SPECIFIER> ";" }** de maneira a compor a regra de formação dos membros do tipo Union. *(linha 16)*

Dessa forma, uma declaração de UNION em Frankie será como a seguir:

```
union  myUnion
        begin
                x : int;
                y : boolean;
        end;
```

6. Considerações Léxicas

Para a implementação do analisador léxico (*tokenizer*), primeira etapa do processo de compilação, seguem-se algumas considerações.

6.1. Conjunto de Palavras Reservadas

As palavras reservadas são os símbolos imediatos a serem reconhecidos na elaboração de qualquer linguagem de programação, de maneira que a Frankie possui um conjunto de x palavras reservadas. Na etapa de análise léxica agora implementada, não há distinção entre essas palavras e identificadores bem formadas, mas essa poderá ser incorporada no futuro mediante vantagens de inserção antecipada na tabela de símbolos.

A tabela de palavras reservadas da Frankie segue abaixo :

Palavras Reservadas	program	procedure	var
int	boolean	begin	end
if	else	while	or
div	and	not	union
TRUE	FALSE	read	write

6.2. Constantes Permitidas

Como a Frankie só admite dois tipos primitivos de dados, só são permitidas atribuições constantes booleanas e inteiras. As atribuições booleanas podem assumir **“true”** e **“false”**. Já os inteiros serão de 16 bits, o que implica no maior inteiro valendo **+32767** e o menor assumindo **-32768**. Números além desse escopo não serão reconhecidos como válidos pelo *tokenizer*.

6.3. Caracteres Não-Imprimíveis e Símbolos Especiais

Os caracteres **“/n”**, **“/t”**, **“LF”**, **“CR”** e outros de controle de linhas utilizados em editores de texto ASCII não serão imprimidos na etapa de análise léxica. Além disso, são tratados como símbolos especiais os seguinte caracteres listado na tabela a seguir. A Frankie não suporta caracteres fora da codificação ASCII.

.	**;**	**>**	**>=**	**:**
<	**<=**	**:=**	**(**	**)**
+	**-**	*****	**,**	**=**

6.4. Tipos e Tratamento dos Comentários

A Frankie suporta dois tipos de comentário : de linha e de bloco

Comentários de linha são obtidos iniciando o excerto com **"//"**, assim como na linguagem C. Já comentários de bloco são utilizados através dos delimitadores **"{"** e **"}"** , assim como na linguagem Pascal.

```
1.  //   Isso é uma linha comentada
2.
3.  {
4.       Esse é um bloco comentado.
5.       Ele contempla múltiplas linhas.
6.  }
```

A implementação do analisador léxico permite que a Frankie tenha suporte a comentários aninhados. Exemplos de uso desse tipos de comentário e sua validação pelo *tokenizer* estão nos arquivos de implementação entregues no email da disciplina.

6.5. Tamanho Máximo e Formação dos Identificadores

A Frankie suporta identificadores formados por até 32 caracteres. Esses caracteres podem ser letras minúsculas ou maiúsculas, números ou o caractere **"_"** (*underline*). O analisador léxico implementado é capaz de invalidar identificadores que utilizem caracteres fora desse cojunto (por exemplo, **"@"**), mas não é capaz de invalidar um identificador mal formado no caso de começar por número (por exemplo, **1a**).

Esse grupo chegou à conclusão que, através do *compiler compiler* JavaCC, não é possível reconhecer identificadores iniciados por um número como inválidos. Esse será um erro tratado, portanto, em nível sintático.

Por fim, a Frankie será implementada como *case sensitive*, de maneira que nenhum tratamento em relação ao uso de caixa alta ou baixa foi implementado em nível léxico.

7. Comentários sobre a Implementação Corrigida do Analisador Léxico

Para a implementação do analisador léxico foi utilizada a versão mais recente do JavaCC (**javacc-5.0**), além do ambiente **Java 1.6.0_24**.

No intuito de auxiliar a correção dessa etapa do projeto, foram desenvolvidos *scripts* de ambiente UNIX para compilação e execução automática de testes. Para os casos de teste, foram desenvolvidos diversos programas em Frankie, com base na adaptação de algoritmos clássicos implementados em Pascal ou em C, com QuickSort, Fibonacci e outros. Também implementados programas de exemplo para os principais erros léxicos reconhecíveis pelo JavaCC, a saber:

- Comentário de bloco em aberto
- Identificador formado por caractere inválido
- Constante além do limite permitido
- Identificar além do limite permitido
- Número inteiro mal formado

Para os comando **read( )** e **write( )**, foram reimplementados os reconhecedores léxicos, de modo que cadeias de caracteres internas aos parâmetros não são agora mais suportadas.

8. Considerações Sintáticas

Para a implementação do analisador sintático (*parser*), segunda etapa do processo de compilação, seguem-se algumas considerações.

8.1. Tipos de Dados Elementares

Conforme descrito na seção 2, FRANKIE suportará dois tipos de dados básicos : inteiros (*int*) e lógicos (*boolean*). Além disso, conforme extensão proposta pela linguagem, o usuário pode definir sua própria estrutura de dados UNION.

8.2. Operações sobre Tipos Básicos

FRANKIE oferece suporte à operações básicas associadas a cada tipo de dado, a saber:

Operações Aritméticas : adição, subtração, multiplicação, divisão inteira
Operações Lógicas : conjunção, disjunção, negação

8.3. Sobre Tipos Abstratos de Dados

Sendo uma linguagem com proposta didática, FRANKIE não oferece suporte a tipos abstratos de dados.

8.4. Comandos de Entrada e Saída

Em FRANKIE, entrada e saída de programas serão realizadas através dos procedimentos pré-definidos read e write. Apenas variáveis inteiras podem ser lidas.

8.5. Funções Pré-Definidas

FRANKIE não oferece nenhuma funções pré-definida.

8.6. Fluxo de Controle ao Nível de Cláusulas

Ao nível de cláusulas, FRANKIE oferece em sua versão mínima não estendida apenas um comando condicional (if) e um comando repetitivo (while).

8.7. Fluxo de Controle ao Nível de Programa

Ao nível de programa, FRANKIE oferece suporte completo a procedimentos, de maneira que um programa pode ser facilmente modularizado em subrotinas.

9. Comentários sobre a Implementação do Analisador Sintático

A partir da gramática da FRANKIE modificada para ser LL(1), a ferramenta JavaCC implementou também o parser de maneira automática. Ficou evidenciado para o grupo a separação entra a análise léxica e sintática, de maneira que a primeira é subrotina da segunda.

10. Referências

1. **Sandra Maria Aluísio** - Notas de Aula. São Carlos, 2011.
2. **T. Kowaltowski** - *Implementação de Linguagens de Programação*. Guanabara Dois, 1983
3. **A. V. Aho, M. S. Lam, J. D. Ullman** - *Compilers: Principles, Techniques, and Tools (2nd edition).* Addison Wesley, 2006
4. **Thiago S. Pardo** - *Notas de Aula*. São Carlos, 2010.
5. **A. W. Appel and J. Palsberg** - *Modern Compiler Implementation in Java (2nd edition)*. Cambridge University Press, 2002