

**Tabela de Símbolos: Implementação
e Avaliação das Principais Operações
em Diferentes Paradigmas de Programação**

**Yêdda Aires Duarte Silva
Mariza A. S. Bigonha
Roberto S. Bigonha**

LLP04/99

Relatório Final: POC2

Sumário

1	Introdução	4
2	Motivação e Objetivos	4
3	Revisão da Literatura	5
3.1	Tabelas de Símbolos	5
3.2	O Sistema Delphi	6
3.3	A linguagem C	7
3.4	O Sistema Java	8
4	Especificação das Organizações de Tabela de Símbolos para Linguagens com Estruturas de Blocos	9
4.1	Introdução	9
4.2	Organização Linear	12
4.2.1	Lista Não-Ordenada	12
4.2.2	Lista Ordenada	14
4.3	Organização na Forma de Árvore Binária	15
4.4	Organização na Forma de Floresta de Árvores Binárias	17
4.5	Organização na Forma de HASH	20
4.6	Conclusão	22
5	Implementação das Organizações da Tabela de Símbolos para Linguagens com Estruturas de Blocos	22
5.1	Introdução	22
5.2	Estruturas de Dados	23
5.3	Implementação em Delphi	23
5.3.1	Interfaces	24
5.3.2	Janelas de Interface	24
5.4	Implementações em C e Java	26
5.4.1	Interfaces em C e Java	26
6	Análise dos Resultados	27
6.1	Testes	27
6.1.1	Teste 01	27
6.1.2	Teste 02	30
6.1.3	Teste 03	35
6.2	Gráficos	40
6.2.1	Gráficos Referentes ao Teste01 para a Linguagem Delphi	40
6.2.2	Gráficos Referentes ao Teste01 para a Linguagem C	42
6.2.3	Gráficos Referentes ao Teste01 para a Linguagem Java	44
6.2.4	Gráficos Referentes ao Teste01 com o Desempenho entre as Linguagens Delphi, C e Java	48
6.2.5	Gráficos Referentes ao Teste02 para a Linguagem Delphi	52
6.2.6	Gráficos Referentes ao Teste02 para a Linguagem C	54
6.2.7	Gráficos Referentes ao Teste02 para a Linguagem Java	56

6.2.8	Gráficos Referentes ao Teste02 com o Desempenho entre as Linguagens Delphi, C e Java	60
6.2.9	Gráficos Referentes ao Teste03 para a Linguagem Delphi	64
6.2.10	Gráficos Referentes ao Teste03 para a Linguagem C	66
6.2.11	Gráficos Referentes ao Teste03 para a Linguagem Java	68
6.2.12	Gráficos Referentes ao Teste03 com o Desempenho entre as Linguagens Delphi, C e Java	72
6.3	Avaliação Final	76
7	Conclusão	77
8	Bibliografia	77
A	Fontes dos Programas na Linguagem Delphi	78
A.1	Abordagem Linear na Linguagem Delphi	78
A.1.1	Lista Não Ordenada	78
A.1.2	Lista Ordenada	83
A.2	Abordagem Árvore Binária na Linguagem Delphi	88
A.3	Abordagem Floresta de Árvore Binária na Linguagem Delphi	94
A.4	Abordagem <i>Hash</i> na Linguagem Delphi	100
B	Fontes dos Programas na Linguagem C	107
B.1	Abordagem Linear na Linguagem C	107
B.1.1	Lista Não Ordenada	107
B.1.2	Lista Ordenada	110
B.2	Abordagem Árvore Binária na Linguagem C	114
B.3	Abordagem Floresta de Árvore Binária na Linguagem C	118
B.4	Abordagem <i>Hash</i> na Linguagem C	122
C	Fontes dos Programas na Linguagem JAVA	127
C.1	Abordagem Linear na Linguagem JAVA	127
C.1.1	Lista Não Ordenada	127
C.1.2	Lista Ordenada	129
C.2	Abordagem Árvore Binária na Linguagem JAVA	133
C.3	Abordagem Floresta de Árvores Binárias na Linguagem JAVA	136
C.4	Abordagem <i>Hash</i> na Linguagem JAVA	139

1 Introdução

Este documento relata as decisões de projeto, provê os algoritmos, suas respectivas implementações e os resultados obtidos a partir do desenvolvimento de um pacote *Tabela de Símbolos* para auxiliar o projetista de compiladores na implementação de linguagens com estrutura de blocos. Este pacote foi implementado nas linguagens Delphi [8], C [6] e Java [2] [8] e contém as operações para entrar em um bloco, sair de um bloco, pesquisar por um nome, adicionar novas informações ao nome, remover e inserir nomes.

Um compilador lê um programa escrito em uma linguagem fonte e o traduz em um programa equivalente em uma outra linguagem, a linguagem alvo. Normalmente, ele é organizado em duas partes: a análise e a síntese. Durante a análise, o programa fonte é dividido nas partes constituintes e cria uma representação intermediária do mesmo. A parte de síntese constrói o programa alvo desejado, a partir da representação intermediária [1] [4] [5]. Uma função essencial do compilador é registrar os identificadores usados no programa fonte e coletar as informações sobre os seus diversos atributos. Esta função é desempenhada pela tabela de símbolos.

Este relatório está organizado da seguinte forma: na Seção 2 são descritos os objetivos e a motivação para o desenvolvimento deste trabalho. A Seção 3 mostra a revisão da literatura. Nela são apresentados aspectos fundamentais das implementações, bem como das linguagens abordadas. A Seção 4 mostra a representação da tabela de símbolos, apresentando os algoritmos com suas principais operações, nas abordagens citadas na Seção 2. Todos os algoritmos mostrados nesta seção foram projetados pelo professor Roberto da Silva Bigonha [3]. A Seção 5 apresenta a implementação dos algoritmos mostrados na Seção 4 para as diferentes abordagens. A Seção 6 apresenta uma avaliação dos resultados obtidos após a implementação das operações de Inserção, Remoção e Pesquisa de símbolo, Entrada e Saída de Bloco para as diferentes abordagens mostradas na Seção 4. A Seção 7 apresenta a conclusão deste relatório seguida dos Apêndices A, B e C com as listagens dos fontes nas linguagens Delphi, C e Java.

2 Motivação e Objetivos

As informações contidas na tabela de símbolos são usadas em várias fases do compilador:

- Durante a análise semântica:
verifica-se se o uso dos nomes estão consistentes com suas declarações implícitas ou explícitas.
- Durante a operação de código:
as informações da tabela de símbolos são utilizadas para saber o quanto e que tipo de armazenamento deve ser alocado para o nome durante a execução do programa.
- Na recuperação de erro:
usa-se as informações da tabela de símbolos para evitar repetição de mensagens de erro do tipo "variável A definida" .

- Na otimização de código:
para descobrir variáveis temporárias usadas mais de uma vez.

Dada a importância da tabela de símbolos seria interessante que um estudo fosse feito afim de avaliar as diversas abordagens existentes para que o projetista de compiladores pudesse escolher com segurança uma determinada organização. Este foi um dos motivos que nos levou a propor este projeto. Outro é que não existe um pacote que implemente as operações relacionadas na Seção 1 nas linguagens mencionadas. Levando em consideração os motivos expostos e o fato de que o ponto mais importante que determina a escolha de uma das abordagens descritas na Seção 1 é a rapidez com que um nome pode ser inserido e acessado na tabela, nosso objetivo ao propor o projeto *Tabela de Símbolos* foi produzir um pacote para as operações mencionadas nas três linguagens de programação C, Delphi e Java, e, principalmente obter como subproduto, a avaliação dessas abordagens. Para atingir nosso objetivo efetuamos as seguintes tarefas:

1. implementamos os algoritmos para as principais operações realizadas pela tabela de símbolos, Abrir e Fechar Blocos, Inserir, Remover e Pesquisar por símbolos, nas organizações: Linear, Árvore Binária, Floresta de Árvores Binárias e *Hash* em três linguagens, C, Delphi e Java;
2. avaliamos os algoritmos nas diferentes linguagens. Nesta parte nosso interesse se concentrou no desempenho das diferentes abordagens $\times$ diferentes linguagens, em relação ao tempo de acesso as variáveis e o espaço gasto.

3 Revisão da Literatura

Nesta seção são abordados os aspectos mais importantes da tabela de símbolos e são mostrados também as principais características das linguagens Delphi, C e Java usadas no desenvolvimento do trabalho proposto.

3.1 Tabelas de Símbolos

Uma tabela de símbolos é uma estrutura de dados usada pelo compilador para monitorar as informações de escopo e das amarrações a respeito dos nomes [1, 5]. Ela consiste, em geral em duas partes básicas: o nome do símbolo e seus atributos. O nome é geralmente um *string* de caracteres alfanuméricos. Os atributos de um nome podem ser: tipo, nível, etc. O tipo, por exemplo, se refere aos tipos primitivos de uma linguagem de programação: *real*, *integer*, *char*, etc. O nível refere-se ao local onde o nome aparece no programa e conseqüentemente ao conjunto de blocos do programa fonte onde o nome é válido. As informações são introduzidas na tabela de símbolos em vários momentos.

Informações sobre o programa fonte podem ser inseridas na tabela de símbolos durante a análise léxica e análise semântica ou somente durante a análise semântica. Durante a análise léxica, o analisador léxico reconhece seqüências de letras e dígitos da entrada para determinar se uma palavra-chave ou um nome foi coletado. Neste ponto ele pode ou não introduzir as mesmas na tabela de símbolos. É mais eficiente instalar as palavras-chaves em uma tabela a parte, deixando a tabela de símbolos somente para identificadores. Com essa abordagem, as palavras-chave precisam estar na tabela antes da análise léxica começar. Se na linguagem

para a qual está se construindo um compilador, as palavras-chave não são reservadas, é essencial que as mesmas sejam introduzidas na tabela com um aviso de seus possíveis usos como palavras-chave.

Os atributos de um nome são introduzidos em resposta às declarações, que podem ser implícitas. Por exemplo, os rótulos são frequentemente identificadores seguidos por um ponto-e-vírgula e dessa forma uma ação associada ao reconhecimento de um identificador pode introduzir esse fato na tabela de símbolos. Similarmente, a sintaxe das declarações dos procedimentos especifica que certos identificadores são parâmetros formais. A tabela de símbolos contém também informações sobre a memória reservada para os identificadores. Para nomes de procedimentos, ela armazena o número e os tipos de seus argumentos, o método de transmissão de cada parâmetro e o tipo retornado. Em uma linguagem estaticamente tipada, esses identificadores devem aparecer nas declarações e à medida que os mesmos são lidos do programa, eles são armazenados na tabela de símbolos. As principais operações que uma tabela de símbolos deve realizar são:

- determinar se um nome está na tabela;
- adicionar nomes à tabela;
- ter acesso aos atributos de um nome;
- adicionar novas informações a um nome;
- remover nomes da tabela.

3.2 O Sistema Delphi

Delphi [8] é um sistema de desenvolvimento de aplicações baseado em componentes que permite que um programador construa poderosos programas em ambiente *Window* com o mínimo de código. O programador utiliza elementos do ambiente de programação Delphi para criar a interface de sua própria aplicação. O editor de código é um editor muito poderoso. Ele permite ao desenvolvedor escrever sua aplicação e oferece muitos recursos. Quando é aberto um novo projeto, Delphi adiciona a página no editor de código para a *unit* do *form* inicial. Em tempo de compilação, caso tenha ocorrido um erro, Delphi age da seguinte forma:

- mostra o erro em uma linha de *status* no editor de código;
- limpa a linha que possui o erro.

Um projeto em Delphi é uma coleção de arquivos que compõem uma aplicação ou uma biblioteca disponível. Quando o programador primeiramente, inicia Delphi, um novo projeto padrão é aberto. O desenvolvedor pode então trabalhar com o projeto padrão, abrir outro projeto existente, ou começar um projeto diferente usando um dos *templates* de projetos predefinidos em Delphi. O *template* usado no projeto padrão é o *Blank*. O programador pode mudá-lo usando o *dialog box Gallery Options*. Assim, os novos projetos futuros, terão o *template* especificado ao iniciar a plataforma. O *form* é o ponto principal em uma aplicação de Delphi. Adicionando componentes ao *form*, editando suas respectivas propriedades, ou codificando a *unit* do *form* o programador está editando o *form*. *Forms* são guardados em dois arquivos separados:

- o arquivo *.DFM* guarda a imagem binária do *form*. Qualquer mudança feita nas propriedades visuais do *form*, como largura, cor, bordas e etc, são guardadas no arquivo *.DFM*.
- o arquivo de *unit* *.PAS* guarda o código do arquivo *.DFM*. No arquivo *.PAS*, o programador escreve eventos que especificam como o *form* e seus componentes se comportarão na execução da aplicação.

Delphi mantém os arquivos binários e de código sincronizados, quando um *form* é criado ou modificado. Se um novo *form* é adicionado à aplicação, Delphi cria um arquivo de *unit* associado e o adiciona nas cláusulas *uses* do arquivo de projeto. Componentes são blocos construídos e formam a base da biblioteca visual de componentes de Delphi. Eles podem ser visuais ou não. Cada componente tem atributos específicos que permitem ao programador controlar a aplicação. Esses atributos são propriedades, eventos, e métodos. As vantagens de se trabalhar com Delphi, que explica a escolha dessa linguagem são:

- a facilidade de programação na linguagem Pascal, que é a base da plataforma Delphi;
- a alta tecnologia dos componentes de Delphi;
- a estreita integração com a programação *Windows*;
- o suporte a banco de dados;
- um compilador rápido;
- a abordagem baseada em formulários e orientada a objeto;
- a disponibilidade de códigos-fonte das bibliotecas;
- o editor, o depurador, o *browser*, e outras ferramentas presentes na linguagem;

Ao longo deste documento, a ferramenta Delphi será tratada como linguagem, objetivando o melhor entendimento do mesmo.

3.3 A linguagem C

C foi inventada e implementada primeiramente por Dennis Ritchie em um DEC PDP-11, que utilizava o sistema operacional UNIX [6]. C é o resultado de um processo de desenvolvimento que começou com uma linguagem mais antiga, BCPL, que ainda está em uso, em sua forma original, na Europa. BCPL foi desenvolvida por Martin Richards e influenciou uma linguagem chamada B, inventada por Ken Thompson. Na década de 70, B levou ao desenvolvimento de C.

Com a popularidade dos microcomputadores, um grande número de implementações de C foi criado. Quase que por milagre, os códigos-fontes aceitos por essas implementações eram altamente compatíveis. Porém, por não existir nenhum padrão, havia discrepâncias. Para remediar essa situação, o ANSI estabeleceu, no verão de 1983, um comitê para criar um padrão que definiria de uma vez por todas a linguagem C.

C é tratada como uma linguagem de médio nível porque combina elementos de linguagem de alto nível com a funcionalidade da linguagem *assembly*. C permite a manipulação de *bits*,

bytes e endereços, os elementos básicos com os quais o computador funciona. Um código escrito em C é muito portátil. Embora C tenha cinco tipos de dados internos, ela não é uma linguagem única em tipos de dados como Pascal e ADA.

C permite quase todas converções de tipos. Por exemplo, os tipos caractere e inteiro podem ser livremente misturados na maioria das expressões. C não efetua nenhuma verificação em tempo de execução, como a validação dos limites das matrizes. Esses tipos de verificações são de responsabilidade do programador. Um outro aspecto importante de C é que ela tem apenas 32 palavras-chaves.

Embora o termo linguagem estruturada em blocos não seja rigorosamente aplicável a C, ela é normalmente referida simplesmente como linguagem estruturada. C tem muitas semelhanças com outras linguagens estruturadas, como ALGOL, Pascal e Modula-2. O principal componente estrutural de C é a função, a sub-rotina isolada de C. Em C, funções são os blocos de construção em que toda a atividade do programa ocorre. Elas admitem que se defina e codifique separadamente as diferentes tarefas de um programa, permitindo, então, que seu programa seja modular. Após uma função ter sido criada, pode-se ter certeza que ela trabalhará adequadamente em várias situações, sem criar efeitos inesperados em outras partes do programa. O fato de poder criar funções isoladas é extremamente importante em projetos maiores onde um código de um programador não deve afetar acidentalmente o de outro.

Uma outra maneira de estruturar e compartimentalizar o código em C é por meio do uso de blocos de código. Um bloco de código é um grupo de comandos de programa conectado logicamente é tratado como uma unidade. Um bloco de código é criado colocando-se uma sequência de comandos entre chaves.

Todo compilador C vem com uma biblioteca C padrão de funções que realizam as tarefas necessárias mais comuns. Em algumas implementações de C, a biblioteca aparece em um grande arquivo; em outras, ela está contida em muitos arquivos menores, uma organização que aumenta a eficiência e a praticidade. Quando chama uma função que não faz parte do programa escrito, o compilador C memoriza seu nome. Mais tarde, o ligador combina o código escrito com o código-objeto já encontrado na biblioteca padrão. Esse processo é chamado de ligação. Alguns compiladores C tem seu próprio ligador, enquanto outros usam o ligador padrão fornecido pelo seu sistema operacional.

C permite que um programa seja contido em muitos arquivos e que cada arquivo seja compilado separadamente. Uma vez que todos os arquivos estejam compilados, eles são ligados com qualquer rotina de biblioteca, para formar um código-objeto completo. A vantagem da compilação separada é que, se houver uma mudança no código de um arquivo, não será necessária a recompilação do programa todo. Em tudo, menos nos projetos mais simples, isso economiza um tempo considerável.

3.4 O Sistema Java

Para muitos, a linguagem Java [2] é conhecida primeiramente como uma ferramenta para criar *applets* para a *World Wide Web*. Applet é o termo que Java utiliza para uma mini-aplicação que é executada dentro de uma página Web. Uma *applet* pode desempenhar tarefas e interagir com o usuário em uma página visualizada por seu *browser* sem utilizar recursos do servidor Web após ter sido descarregada. Algumas *applets* podem, logicamente, interagir com o servidor para seus próprios propósitos. A linguagem de programação Java foi muito bem recebida pela comunidade mundial dos desenvolvedores de *software* e provedores da

Internet. Os usuários da *Internet* e *World Wide Web* beneficiam-se do acesso a aplicações seguras, independentes de plataforma, que podem vir de qualquer lugar na Internet [2] [8]. Desenvolvedores de *software* que criam aplicações em Java beneficiam-se do desenvolvimento e da codificação, sem necessidade de portar suas aplicações para diversas plataformas de *software* e *hardware*. Java, é de fato, valiosa para ambientes de rede distribuídos como a Web. Contudo, Java vai além deste domínio a fim de fornecer uma poderosa linguagem de programação de propósito geral, adequada à construção de uma série de aplicações que não dependam das características da rede ou para quem as quer por diferentes razões. A capacidade de Java em executar código em computadores remotos de forma segura é um requisito crítico para muitas empresas. Outros grupos também utilizam Java como uma linguagem de programação de propósito geral para projetos onde a independência de máquina não é tão importante. Java é fácil de programar e as características de segurança ajudam a produzir rapidamente código sem problemas. Alguns erros comuns em outras linguagens de programação nunca ocorrem devido as suas características como a coleta de lixo e às referências de segurança de tipo.

Aplicações modernas de rede e de interface gráfica de usuário que devem atender a múltiplas tarefas simultaneamente são fornecidas pelo suporte de Java para multilinha, enquanto o mecanismo de tratamento de exceções facilita a tarefa de lidar com as condições de erro.

Embora suas ferramentas sejam poderosas, a linguagem Java é uma linguagem simples, na qual programadores podem rapidamente se tornar peritos. Java é designada com o máximo de portabilidade, e é especificamente designada para ter tão poucas dependências de implementação quanto possível. Um *int*, por exemplo, é um inteiro em complemento de dois de 32 bits com sinal em todas as implementações Java, qualquer que seja a arquitetura da CPU em que o programa Java possa ser executado. Java compartilha muitas características de linguagem comuns à maioria das linguagens de programação em uso hoje em dia. Contudo, diferentemente de C e de C++ [6], Java oferece gerenciamento automático de armazenamento e tratamento de exceções, integrado com suporte a multilinha.

4 Especificação das Organizações de Tabela de Símbolos para Linguagens com Estruturas de Blocos

4.1 Introdução

Há mais de uma forma possível para representar uma tabela de símbolos. As duas formas mais comuns são apresentadas na Figura 1. A primeira mostra uma tabela com dois campos: o nome e o atributo. Esta forma é mais apropriada para linguagens que possuem um modesto limite superior no comprimento do identificador. Por exemplo, no Fortran da IBM, o identificador tem no máximo oito caracteres. A segunda forma é mais apropriada para as linguagens C, Pascal, PL/1, Algol, etc, porque não existe um limite máximo para os identificadores. O limite depende da implementação. O nome é dividido em dois campos, onde um deles é apontador para um outro arranjo de caracteres, a tabela de nomes propriamente dita, e outro contém o tamanho da *string* do identificador. Esta indireção permite que

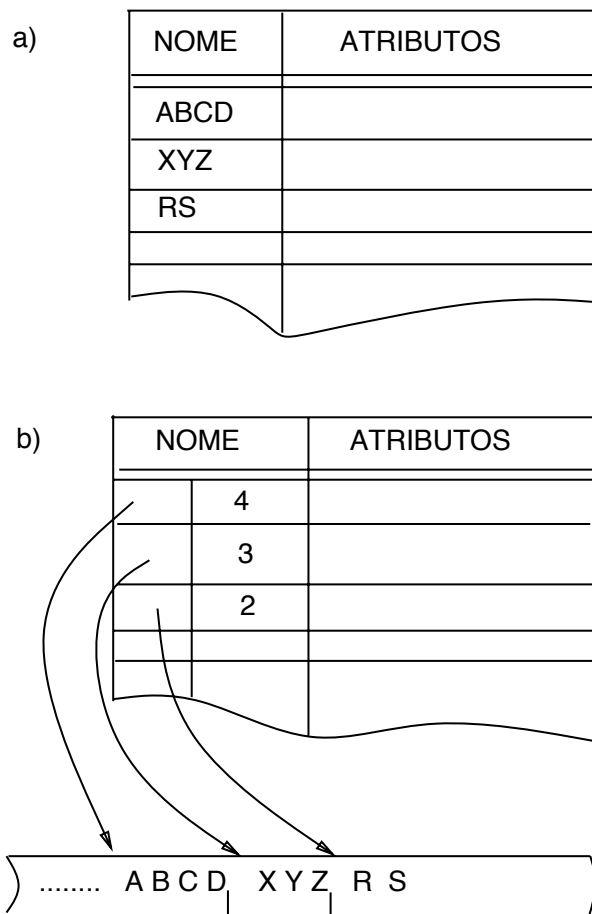


Figura 1: Formas de Representar Tabelas de Símbolos

o tamanho do campo nome da tabela de Símbolos permaneça uma constante. Usaremos em nossas implementações, uma tabela de símbolos semelhante a segunda forma descrita. Ela será um registro com os campos:

- nome: contém o nome do identificador;
- atributo: reservado para outros atributos do nome;
- nível: guarda em qual nível o identificador foi declarado;
- esquerda e direita: aparecem nas abordagens árvore binária e floresta de árvores binárias, e representam os filhos a esquerda e a direita de um nodo.

Quando um compilador inicia a tradução de um programa fonte, a tabela de símbolos está vazia. Com o progresso da tradução, símbolos são adicionados na tabela, mas antes de instalá-los é necessário que se pesquise pela sua ocorrência na tabela, portanto um símbolo só é inserido se ele não estiver presente na tabela, dentro daquele escopo. Como a maior parte

do tempo de tradução é gasto nesta parte, é importante estudar as formas de organização que providenciam uma pesquisa eficiente. Neste projeto, comparamos os diferentes métodos em relação ao tempo de pesquisa.

Existem várias alternativas para implementar uma tabela de símbolos. A mais simples é usar um arranjo linear de registros com um registro para cada nome. Outras alternativas são: Árvores Binárias de Pesquisa, Tabelas *Hash* e Florestas de Árvores Binárias [1] [3], [4], [5], [7]. Usaremos um pequeno exemplo para ilustrar a aplicação dos algoritmos nas diferentes abordagens. Este exemplo pode ser visualizado na Figura 2. Como a representação das variáveis na tabela de símbolos muda de acordo com a abordagem enfocada, sua representação será dada no momento apropriado. As próximas seções mostra as operações para Inserir, Remover, Pesquisar por símbolos e Entrar e Sair de Blocos nas quatro alternativas apresentadas neste parágrafo.

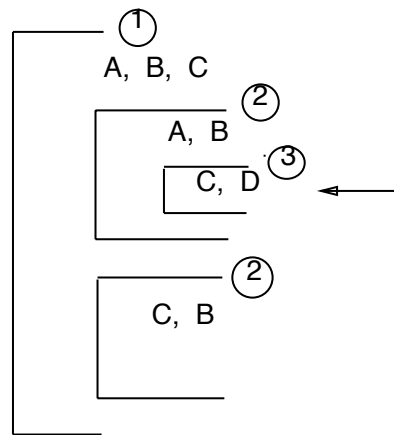


Figura 2: Exemplo1

4.2 Organização Linear

Nesta organização, a tabela é uma lista linear de registros e consiste na abordagem mais simples de implementação da tabela de símbolos. As listas se apresentam de duas formas, uma delas ordena os registros à medida em que eles são inseridos, e outra não leva a ordem em consideração. Ambas são descritas a seguir.

4.2.1 Lista Não-Ordenada

A forma mais simples de organizar uma tabela é adicionar nomes em uma lista linear de registros, à medida em que eles forem aparecendo, sem dar importância à ordem. A constituição de uma entrada para um nome e a procura pelo nome são operações independentes. Uma pesquisa requer a comparação com cada entrada da tabela até que o argumento seja encontrado. Esse algoritmo é simples e iterativo, mas eficiente apenas para tabelas pequenas. Para uma tabela com n entradas, é feita em média $n/2$ comparações. A Figura 3 ilustra como as variáveis do exemplo dado são representadas nesta abordagem. Os algoritmos para esta abordagem são apresentados a seguir.

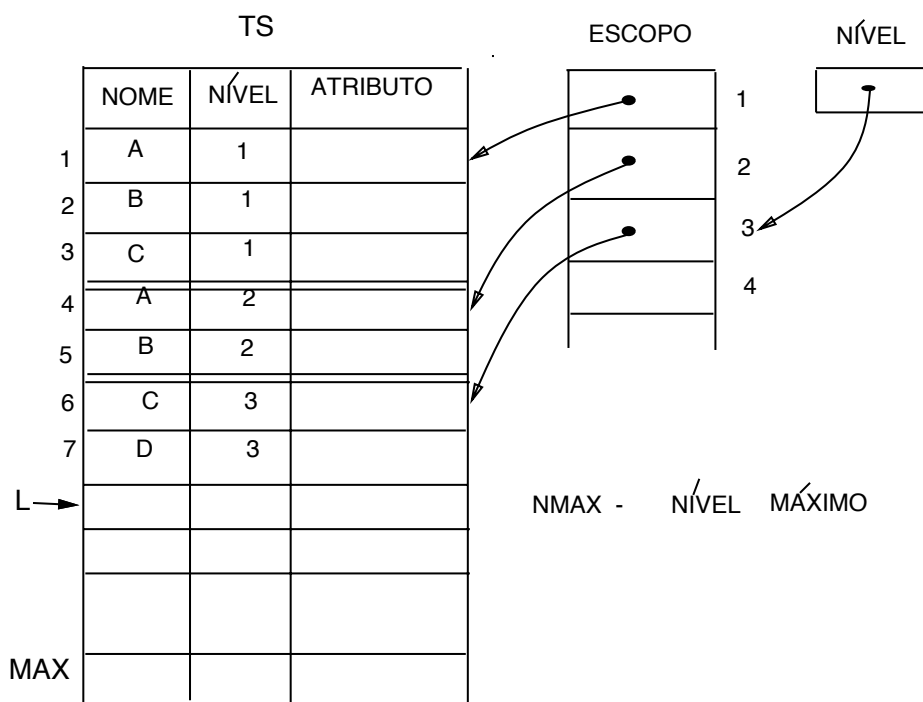


Figura 3: Representação na Abordagem Linear

1. Algoritmo para Entrar em um Bloco:

AbreBloco

NÍVEL := NÍVEL + 1

```

    if NÍVEL > NMAX then erro
    ESCOPO[NÍVEL] := L
end

```

2. Algoritmo para a Sair de um Bloco:

```

SaiBloco
    L := ESCOPO[NÍVEL]
    NÍVEL := NÍVEL - 1
end

```

3. Algoritmo para Pesquisar um Registro:

nesta operação, a pesquisa é feita de dentro para fora, do nível mais interno para o nível mais externo, e ao encontrar a primeira ocorrência do registro o seu nível, e os atributos são retornados.

```

Pesquisa(X)
    K := L (0 – não achou; K – endereço símbolo)
    while K > 1 do
        K := K - 1
        if X = TS.NOME[k]
        then return(K)
    end
    return(0)
end

```

4. Algoritmo para Inserir um Registro:

nesta operação, antes de tudo, verifica se o registro não foi anteriormente inserido dentro do mesmo escopo. Caso não tenha sido, o registro é inserido na posição que a variável L aponta.

```

Instala(X, Atributo)
    K := L
    while K > ESCOPO[NÍVEL] do
        K := K - 1
        if X = TS.NOME[k] then erro
    end
    if L = MAX + 1 then erro
    TS.NOME[L] := X
    TS.NÍVEL[L] := NÍVEL
    TS.ATRIBUTO[L] := ATRIBUTO
    L := L + 1
end

```

4.2.2 Lista Ordenada

Quando as entradas da tabela são efetuadas seguindo algum tipo de ordenação, a pesquisa é mais eficiente. Como os argumentos são *strings* de caracteres, a ordenação mais natural é a alfabética. Um método eficiente de pesquisa para a lista ordenada, de n entradas, é a pesquisa binária. O símbolo S para ser encontrado, é comparado com o argumento que se encontra na posição no meio da tabela $(n+1)/2$. Se a entrada não for encontrada, devemos compará-la ou com as entradas das posições 1 a $(n+1)/2-1$ ou com as posições $(n+1)/2+1$ a n , dependendo se S é menor ou maior que a entrada comparada. O número máximo de comparações efetuadas é $1+\log(\text{base } 2)n$. Os algoritmos para esta abordagem são:

1. Algoritmo para a Entrar em um Bloco:

```
AbreBloco
  NÍVEL := NÍVEL + 1
  if NÍVEL > NMAX then erro
  ESCOPO[NÍVEL] := L
end
```

2. Algoritmo para Sair de um Bloco:

```
SaiBloco
  L := ESCOPO[NÍVEL]
  NÍVEL := NÍVEL - 1
end
```

3. Algoritmo para Pesquisar um Registro:

nesta operação, a pesquisa é feita levando-se em conta a ordem, e é mais comumente chamada de pesquisa binária. Ao encontrar a primeira ocorrência do nome é retornado o nível e o atributo do nome.

```
Pesquisa(X)
  K := L (0 – não achou; K – endereço símbolo)
  Dir := K
  repeat
    i := (Esq+Dir)div 2
    if X > TS.NOME[i]
      then Esq:=i+1
      else Dir:=i-1
    until X=TS.NOME[i] or Esq > Dir
    if X = TS.NOME[i]
      then return(i)
      else return(0)
  end
```

4. Algoritmo para Inserir um Registro:

nesta operação, a Inserção de um registro é feita considerando a ordem alfabética dos nomes, ao contrário da lista não-ordenada.

```
Instala(X, Atributo)
  K := ESCOPO[NÍVEL]
  while K < L do
    i := K - 1
    while X < TS.NOME[i] do
      TS.NOME[i+1] := TS.NOME[i]
      TS.NÍVEL[i+1] := TS.NÍVEL[i]
      TS.ATRIBUTO[i+1] := TS.ATRIBUTO[i]
      i := i-1
    end
    TS.NOME[i+1] := NOME
    TS.NÍVEL[i+1] := NÍVEL
  end
end
```

4.3 Organização na Forma de Árvore Binária

Uma árvore binária é definida como um conjunto de nodos onde cada nodo tem exatamente 0 ou 2 filhos; quando um nodo tem 2 filhos, esses são chamados de filhos à esquerda e à direita do nodo. Uma árvore binária de pesquisa é uma árvore binária em que todo nodo interno contém um registro, e, para cada nodo todos os registros com chaves menores estão na subárvores esquerda e todos os registros com chaves maiores estão na subárvore direita. A Figura 5 apresenta a representação das variáveis do exemplo mostrado na Figura 4 nesta abordagem. Os algoritmos para as operações de Abrir e Fechar Bloco, Inserir, Remover e Pesquisar por um símbolo são:

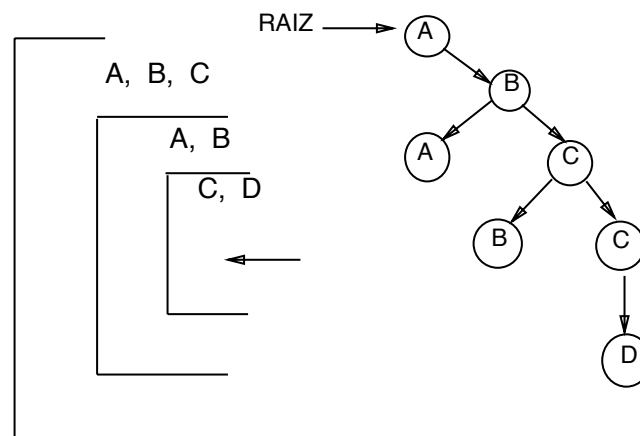


Figura 4: Exemplo2

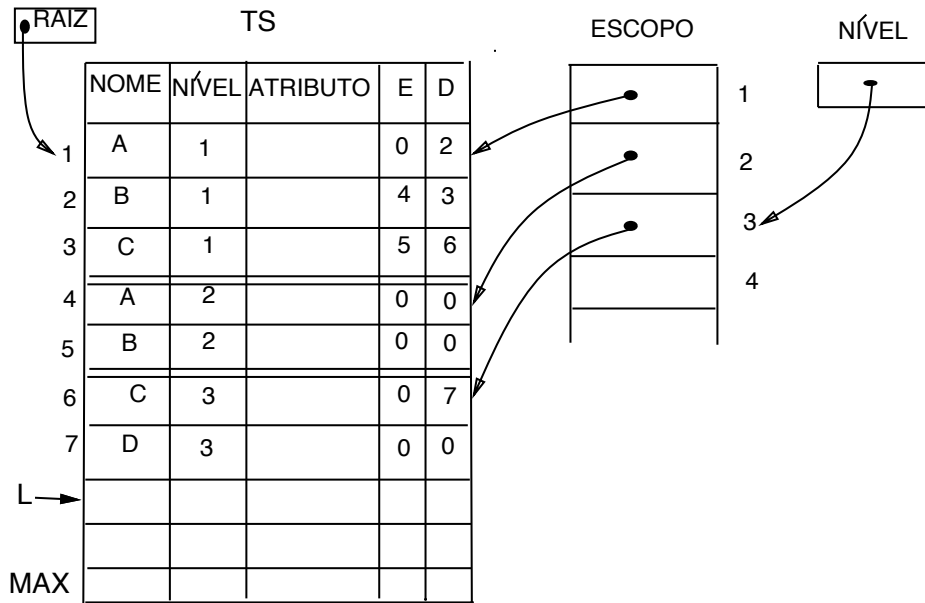


Figura 5: Representação na Abordagem Árvore Binária

1. Algoritmo para Entrar em um Bloco:

AbreBloco

NÍVEL := NÍVEL + 1

if NÍVEL > NMAX **then** *erro*

ESCOPO[NÍVEL] := L

end

2. Algoritmo para Sair de um Bloco:

nesta operação, ao sair de um bloco, os apontadores do mesmo são removidos. A variável L indica a posição inicial do bloco corrente.

SaiBloco

L := ESCOPO[NÍVEL] (libera folhas)

if RAIZ ≥ L **then** RAIZ := 0

else for I := 1 **to** L - 1

if TS.E[I] ≥ L **then** TS.E[I] := 0

if TS.D[I] ≥ L **then** TS.D[I] := 0

end

NÍVEL := NÍVEL - 1

end

3. Algoritmo para Pesquisar um Registro:

nesta operação, a pesquisa é feita percorrendo-se os filhos de um nodo à direita e à esquerda, que são representados respectivamente pelas variáveis D e E. O algoritmo pára quando tais variáveis são nulas. Ao encontrar a última ocorrência do nome é retornado o nível e os atributos do nome.

```
Pesquisa(X)
  S := RAIZ
  K := 0 (0 - não achou; K - endereço de X)
  while S ≠ 0 do
    if X = TS.NOME[S] then K := S; S := TS.D[S]
    else
      if X < TS.NOME[S] then S := TS.E[S]
      else S := TS.D[S]
    end
  end
  return(K)    end
```

4. Algoritmo para Inserir um Registro:

nesta abordagem, o algoritmo não pode parar ao encontrar a primeira ocorrência do registro. Deve-se achar todas as demais ocorrências do nome nos níveis mais internos, *loop while*. No final, é retornado a posição da última ocorrência da variável. Insere-se então o registro nessa posição, e atualiza-se os filhos E e D.

```
Instala(X, Atributo)
  S := RAIZ; K := 0; i := RAIZ
  while S ≠ 0 do
    i := S
    if X = TS.NOME[S] then K := S; S := TS.D[S] else
    if X < TS.NOME[S] then S := TS.E[S]
    else S := TS.D[S]
  end
  if K ≥ ESCOPO[NÍVEL] then erro
  if L ≥ MAX + 1 then erro
  TS.NOME[L] := X; TS.NÍVEL[L] := NÍVEL
  TS.ATRIBUTO[L] := ATRIBUTO; TS.E[L] := TS.D[L] := 0
  if RAIZ = 0 then RAIZ := L else
  if X < TS.NOME[i] then TS.E[i] := L
else TS.D[i] := L;
L := L + 1
end
```

4.4 Organização na Forma de Floresta de Árvores Binárias

Em uma floresta de árvores binárias, a cada entrada em um novo bloco, uma nova árvore binária é montada separadamente. Mesmo quando as árvores não são balanceadas, esse algoritmo é mais eficiente que os demais como vamos ver na Seção 6. No algoritmo de

pesquisa, por exemplo, considera-se que referências às variáveis locais são mais frequentes que as variáveis globais, favorecendo a pesquisa local. A pesquisa é feita do fim para o início, sem a necessidade de percorrer a árvore toda. Inicialmente, pesquisa-se a árvore mais interna, nível corrente, caso não encontre o nome procurado, então pesquisa-se nas outras árvores, níveis anteriores. Isso torna o algoritmo mais eficiente que os outros. A Figura 7 mostra como se representa na tabela de símbolos, usando essa abordagem, as variáveis do exemplo mostrado na Figura 6.

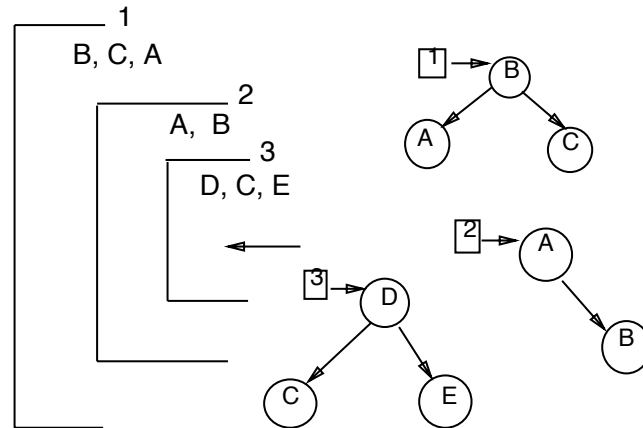


Figura 6: Exemplo3

1. Algoritmo para Entrar em um Bloco:

AbreBloco

NÍVEL := NÍVEL + 1

if NÍVEL > NMAX **then** *erro*

ESCOPO[NÍVEL] := 0

end

2. Algoritmo para Sair de um Bloco:

nesta operação, não é necessário atualizar os apontadores como no algoritmo anterior.

SaiBloco

if ESCOPO[NÍVEL] $\neq$ 0

then L := ESCOPO[NÍVEL]

NÍVEL := NÍVEL - 1 **end**

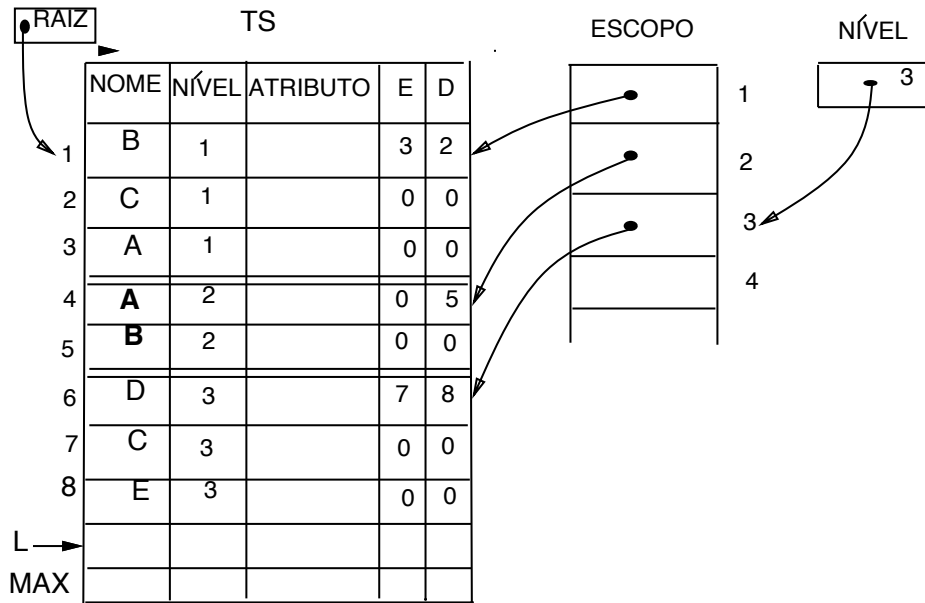


Figura 7: Representação na Abordagem Floresta de Árvores Binárias

3. Algoritmo para Pesquisar um Registro:

Pesquisa(X)

n := NÍVEL (0 - não achou; K - endereço de X na TS)

while n > 0 **do**

K := ESCOPO[n]

while K ≠ 0 **do**

if X = TS.NOME[K] **then** return(K) **else**

if X < TS.NOME[K] **then** K := TS.E[K] **else** K := TS.D[K]

end

n := n - 1

end; return(0) **end**

4. Algoritmo para Inserir um Registro :

Instala(X, Atributo)

S := ESCOPO[NÍVEL]

while S ≠ 0 **do**

i := S

if X = TS.NOME[S] **then** erro **else**

if X < TS.NOME[S] **then** S := TS.E[S] **else** S := TS.D[S]

```

end
if L = MAX + 1 then erro
TS.NOME[L] := X; TS.D[L] := 0; TS.E[L] := 0
if ESCOPO[NÍVEL] = 0 then ESCOPO[NÍVEL] := L else
if X < TS.NOME[i] then TS.E[i] := L else TS.D[i] := L
L := L + 1
end

```

4.5 Organização na Forma de HASH

A idéia central de uma tabela *hash* é mapear cada um dos nomes que entrarão na tabela para um número fixo de posições na tabela *hash*. O método de *hashing* é constituído de duas etapas principais:

- computar o valor da função de transformação função *hashing*, a qual transforma a chave de pesquisa em um endereço da tabela;
- criar um método para lidar com as colisões.

As colisões ocorrem porque o número de nomes que podem entrar na tabela de símbolos é geralmente, muito maior que o número de endereços na tabela *hash*. Assim, para nomes n_1 e n_2 ($n_1 \neq n_2$), $h(n_1) = h(n_2)$ temos uma colisão. Existem vários métodos para solucionar as colisões:

- Resolução Linear: seja n um inteiro correspondente à chave, e m o tamanho da tabela, se a posição $h(n)$ está ocupada, tente $(h(n)+1) \bmod m$, $(h(n)+2) \bmod m$...
- Solução pela Adição ao *Hash*: seja n um inteiro correspondente à chave, e m o tamanho da tabela, se $h(n)$ está ocupada, tente $(2 \cdot h(n)) \bmod m$, $(3 \cdot h(n)) \bmod m$...
- Solução Quadrática: seja n um inteiro correspondente à chave, e m o tamanho da tabela, se a posição $h(n)$ está ocupada, tente $(h(n)+1^{**2}) \bmod m$, $(h(n)+2^{**2}) \bmod m$...
- Listas Encadeadas: constrói-se uma lista encadeada para cada endereço da tabela. Assim, todas as chaves com mesmo endereço são encadeadas em uma lista linear. As vantagens dessa solução são a economia de espaço de armazenamento, ausência de grandes falhas, facilidade ao remover nomes da tabela, ao contrário das outras técnicas.

A Figura 8 ilustra a representação dos símbolos do Exemplo2, Seção 4.3 para esta abordagem.

1. Algoritmo para Entrar em um Bloco:

```

AbreBloco
  NÍVEL := NÍVEL + 1
  if NÍVEL > NMAX then erro
  ESCOPO[NÍVEL] := L
end

```

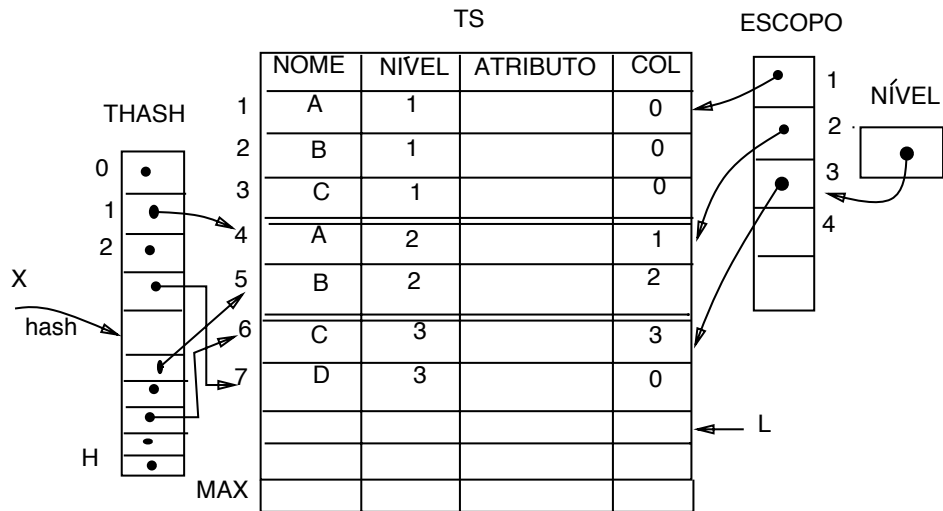


Figura 8: Representação na Abordagem *Hash* do Exemplo2

2. Algoritmo para Sair de um Bloco:

```

SaiBloco
  S := L
  B := ESCOPO[NÍVEL]
  while S > B do % desfaz hash
    S := S - 1
    K := hash(TS.NOME[S])
    THASH[K] := TS.COL[S]
  end
  NÍVEL := NÍVEL - 1
  L := B
end

```

3. Algoritmo para Pesquisar um Registro:

```

Pesquisa(X)
  n := hash(X) (0 - não achou; K - endereço de X)
  K := THASH[n]
  while K ≠ 0 do
    if X = TS.NOME[K] then return(K)
    K := TS.COL[K]
  end
  return(0); end

```

4. Algoritmo para Inserir um Registro:

```
Instala(X, Atributo)
  n := hash(X); K := THASH[n]
  while K ≥ ESCOPO[NÍVEL] do
    if X = TS.NOME[K] then erro
    K := TS.COL[K]
  end
  if L = MAX + 1 then erro
  TS.NOME[L] := X; TS.NÍVEL[L] := NÍVEL
  TS.ATRIBUTO[L] := ATRIBUTO; TS.COL[L] := THASH[n];
  THASH[n] := L ; L := L + 1
end
```

4.6 Conclusão

Esta seção apresentou a especificação dos algoritmos para Inserir, Remover, Pesquisar e Adicionar nomes à Tabela de Símbolos para as abordagens Linear, Árvore Binária, Floresta de Árvores Binárias e *Hash*. A próxima seção descreve as decisões de projeto tomadas em relação a implementação destes algoritmos e a implementação propriamente dita nas linguagens Delphi, C e Java.

5 Implementação das Organizações da Tabela de Símbolos para Linguagens com Estruturas de Blocos

5.1 Introdução

Para a implementação dos algoritmos descritos na Seção 4 foram adotadas as seguintes políticas:

- a tabela de símbolos implementada contém três campos: nome, nível e atributo. O campo atributo, na verdade representa os possíveis atributos que um símbolo necessita. Por exemplo, ele poderia ser substituído por: classe, tipo, offset, etc. O número de parâmetros das operações podem variar de acordo com a necessidade. Este fato não interfere com o algoritmo em si. Existe um vetor de apontadores que apontam para as primeiras posições de cada bloco e uma variável identificada por nível que simboliza o número do bloco tratado no momento;
- os algoritmos fazem a distinção entre identificadores com letras minúsculas e letras maiúsculas;
- os algoritmos consideram iguais as variáveis com o mesmo nome, em um mesmo escopo, tendo ou não os mesmos atributos;
- ao sair de um bloco, todas as variáveis locais são apagadas;
- uma variável não pode ser inserida mais de uma vez na tabela, no mesmo escopo;

- uma variável não pode ser inserida quando a tabela já está cheia;

5.2 Estruturas de Dados

As abordagens Linear, Árvore Binária, Floresta de Árvore Binária e *Hash* para as implementações nas linguagens Delphi, Java e C utilizam as seguintes estruturas de dados:

Tabela de Símbolos propriamente dita:

recebe o nome de *tabtokens* em todas as abordagens e consiste em um vetor de tokens.

Token:

consiste em um registro com diversos campos que são diferenciados conforme as abordagens.

Nome:

representa o nome do identificador e é um campo do *token* em todas as abordagens.

Nível:

simboliza o número do bloco tratado no momento e faz parte dos campos do *token* em todas as abordagens.

Atributo:

representa o atributo de um identificador, e está presente em *tokens* de todas as abordagens.

Esquerda:

corresponde ao índice da tabela que se refere ao filho do lado esquerdo do *token*. Está presente nas abordagens Árvore Binária e Floresta de Árvore Binária.

Direita:

corresponde ao índice da tabela que se refere ao filho do lado direito do *token*. Está presente nas abordagens Árvore Binária e Floresta de Árvore Binária.

Max:

constante global que especifica o tamanho máximo da tabela. Presente em todas as abordagens.

Escopo:

variável global que é um vetor com os índices dos primeiros *tokens* de cada nível. Presente em todas as abordagens.

L:

variável global que contém o índice da tabela referente a próxima posição da tabela a ser preenchida.

5.3 Implementação em Delphi

Os códigos fontes referentes às operações da tabela de símbolos implementados em Delphi encontram-se no Apêndice A.

5.3.1 Interfaces

Todos os algoritmos apresentam uma interface inicial, possuidora dos botões :

- Entra em um bloco: significa a entrada em um bloco, procedimentos ou funções, escrevendo o número do mesmo.
- Sai de um bloco: significa a saída de um bloco, procedimentos ou funções, no qual se entrou mais recentemente.
- Pesquisa item na tabela: procura o registro desejado após o campo indicado como item ser preenchido.
- Instala item na tabela: instala um item e seu respectivo atributo, após os campos indicados como item e atributo serem preenchidos.
- Visualizar arquivo com a tabela: permite ao usuário observar, durante toda a execução do programa, a tabela de símbolos.
- Fechar: sai do programa.
- Voltar: botão habilitado somente quando a estrutura da tabela de símbolos está sendo visualizada. Volta para a interface inicial.

O código fonte relativo á implementação em Delphi pode ser visto no Apêndice A.

5.3.2 Janelas de Interface

Interface Principal

Interface Secundária

Interface de Visualização da Tabela de Símbolos

5.4 Implementações em C e Java

Os códigos fontes referentes as operações da tabela de símbolos implementados nas linguagens C e Java encontram-se nos Apêndices B e C respectivamente.

5.4.1 Interfaces em C e Java

A apresentação de interfaces na linguagem Delphi teve como objetivo fazer com que o usuário tivesse um melhor contato com as operações realizadas pela Tabela de Símbolos. Isso não foi feito nas linguagens C e Java, já que tais operações estão implícitas em um compilador.

6 Análise dos Resultados

6.1 Testes

Foram implementados três testes para analisar cada um das cinco operações: Inserção, Pesquisa, Remoção de símbolos, Entrada e Saída de blocos nas quatro abordagens: Linear, Árvore Binária, Floresta de Árvore Binária e *Hash* nas linguagens trabalhadas. Na implementação dos testes foi levado em conta situações com várias instalações de identificadores, pesquisas de variáveis locais e globais, entradas e saídas de blocos. Para obtermos a medida do tempo, os testes foram efetuados com 100000 iterações nos procedimentos.

6.1.1 Teste 01

O Teste01 consiste em montar quatro blocos aninhados, dotados de duas variáveis cada um. A cada operação de Entrada de Bloco, as variáveis do primeiro nível são pesquisadas. O enfoque principal deste teste é na operação **pesquisa de variável global**. Espera-se um maior tempo médio de pesquisa em relação aos demais tempos das outras operações em todas as abordagens.

6.1.2 Teste 02

O Teste02 consiste em montar blocos paralelos e blocos aninhados. O enfoque principal deste teste é levantar o comportamento das abordagens levando em conta como a pesquisa é feita por identificadores, quando há duas variáveis: local e global com o mesmo nome. A cada operação de Entrada de Bloco, as variáveis globais são pesquisadas. Espera-se um relativo aumento do tempo médio de pesquisa em relação ao Teste01 porque neste caso temos blocos paralelos aos blocos aninhados.

6.1.3 Teste 03

O Teste03 consiste em montar três blocos aninhados dotados de vários identificadores cada um. O enfoque principal deste teste é a operação de **Instalação de símbolos**. A cada operação de Entrada de Bloco, as variáveis globais são pesquisadas antes e após as instalações das variáveis locais. Com o maior número de identificadores instalados por bloco, espera-se um grande aumento do tempo médio da operação Instala em relação aos Teste01 e Teste02. O tempo médio de pesquisa deve se manter semelhante aqueles obtidos nos testes apresentados nas Seções 6.1.1 e 6.1.2, respectivamente.

6.2 Gráficos

6.2.1 Gráficos Referentes ao Teste01 para a Linguagem Delphi

Podemos observar pelos gráficos, um melhor desempenho da abordagem Floresta de Árvores Binárias em relação as demais abordagens considerando o tempo total de execução deste teste. A abordagem usando *Hash* apresenta um bom desempenho, seguido pela abordagem Árvore Binária.

Como esperado, na abordagem Linear, especificamente o algoritmo Lista Não Ordenada apresentou o pior tempo, ultrapassando 5000 segundos(aproximadamente 83 minutos) no final da execução deste teste. Isso explica a padronização do tempo em todos os gráficos, permitindo uma melhor comparação.

Considerando os tempos médios das operações, a operação de Pesquisa obteve o maior tempo médio nas abordagens: Linear, Árvore Binária e *Hash*. O tempo de pesquisa na abordagem Floresta de Árvores Binárias é inferior ao tempo de pesquisa nas outras abordagens considerando a implementação nesta linguagem.

Apesar de um grande tempo na pesquisa, *Hash* obteve bons tempos nas Inserções e Entrada de bloco, assegurando um bom desempenho global no Teste01.

6.2.2 Gráficos Referentes ao Teste01 para a Linguagem C

Na linguagem C, o desempenho da abordagem Floresta de Árvores Binárias continua sendo superior. Podemos observar uma melhoria no desempenho do algoritmo *Hash*, aproximando-se da abordagem Floresta de Árvores Binárias. No entanto, a operação Saída de Bloco faz com que a tabela *Hash* seja refeita, ocasionando um tempo maior em relação a esta abordagem.

As abordagens Linear e Árvore Binária apresentam desempenhos semelhantes. Isso ocorre porque o tempo de Pesquisa da Árvore Binária aproxima-se do tempo da abordagem Linear, já que no caso da Árvore Binária todos os nodos à direita e à esquerda devem ser percorridos. Além disso, a operação Saída de Bloco apresenta um maior tempo em relação ao tempo obtido na abordagem Linear.

O tempo nos gráficos é padronizado em 8 segundos, adiantando a observação do melhor desempenho da linguagem C em relação a linguagem Delphi.

Em relação aos tempos médios, a abordagem Linear obteve um maior tempo para a operação pesquisa. As outras abordagens apresentaram maior tempo na operação de Inserção.

6.2.3 Gráficos Referentes ao Teste01 para a Linguagem Java

Na linguagem Java, o desempenho da abordagem Floresta de Árvores Binárias é pior do que nas linguagens C e Delphi, apesar de continuar obtendo o melhor resultado entre os algoritmos avaliados. Podemos observar uma maior semelhança no desempenho obtido nas abordagens *Hash* e Floresta de Árvore Binárias, e um pior desempenho na abordagem Árvore Binária em relação as linguagens C e Delphi.

O tempo nos gráficos é padronizado em 100 segundos, adiantando a observação do segundo melhor desempenho entre as linguagens por Java.

6.2.4 Gráficos Referentes ao Teste01 com o Desempenho entre as Linguagens Delphi, C e Java

Muito embora não seja preciso o resultado obtido da comparação mostrada nesta seção para os cinco algoritmos, considerando as três linguagens, Delphi, C e Java, devido ao fato de cada uma dessas linguagens pertencerem a diferentes ambientes de programação, tentou-se nestes gráficos padronizar o tempo em 1250 segundos, para a melhor comparação dos mesmos, em relação ao Teste01, em cada uma dessas linguagens.

Nos cinco algoritmos apresentados, a linguagem C obteve o melhor tempo na execução do Teste01. O pior resultado foi o da ferramenta Delphi. Como Delphi utiliza o compilador Pascal e é dotada de muitos recursos, como componentes visuais, suporte a banco de dados, esse é um resultado esperado. O fato de a Linguagem Java ser interpretada, isto confere a ela o segundo lugar na execução global do Teste01.

6.2.5 Gráficos Referentes ao Teste02 para a Linguagem Delphi

Podemos observar pelos gráficos, um melhor desempenho da abordagem Floresta de Árvores Binárias em relação aos demais algoritmos.

A abordagem Árvore Binária apresenta-se melhor que o *Hash*, que se aproxima bastante da abordagem Linear, especificamente da Lista Ordenada.

Em relação ao tempo médio, todas as abordagens apresentaram um tempo de execução pior em relação ao Teste01. Isso ocorreu devido ao aumento do tempo médio de pesquisa e, em alguns casos, do tempo médio de Inserção. Como o Teste02 abre um maior número de blocos, o tempo médio de pesquisa e Inserção é aumentado.

6.2.6 Gráficos Referentes ao Teste02 para a Linguagem C

Como na ferramenta Delphi, as abordagens apresentaram um aumento no tempo de execução em relação Teste01. No entanto, esse aumento não foi tão expressivo como em Delphi.

O desempenho da abordagem Floresta de Árvores Binárias continua sendo superior, seguido pelo algoritmo *Hash*.

As abordagens Linear e Árvore Binária apresentaram desempenhos semelhantes. Isso ocorreu porque o tempo de Pesquisa da Árvore Binária aproximou-se do tempo da abordagem Linear, já que na abordagem Árvore Binária, todos os nodos à direita e à esquerda devem ser percorridos. Além disso, a operação Saída de Bloco apresentou um maior tempo em relação ao tempo obtido na abordagem Linear.

Em relação ao tempo médio, a operação de Inserção na Árvore Binária foi relativamente menor do que na abordagem Linear.

6.2.7 Gráficos Referentes ao Teste02 para a Linguagem Java

Como podemos observar nestes gráficos, o desempenho da abordagem Floresta de Árvores Binárias continua obteve o melhor resultado entre os algoritmos avaliados.

Podemos observar uma aproximação do tempo de execução obtido no algoritmo *Hash* da abordagem Floresta de Árvores Binárias, e um pior desempenho da abordagem Árvore Binária em relação às linguagens C e Delphi.

As abordagens Linear e *Hash* obtiveram desempenhos semelhantes.

6.2.8 Gráficos Referentes ao Teste02 com o Desempenho entre as Linguagens Delphi, C e Java

Muito embora não seja preciso o resultado obtido da comparação mostrada nesta seção para os cinco algoritmos, considerando as três linguagens, Delphi, C e Java, devido ao fato de cada uma dessas linguagens pertencerem a diferentes ambientes de programação, tentou-se nestes gráficos padronizar o tempo em 1250 segundos, para a melhor comparação dos mesmos, em relação ao Teste02, em cada uma dessas linguagens.

Como no Teste01, nos cinco algoritmos apresentados, a linguagem C obteve o melhor tempo na execução global deste teste. O pior resultado foi o da ferramenta Delphi, seguido por Java.

Na abordagem Árvore Binária, o desempenho da linguagem Java piorou, em relação aos demais algoritmos.

6.2.9 Gráficos Referentes ao Teste03 para a Linguagem Delphi

Todas as abordagens apresentaram um pior tempo de execução em relação aos testes apresentados nas Seções 6.1.1 e 6.1.2, respectivamente.

A abordagem Linear, especificamente a Lista Não Ordenada apresentou o segundo melhor desempenho, já que a operação Inserção foi a mais simples de todos os algoritmos, e é avaliada neste teste.

A Árvore Binária apresentou um desempenho ruim, devido á piora no tempo da operação Entrada de Bloco, explicada pelo maior número de variáveis em cada bloco.

Podemos observar que a abordagem Floresta de Árvores Binárias apresentou um melhor desempenho, seguida pela abordagem Linear, especificamente, Lista Não Ordenada.

Em relação aos tempos médios, a abordagem *Hash* obteve um aumento nos tempos médios de Pesquisa, Inserção e Entrada de Bloco. Como o Teste03 insere um maior número de variáveis, é natural que o tempo médio de Inserção em todas as abordagens seja acrescido.

6.2.10 Gráficos Referentes ao Teste03 para a Linguagem C

Podemos observar que a Floresta de Árvores Binárias, pela primeira vez, não apresentou o melhor desempenho em relação aos demais algoritmos. *Hash* obteve o melhor desempenho, seguido por Floresta de Árvores Binárias. Isso é explicado pelo maior número de variáveis locais, aumentando o tempo médio de Inserção e em alguns casos o tempo de pesquisa.

As abordagens Árvore Binária e Linear apresentaram um desempenho semelhante.

Houve um aumento no tempo médio de pesquisa nos algoritmos lineares.

6.2.11 Gráficos Referentes ao Teste03 para a Linguagem Java

Como podemos observar, os desempenhos de todas as abordagens foram piores em relação aos testes para esta mesma linguagem apresentados nas Seções 6.1.1 e 6.1.2, respectivamente.

Houve um aumento nos tempos médios de Inserção nos algoritmos Linear, Árvore Binária e Floresta de Árvores Binárias.

Floresta de Árvores Binárias e *Hash* apresentaram os melhores desempenhos, ainda que não tão bons.

As abordagens Linear e Árvore Binária obtiveram desempenhos semelhantes.

Os tempos médios de pesquisa e Entrada de Bloco aumentaram na abordagem Linear.

6.2.12 Gráficos Referentes ao Teste03 com o Desempenho entre as Linguagens Delphi, C e Java

Muito embora não seja preciso o resultado obtido da comparação mostrada nesta seção para os cinco algoritmos, considerando as três linguagens, Delphi, C e Java, devido ao fato de cada uma dessas linguagens pertencerem a diferentes ambientes de programação, tentou-se nestes gráficos padronizar o tempo em 1250 segundos, para a melhor comparação dos mesmos, em relação ao Teste03, em cada uma dessas linguagens.

Como nos testes apresentados nas Seções 6.1.1, 6.1.2, nos cinco algoritmos apresentados, a linguagem C obteve o melhor tempo na execução deste teste. O pior resultado foi o da ferramenta Delphi, seguido por Java. Em todas as abordagens, o desempenho da linguagem Java piorou, em relação aos demais algoritmos.

6.3 Avaliação Final

Em Delphi, as operações de Instalação, Pesquisa e Entrada no Bloco, efetuadas com a abordagem Linear, especificamente na Lista Não Ordenada, obtiveram a ordem de 290 segundos nos três testes: Teste01, Teste02 e Teste03.

A operação Saída do Bloco apresentou menor tempo, aproximadamente 1 segundo.

Já abordagem Linear, especificamente a Lista Ordenada demonstrou-se melhor que a Lista Não Ordenada, apresentando para as operações Pesquisa e Entrada de Bloco um tempo relativamente menor. Isso aconteceu devido a ordenação interna da tabela de símbolos, tornando a Pesquisa mais eficiente.

A organização *Hash* apresentou tempos de pesquisa parecidos com os da abordagem Linear.

A operação Instalação na Árvore Binária obteve o tempo reduzido em relação as abordagens Linear e *Hash*. Isso ocorreu devido à organização interna dos nodos da tabela de símbolos.

A abordagem Floresta de Árvores Binárias obteve o melhor resultado dentre as demais organizações. Os tempos de Instalação e Pesquisa foram muito inferiores a todos os demais algoritmos. Por outro lado, a operação Entrada de Bloco apresentou tempo pior em relação as abordagens Linear, especificamente a Lista Ordenada, Árvore Binária e *Hash*. Este fato é explicado porque a cada operação Entrada de Bloco, uma nova árvore é construída.

Na linguagem C, os resultados obtidos conferem com aqueles observados na linguagem Delphi. As operações Instalação, Pesquisa e Entrada no Bloco, para a abordagem Linear, especificamente a Lista Não Ordenada, obtiveram a ordem de dez centésimos de segundos nos três testes. O método Saída do Bloco apresentou menor tempo, aproximadamente 0,001 segundos. O algoritmo Lista Ordenada se demonstrou melhor que o anterior, apresentando para as operações Pesquisa e Entrada de Bloco com tempo relativamente menores. Isso acontece devido à ordenação interna da tabela de símbolos, tornando a Pesquisa mais eficiente.

A organização *Hash* apresentou tempos menores na operação Pesquisa em relação a abordagem Linear.

Para a abordagem Árvore Binária, a operação Pesquisa obteve o tempo reduzido em relação a abordagem Linear e *Hash*. Devido a organização interna da tabela de símbolos com nodos, a Instalação de identificadores apresenta maior eficiência, diminuindo também, razoavelmente o tempo na operação Pesquisa.

A abordagem Floresta de Árvores Binárias obteve o melhor resultado dentre as demais organizações. Os tempos de Instalação e Pesquisa foram inferiores a um segundo, apresentando excelente desempenho quando comparado as outras abordagens.

Em Java, as operações Instalação e Pesquisa, efetuadas na abordagem Linear, obtiveram tempos entre dois e cinco segundos nos três testes.

Na abordagem Linear, especificamente Lista Não Ordenada, as operações Entrada e Saída do Bloco apresentaram menores tempos, na ordem de décimos de segundos.

O algoritmo Lista Ordenada se mostrou superior a Lista Não Ordenada, apresentando as operações Pesquisa e Instalação com tempo relativamente menores. Isso também aconteceu devido à ordenação interna da tabela de símbolos, tornando a Pesquisa mais eficiente.

A organização *Hash* apresentou tempos semelhantes para as operações Pesquisa e Instalação em relação ao algoritmo Lista Ordenada.

Na abordagem Árvore Binária, as operações Pesquisa e Instalação obtiveram tempos semelhantes em relação aos algoritmos Lista Ordenada e *Hash*.

O algoritmo Floresta de Árvores Binárias obteve o melhor resultado entre os algoritmos. Os tempos de Pesquisa foram inferiores a um segundo, apresentando excelente desempenho quando comparado com as outras abordagens.

7 Conclusão

Concluimos que o algoritmo Floresta de Árvores Binárias apresenta o melhor desempenho entre todos os outros. Considerando que a pesquisa é feita do fim para o início, sem a necessidade de percorrer a árvore toda, e que as referências às variáveis locais são mais frequentes que as variáveis globais, favorecendo a pesquisa local, este algoritmo é mais eficiente que os outros.

A abordagem *Hash* apresenta o segundo melhor desempenho na maioria dos casos, sendo sempre seguido pela Árvore Binária.

A linguagem que apresenta os melhores tempos é a Linguagem C. Isso se explica pelo fato da linguagem Delphi possuir muitos recursos que influenciam no tempo de compilação e Java por ser uma linguagem interpretada, o que a torna mais lenta no quesito execução.

8 Bibliografia

Referências

- [1] A. V. Aho, J. D. Ullman, and R. S. Sethi. *Compilers, Principles, Techniques and Tools*. Addison Wesley Publishing Company Co., 1986.
- [2] Gosling J. Arnold K. *Programando em JavaTM*. Makron Books do Brasil Editora Ltda, 1997.
- [3] Roberto S. Bigonha and Mariza A. S. Bigonha. *Organização de Tabelas de Símbolos*. Notas de aula, Departamento de Ciência da Computação, UFMG, 1994.
- [4] C.N. Fischer and LeBlanc Jr. *Crafting a Compiler with C*. The Benjamin/Cummings Publishing Company, Inc., 1991.
- [5] D. Gries. *Compiler Construction for Digital Computers*. Wiley, 1972.
- [6] Schildt H. *C Completo e Total*. Makron Books do Brasil Editora Ltda, 1991.
- [7] Aleen I. Holub. *Compiler Design in C*. Prentice Hall, ICN, 1990.
- [8] Gosling J., Joy B., and Steele. *The Java Language Specification*. Addison-Wesley, 1996.

A Fontes dos Programas na Linguagem Delphi

A.1 Abordagem Linear na Linguagem Delphi

A.1.1 Lista Não Ordenada

```
unit ListNOrd;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs,
  StdCtrls;

type
  TFrmListNOrd = class(TForm)
    LNEntra: TButton;
    LNSai: TButton;
    LNPesquisa: TButton;
    LNLabItemInst: TLabel;
    LNItemPesqu: TEdit;
    LNInstala: TButton;
    LNLabItemPesqu: TLabel;
    LNItemInst: TEdit;
    LNItemInstAtrib: TEdit;
    LNLabItemAtrib: TLabel;
    LNFecha: TButton;
    LNVolta: TButton;
    LNVisualiza: TButton;
    LNTabela: TMemo;
    LNAviso2: TLabel;
    LNAviso1: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure LNEntraClick(Sender: TObject);
    procedure LNSaiClick(Sender: TObject);
    procedure LNPesquisaClick(Sender: TObject);
    procedure LNInstalaClick(Sender: TObject);
    procedure TiraEspaco(var palavra: string);
    procedure LNFechaClick(Sender: TObject);
    procedure LNVisualizaClick(Sender: TObject);
    procedure LNVoltaClick(Sender: TObject);
    procedure LNItemPesquClick(Sender: TObject);
    procedure LNItemInstClick(Sender: TObject);
    procedure LNItemInstAtribClick(Sender: TObject);
  private
    { Private declarations }
```

```

public
  { Public declarations }
end;

const max=10000; {Define o tamanho maximo da tabela de smmbolos}
type
  token=record {Registro que forma a tabela de smmbolos}
    nome:string[10];{Contim o nome do token}
    nivel:integer;{Contim o nmvel em que o token se
    atributo:string[15];{Contim o atributo do token}
  end;

var
  FrmListNOrd: TFrmListNOrd;
  tabtokens:array [1..max] of token;{Representa a tabela de smmbolos}
  escopo:array [1..max]of integer;{Vetor que contem os mndices dos  primeiros}
  nivel: integer;{Contim o nzmero do nmvel que o token esta}
  raiz:integer;{Contim o mndice da tabela que contim a raiz}
  L:integer;{Contim o mndice da tabela referente a prsxima posigco da
  tabela a ser preenchida}
  entroubloco:boolean;{Diz se entrou em um bloco ou nco}
implementation
{$R *.DFM}

uses Principal;
procedure TFrmListNOrd.FormCreate(Sender: TObject);
begin
  nivel:=1;{Nivel 1 i considerado o das variaveis globais}
  L:=1;
  raiz:=0;
  entroubloco:=false;
  escopo[nivel]:=1;
end;

procedure TFrmListNOrd.LNEntraClick(Sender: TObject);
begin
  inc(nivel);
  LNAviso1.caption:='';
  entroubloco:=true;
  if (nivel > max) then  LNAviso1.caption:=' ERRO!!! A tabela ja
  completa}
  else escopo[nivel]:=L;
  LNAviso1.caption:='Entrada no bloco ' +inttostr(nivel);
end;

procedure TFrmListNOrd.LNSaiClick(Sender: TObject);

```

```

begin
    L:=escopo[nivel];
    dec(nivel);
    LNAviso1.caption:='';
    LNAviso2.caption:='';
end;

procedure TFrmListNOrd.TiraEspaco(var palavra: string);
var caracter:string;
    i,tamanho:integer;
begin
    i:=1;
    caracter:=copy(palavra,i,1);
    while caracter=' ' do
    begin
        inc(i);
        caracter:=copy(palavra,i,1);
    end;
    tamanho:=length(palavra)-i+1;
    palavra:=copy(palavra,i,tamanho);
end;

procedure TFrmListNOrd.LNPesquisaClick(Sender: TObject);
var S,K,tamanho:integer;
    nome:string;
    achou:boolean;
begin
    LNAviso1.caption:='';
    LNAviso2.caption:='';
    K:=L;
    achou:=false;
    nome:=LNItemPesqu.text;
    TiraEspaco(nome);
    tamanho:=length(nome);
    if (LNItemPesqu.text<>'') then
    begin
        while ((K > 1) and (not achou)) do
        begin
            dec(K);
            if (nome=copy(tabtokens[K].nome,1,tamanho))
            then achou:=true;
        end;
        if (achou) then LNAviso2.caption:= ' 0 item esta no nivel '+
        inttostr(tabtokens[K].nivel) + ', no mndice '+inttostr(K)+' e possui o
        atributo '+ tabtokens[K].atributo+'.'
        else LNAviso2.caption:= ' 0 item pesquisado nao foi encontrado!';
    end
end

```

```

        else LNAviso2.caption:= '0 campo item deve ser preenchido.';
        LNItemPesqu.text:='';
    end;

procedure TFrmListNOrd.LNInstalaClick(Sender: TObject);
var K,tamanho:integer;
    nome,atrib:string;
    igual:boolean;
begin
    LNAviso1.caption:='';
    igual:=false;
    LNAviso2.caption:='';
    nome:= LNItemInst.text;
    TiraEspaco(nome);
    tamanho:=length(nome);
    atrib:=LNItemInstAtrib.text;
    TiraEspaco(atrib);
    K:=L;
    if (nome<>'')and(atrib<>'')then
    begin
        while K > escopo[nivel] do
        begin
            dec(k);
            if nome=copy(tabtokens[K].nome,1,tamanho)
            then begin
                LNAviso2.caption:='ERRO! Este item ja foi inserido';
                igual:=true;
            end;
        end;
        if (L=max+1) then LNAviso2.caption:='A tabela ja se encontra
        cheia'
        else if (not igual)
        then begin
            tabtokens[L].nome:=nome;
            tabtokens[L].nivel:=nivel;
            tabtokens[L].atributo:=atrib;
            inc(L);
        end;
    end
    else LNAviso2.caption:='0s campos nome e atributo de vem ser
    preenchidos.';
    LNItemInst.text:='';
    LNItemInstAtrib.text:='';
end;

procedure TFrmListNOrd.LNFechaClick(Sender: TObject);
begin

```

```

        FrmListNOrd.visible:=false;
        FrmPrincipal.visible:=true;
end;

procedure TFrmListNOrd.LNVisualizaClick(Sender: TObject);
var f:textfile;
    i,arquivo:integer;
begin
    LNVolta.enabled:=true;
    assignfile(f,'tabela.txt');
    rewrite(f);
    i:=1;
    while i <= L-1 do
    begin
        write(f,'Nome : ',tabtokens[i].nome,' ');
        write(f,'Atributo : ',tabtokens[i].atributo,' ');
        writeln(f,'Nivel : ',tabtokens[i].nivel,' ');
        inc(i);
    end;
    closefile(f);
    LNTabela.lines.loadfromfile('tabela.txt');
    LNTabela.visible:=true;
end;

procedure TFrmListNOrd.LNVoltaClick(Sender: TObject);
begin
    LNTabela.visible:=false;
    LNVolta.enabled:=false;
end;

procedure TFrmListNOrd.LNItemPesquClick(Sender: TObject);
begin
    LNAviso2.caption:='';
    LNAviso1.caption:='';
end;

procedure TFrmListNOrd.LNItemInstClick(Sender: TObject);
begin
    LNAviso2.caption:='';
    LNAviso1.caption:='';
end;

procedure TFrmListNOrd.LNItemInstAtribClick(Sender: TObject);
begin
    LNAviso2.caption:='';
    LNAviso1.caption:='';
end;

```

```
end;  
end.
```

A.1.2 Lista Ordenada

```
unit ListOrd;  
interface  
uses  
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,  
    Dialogs,  
    StdCtrls;  
type  
    TFrmListOrd = class(TForm)  
        LEntrar: TButton;  
        LSai: TButton;  
        LPesquisa: TButton;  
        LInstala: TButton;  
        LFecha: TButton;  
        LVolta: TButton;  
        LVisualiza: TButton;  
        LItemPesqu: TEdit;  
        LItemInst: TEdit;  
        LItemInstAtrib: TEdit;  
        LAviso1: TLabel;  
        LAviso2: TLabel;  
        LLabItemPesqu: TLabel;  
        LLabItemInst: TLabel;  
        LLabItemAtrib: TLabel;  
        LTabela: TMemo;  
        procedure FormCreate(Sender: TObject);  
        procedure LEntrarClick(Sender: TObject);  
        procedure LSaiClick(Sender: TObject);  
        procedure TiraEspaco(var palavra: string);  
        procedure LFechaClick(Sender: TObject);  
        procedure LVisualizaClick(Sender: TObject);  
        procedure LVoltaClick(Sender: TObject);  
        procedure LInstalaClick(Sender: TObject);  
        procedure LPesquisaClick(Sender: TObject);  
        procedure LItemPesquClick(Sender: TObject);  
        procedure LItemInstClick(Sender: TObject);  
        procedure LItemInstAtribClick(Sender: TObject);  
  
    private  
        { Private declarations }  
  
    public  
        { Public declarations }  
    end;  
end;
```

```

end;

const max=10000; {Define o tamanho maximo da tabela de smmbolos}
type
  token=record {Registro que forma a tabela de smmbolos}
    nome:string[10];{Contim o nome do token}

    nivel:integer;{Contim o nmvel em que o token se
    encontra}
    atributo:string[15];{Contim o atributo do token}
  end;
var
  FrmListOrd: TFrmListOrd;
  tabtokens:array [1..max] of token;{Representa a tabela de smmbolos}
  escopo:array [1..max]of integer;{Vetor que contim os mndices dos
  primeiros tokens de cada nmvel}
  nivel: integer;{Contim o nzmero do nmvel que o token esta}
  raiz:integer;{Contim o mndice da tabela que contim a raiz}
  L:integer;{Contim o mndice da tabela referente a prxima posigco da
  tabela a ser preenchida}
  entroubloco:boolean;{Diz se entrou em um bloco ou nco}

implementation
{$R *.DFM}
uses Principal;

procedure TFrmListOrd.FormCreate(Sender: TObject);
begin
  nivel:=1;{Nivel 1 i considerado o das variaveis globais}
  L:=1;
  raiz:=0;
  entroubloco:=false;
  escopo[nivel]:=1;
end;

procedure TFrmListOrd.LEntraClick(Sender: TObject);
begin
  entroubloco:=true;
  if (nivel > max) then  LAviso1.caption:=' ERRO!!! A tabela ja
  completa.'
  else escopo[nivel]:=L;
  LAviso1.caption:='Entrada no bloco ' +inttostr(nivel);
end;

procedure TFrmListOrd.LSaiClick(Sender: TObject);
begin
  L:=escopo[nivel];

```



```

        dec(nivel);
    end;

procedure TFrmListOrd.TiraEspaco(var palavra: string);
var caracter:string;
    i,tamanho:integer;
begin
    i:=1;
    caracter:=copy(palavra,i,1);
    while caracter=' ' do
    begin
        inc(i);
        caracter:=copy(palavra,i,1);
    end;
    tamanho:=length(palavra)-i+1;
    palavra:=copy(palavra,i,tamanho);
end;

procedure TFrmListOrd.LFechaClick(Sender: TObject);
begin
    FrmListOrd.visible:=false;
    FrmPrincipal.visible:=true;
end;

procedure TFrmListOrd.LVisualizaClick(Sender: TObject);
var f:textfile;
    i,arquivo:integer;
begin
    Lvolta.enabled:=true;
    assignfile(f,'tabela.txt');
    rewrite(f);
    i:=1;
    while i <= L-1 do
    begin
        write(f,'Nome : ',tabtokens[i].nome,' ');
        write(f,'Atributo : ',tabtokens[i].atributo,' ');
        writeln(f,'Nivel : ',tabtokens[i].nivel,' ');
        inc(i);
    end;
    closefile(f);
    LTabela.lines.loadfromfile('tabela.txt');
    LTabela.visible:=true;
end;

procedure TFrmListOrd.LVoltaClick(Sender: TObject);
begin

```

```

    LTabela.visible:=false;
    LVolta.enabled:=false;
end;

procedure TFrmListOrd.LInstalaClick(Sender: TObject);
var i,K,tamanho,j:integer;
    nome,atrib:string;
    inseriu,DeuErro:boolean;
begin
    LAviso1.caption:='';
    LAviso2.caption:='';
    nome:= LItemInst.text;
    TiraEspaco(nome);
    tamanho:=length(nome);
    atrib:=LItemInstAtrib.text;
    TiraEspaco(atrib);
    K:=escopo[nivel];
    inseriu:=false;
    DeuErro:=false;
    i:=L-1;
    if (nome<>'')and(atrib<>'')then
    begin
        while ((not inseriu)and (not DeuErro)) do
        begin
            if L=escopo[nivel] then
            begin
                inseriu:=true;
                tabtokens[i+1].nome:= nome;
                tabtokens[i+1].atributo:= atrib;
                tabtokens[i+1].nivel:= nivel;
            end
            else begin
                while K < L do begin
                    if
                        nome=copy(tabtokens[K].nome,1,tamanho)
                    then begin
                        dec(L);
                        DeuErro:=true;
                        LAviso2.caption:='ERRO! Este item
                        ja foi inserido.'
                    end;
                    inc(K);
                end;
                if not DeuErro then begin
                    while ((ord(nome[1])<
                        ord(tabtokens[i].nome[1]))and (i>=escopo[nivel])) do
                        begin

```

```

        tabtokens[i+1].nome:=
        tabtokens[i].nome;
        tabtokens[i+1].atributo:=
        tabtokens[i].atributo;
        tabtokens[i+1].nivel:=
        tabtokens[i].nivel;
        dec(i);
    end;
    if L=max+1 then begin
        dec(L);
        LAviso2.caption:='A tabela ja se encontra
        cheia'
    end
    else begin
        inc(i);
        inseriu:=true;
        tabtokens[i].nome:= nome;
        tabtokens[i].atributo:= atrib;
        tabtokens[i].nivel:= nivel;
    end;
end;
    end;
    inc(L);
end;
end
    else LAviso2.caption:='Os campos nome e atributo de vem ser
    preenchidos.';
    LItemInst.text:='';
    LItemInstAtrib.text:='';
end;

procedure TFrmListOrd.LPesquisaClick(Sender: TObject);
var Esq,Dir,K,tamanho,i :integer;
    nome:string;
    achou:boolean;
begin
    LAviso1.caption:='';
    LAviso2.caption:='';
    Esq:=1;
    Dir:=L;
    achou:=false;
    nome:=LItemPesqu.text;
    TiraEspaco(nome);
    tamanho:=length(nome);
    if (nome<>'') then
    begin
        while ((Esq < Dir) and (not achou)) do{acho que nao eh

```

```

        Esq<=Dir, porque na primeira vez nao temn sentido pesquisar nada}
begin
    K:=(Esq+Dir)div 2;
    if nome=copy(tabtokens[K].nome,1,tamanho) then
        achou:=true
    else if (ord(nome[1])> ord(tabtokens[K].nome[1]))
        then Esq:=K+1
        else Dir:=K-1;
    end;
    if (achou) then LAviso2.caption:= ' 0 item esta no nivel
'+ inttostr(tabtokens[K].nivel) + ', no mndice '+inttostr(K)+' e possui
o atributo '+ tabtokens[K].atributo+'.'
        else LAviso2.caption:= ' 0 item pesquisado nao
        foi encontrado!';
    end
    else LAviso2.caption:= '0 campo item deve ser preenchido.';
    LItemPesqu.text:='';
end;

procedure TFrmListOrd.LItemPesquClick(Sender: TObject);
begin
    LAviso1.caption:='';
    LAviso2.caption:='';
end;

procedure TFrmListOrd.LItemInstClick(Sender: TObject);
begin
    LAviso1.caption:='';
    LAviso2.caption:='';
end;

procedure TFrmListOrd.LItemInstAtribClick(Sender: TObject);
begin
    LAviso1.caption:='';
    LAviso2.caption:='';
end;

end.

```

A.2 Abordagem Árvore Binária na Linguagem Delphi

```

unit Arvore;
interface
uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
    Dialogs,

```

```

StdCtrls;
type
  TFrmArvBin = class(TForm)
    AEntra: TButton;
    ASai: TButton;
    APesquisa: TButton;
    AInstala: TButton;
    AItemPesqu: TEdit;
    ALabItemPes: TLabel;
    ALabItemInst: TLabel;
    AItemInst: TEdit;
    AItemInstAtrib: TEdit;
    ALabItemAtrib: TLabel;
    Label5: TLabel;
    AAviso2: TLabel;
    Label7: TLabel;
    AFechar: TButton;
    AAviso1: TLabel;
    AVisualiza: TButton;
    ATabela: TMemory;
    AVolta: TButton;
    procedure FormCreate(Sender: TObject);
    procedure AEntraClick(Sender: TObject);
    procedure ASaiClick(Sender: TObject);
    procedure APesquisaClick(Sender: TObject);
    procedure AInstalaClick(Sender: TObject);
    procedure AFecharClick(Sender: TObject);
    procedure TiraEspaco(var palavra: string);
    procedure AVisualizaClick(Sender: TObject);
    procedure AVoltaClick(Sender: TObject);
    procedure AItemPesquClick(Sender: TObject);
    procedure AItemInstClick(Sender: TObject);
    procedure AItemInstAtribClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

const max=10000; {Define o tamanho maximo da tabela de smmbolos}
type
  token=record {Registro que forma a tabela de smmbolos}
    nome:string[10];{Contim o nome do token}
    nivel:integer;{Contim o nmvel em que o token se encontra}
    atributo:string[15];{Contim o atributo do token}
    esquerda:integer;{Contim o mndice da tabela que se

```

```

        refere ao filho do lado esquerdo do token}
        direita:integer;{Contim o mndice da tabela que se
        refere ao filho do lado direito do token}
    end;

var
    FrmArvBin: TFrmArvBin;
    tabtokens:array [1..max] of token;{Representa a tabela de smbolos}
    escopo:array [1..max]of integer;{Vetor que contim os mndices dos
    primeiros tokens de cada nmvel}
    nivel: integer;{Contim o nzmero do nmvel que o token esta}
    raiz:integer;{Contim o mndice da tabela que contim a raiz}
    L:integer;{Contim o mndice da tabela referente a prxima posigco da
    tabela a ser preenchida}
    entroubloco:boolean;{Diz se entrou em um bloco ou nco}

implementation
uses Principal;
{$R *.DFM}
{Inicializa as variaveis globais principais}
procedure TFrmArvBin.FormCreate(Sender: TObject);
begin
    nivel:=1;{Nivel 1 i considerado o das variaveis globais}
    L:=1;
    raiz:=0;
    entroubloco:=false;
    escopo[nivel]:=1;
end;

{Representa o procedimento entra no bloco}
procedure TFrmArvBin.AEntraClick(Sender: TObject);
begin
    inc(nivel);
    label5.Caption:='';
    entroubloco:=true;
    if (nivel > max) then label5.caption:=' ERRO!!! A tabela ja
    completa.'
    else escopo[nivel]:=L;
    AAviso1.caption:='Entrada no bloco ' +inttostr(nivel);
end;

{Representa o procedimento sai do bloco}
procedure TFrmArvBin.ASaiClick(Sender: TObject);
var i,aux,j: integer;
begin
    L:=escopo[nivel];
    dec(nivel);

```

```

        if (raiz>=L) then raiz:=0
        else for i:=1 to L-1 do
        begin
            if (tabtokens[i].esquerda>=L)then tabtokens[i].esquerda:=0;
            if(tabtokens[i].direita>=L)then tabtokens[i].direita:=0;
        end;
        entroubloco:=false;
        AAviso1.caption:=' Samda do bloco' + inttostr(nivel);
    end;

procedure TFrmArvBin.TiraEspaco(var palavra: string);
var caracter:string;
    i,tamanho:integer;
begin
    i:=1;
    caracter:=copy(palavra,i,1);
    while caracter=' ' do
    begin
        inc(i);
        caracter:=copy(palavra,i,1);
    end;
    tamanho:=length(palavra)-i+1;
    palavra:=copy(palavra,i,tamanho);
end;

{Representa o procedimento pesquisa item na tabela}
procedure TFrmArvBin.APesquisaClick(Sender: TObject);
var S,K,tamanho,i:integer;
    nome:string;
    achou : boolean;
begin
    achou := false;
    AAviso1.caption:='';
    AAviso2.caption:='';
    if (AItemPesqu.text<>'') then
    begin
        nome:=AItemPesqu.text;
        TiraEspaco(nome);
        tamanho:=length(nome);
        S:=raiz;
        K:=0;
        while(S<>0)do
        begin
            if not achou then
            if(nome=copy(tabtokens[S].nome,1,tamanho))then
            begin
                achou := true;
            end;
        end;
    end;
end;

```

```

        K:=S;
        S:=tabtokens[S].direita;
    end
    else if
        (ord(nome[1])<ord(tabtokens[S].nome[1]))then
            S:=tabtokens[S].esquerda
        else S:=tabtokens[S].direita
    else S:=0;
    end;
    if (k<>0) then AAViso2.caption:= ' 0 item pesquisado
    esta no nivel '+inttostr(tabtokens[K].nivel)+ ', no indice
    '+inttostr(K)+' , e possui o atributo '+tabtokens[K].atributo+ ' .'
    else AAViso2.caption:= ' 0 item pesquisado nao foi
    encontrado!';
    end
    else AAViso2.caption:= '0 campo item deve ser preenchido.';
    AItemPesqu.text:=' ';
end;

{Representa o procedimento insere item na tabela}
procedure TFrmArvBin.AInstalaClick(Sender: TObject);
var i,S,K,tamanho,j:integer;
    nome,atributo:string;
    igual:boolean;
    compara:char;
begin
    AAViso1.caption:='';
    AAViso2.caption:='';
    nome:= AItemInst.text;
    TiraEspaco(nome);
    tamanho:=length(nome);
    atributo:=AItemInstAtrib.text;
    TiraEspaco(atributo);
    S:=raiz;
    K:=0;
    i:=raiz;
    igual:=false;
    if (nome<>'')and(atributo<>'')then
    begin
        while (S<>0)do
        begin
            i:=S;
            if (nome=copy(tabtokens[S].nome,1,tamanho)) then
            begin
                K:=S;
                S:=tabtokens[S].direita;
                if (i>=escopo[nivel]) then

```



```

        AAViso2.caption:='ERRO! Este item ja foi inserido!';
    end
    else begin
        compara:=tabtokens[S].nome[1];
        if (ord(nome[1])< ord(compara)) then
            S:=tabtokens[S].esquerda
        else S:=tabtokens[S].direita;
        end;
    end;
    if (L>=max+1) then AAViso2.caption:=' ERRO! A tabela ja
        esta cheia';
    if (K < escopo[nivel])then
    begin
        tabtokens[L].nome:=nome;
        tabtokens[L].nivel:=nivel;
        tabtokens[L].atributo:=atributo;
        tabtokens[L].esquerda:=0;
        tabtokens[L].direita:=0;
        if (raiz=0) then raiz:=L
        else if (ord(nome[1])<ord(tabtokens[i].nome[1]))
            then tabtokens[i].esquerda:=L
            else tabtokens[i].direita:=L;
        inc(L);
    end;
    end
    else AAViso2.caption:='Os campos nome e atributo de vem ser
        preenchidos.';
    AItemInst.text:=' ';
    AItemInstAtrib.text:=' ';
end;

{Fecha a aplicagco}
procedure TFrmArvBin.AFechaClick(Sender: TObject);
begin
    FrmArvBin.visible:=false;
    FrmPrincipal.visible:=true;
    {FrmArvBin.close;}
end;

procedure TFrmArvBin.AVisualizaClick(Sender: TObject);
var f:textfile;
    i,arquivo:integer;
begin
    AVolta.enabled:=true;
    assignfile(f,'tabela.txt');
    rewrite(f);
    i:=1;

```

```

while i <= L-1 do
begin
    write(f,'Nome : ',tabtokens[i].nome,' ');
    write(f,'Atributo : ',tabtokens[i].atributo,' ');
    write(f,'Nivel : ',tabtokens[i].nivel,' ');
    write(f,'Esquerda : ',tabtokens[i].esquerda,' ');
    writeln(f,'Direita : ',tabtokens[i].direita,' ');
    inc(i);
end;
closefile(f);
ATabela.lines.loadfromfile('tabela.txt');
ATabela.visible:=true;
end;

procedure TFrmArvBin.AVoltaClick(Sender: TObject);
begin
    ATabela.visible:=false;
    AVolta.enabled:=false;
end;

procedure TFrmArvBin.AItemPesquClick(Sender: TObject);
begin
    AAviso1.caption:='';
    AAviso2.caption:='';
end;

procedure TFrmArvBin.AItemInstClick(Sender: TObject);
begin
    AAviso1.caption:='';
    AAviso2.caption:='';
end;

procedure TFrmArvBin.AItemInstAtribClick(Sender: TObject);
begin
    AAviso1.caption:='';
    AAviso2.caption:='';
end;
end.

```

A.3 Abordagem Floresta de Árvore Binária na Linguagem Delphi

```

unit Floresta;
interface
uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
    Dialogs,

```

```

StdCtrls;
type
  TFrmFloArvBin = class(TForm)
    FBEntra: TButton;
    FBSai: TButton;
    FBPesquisa: TButton;
    FBInstala: TButton;
    FBFechar: TButton;
    FBVolta: TButton;
    FBVisualiza: TButton;
    FBItemPesqu: TEdit;
    FBItemInst: TEdit;
    FBItemInstAtrib: TEdit;
    FBLabItemPesqu: TLabel;
    FBLabItemInst: TLabel;
    FBLabItemAtrib: TLabel;
    FBAviso1: TLabel;
    FBAviso2: TLabel;
    FBTabella: TMemo;
    FBAviso3: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FBEntraClick(Sender: TObject);
    procedure FBSaiClick(Sender: TObject);
    procedure ABTiraEspaco(var ABpalavra: string);
    procedure FBPesquisaClick(Sender: TObject);
    procedure FBInstalaClick(Sender: TObject);
    procedure FBFecharClick(Sender: TObject);
    procedure FBVisualizaClick(Sender: TObject);
    procedure FBVoltaClick(Sender: TObject);
    procedure FBItemPesquClick(Sender: TObject);
    procedure FBItemInstClick(Sender: TObject);
    procedure FBItemInstAtribClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

const ABmax=10000; {Define o tamanho maximo da tabela de smmbolos}
type
  ABtoken=record {Registro que forma a tabela de smmbolos}
    nome:string[10];{Contim o nome do token}
    nivel:integer;{Contim o nmvel em que o token se
      encontra}
    atributo:string[15];{Contim o atributo do token}
    esquerda:integer;{Contim o mndice da tabela que se
      refere ao filho do lado esquerdo do token}
  end;

```

```

        direita:integer;{Contim o mndice da tabela que se
                           refere ao filho do lado direito do token}
    end;

var
    FrmFloArvBin: TFrmFloArvBin;
    ABtabtokens:array [1..ABmax] of ABtoken;{Representa a tabela de
        simbolos}
    ABescopo:array [1..ABmax]of integer;{Vetor que contim os mndices dos
        primeiros tokens de cada nmvel}
    ABnivel: integer;{Contim o nzmero do nmvel que o token esta}
    ABraiz:integer;{Contim o mndice da tabela que contim a raiz}
    ABL:integer;{Contim o mndice da tabela referente a prxima posigco da
        tabela a ser preenchida}
    ABentroubloco:boolean;{Diz se entrou em um bloco ou nco}

implementation
{$R *.DFM}
uses Principal;
procedure TFrmFloArvBin.FormCreate(Sender: TObject);
begin
    ABnivel:=1;{Nivel 1 i considerado o das variaveis globais}
    ABL:=1;
    ABraiz:=0;
    ABentroubloco:=false;
    ABescopo[ABnivel]:=0;
end;

procedure TFrmFloArvBin.FBEntraClick(Sender: TObject);
begin
    inc(ABnivel);
    FBaviso1.caption :='';
    ABentroubloco:=true;
    if (ABnivel > ABmax) then FBaviso1.caption:=' ERRO!!! A
        tabela ja completa.'
    else ABescopo[ABnivel]:=0;
    FBaviso1.caption:='Entrada no bloco ' +inttostr(ABnivel);
end;

procedure TFrmFloArvBin.FBSaiClick(Sender: TObject);
var i,j: integer;
begin
    if ABescopo[ABnivel]<> 0
    then ABL:=ABescopo[ABnivel];
    ABentroubloco:=false;
    FBaviso1.caption:=' Samda do bloco' + inttostr(ABnivel);
    dec(ABNivel);

```

```

end;

procedure TFrmFloArvBin.ABTiraEspaco(var ABpalavra: string);
var caracter:string;
    i,tamanho:integer;
begin
    i:=1;
    caracter:=copy(ABpalavra,i,1);
    while caracter=' ' do
    begin
        inc(i);
        caracter:=copy(ABpalavra,i,1);
    end;
    tamanho:=length(ABpalavra)-i+1;
    ABpalavra:=copy(ABpalavra,i,tamanho);
end;

procedure TFrmFloArvBin.FBPesquisaClick(Sender: TObject);
var n,K,tamanho,i:integer;
    nome:string;
    achou:boolean;
begin
    achou:=false;
    FBAviso1.caption:='';
    FBAviso2.caption:='';
    if (FBItemPesqu.text<>'') then
    begin
        nome:=FBItemPesqu.text;
        ABTiraEspaco(nome);
        tamanho:=length(nome);
        n:=ABnivel;
        while((n > 0)and(not achou))do
        begin
            K:=ABEscopo[n];
            while ((K<>0)and(not achou)) do
            begin
                if(nome=copy(ABtabtokens[K].nome,1,tamanho))then achou:=true
                else if
                (ord(nome[1])<ord(ABtabtokens[K].nome[1]))then
                    K:=ABtabtokens[K].esquerda
                else K:=ABtabtokens[K].direita;
            end;
            dec(n);
        end;
        if (achou) then FBAviso2.caption:= ' O item pesquisado
esta no nivel '+inttostr(ABtabtokens[K].nivel)+ ', no indice
'+inttostr(K)+', e possui o atributo '+ABtabtokens[K].atributo+ ' .'

```

```

        else FBAviso2.caption:= ' 0 item pesquisado nao foi
encontrado!';
    end
    else FBAviso2.caption:= '0 campo item deve ser preenchido.';
    FBItemPesqu.text:=' ';
end;

procedure TFrmFloArvBin.FBInstalaClick(Sender: TObject);
var i,S,K,tamanho,j:integer;
    nome,atributo:string;
    igual:boolean;
    compara:char;
begin
    FBAviso1.caption:='';
    FBAviso2.caption:='';
    igual:=false;
    nome:= FBItemInst.text;
    ABTiraEspaco(nome);
    tamanho:=length(nome);
    atributo:=FBItemInstAtrib.text;
    ABTiraEspaco(atributo);
    S:=ABEscopo[ABnivel];
    if (nome<>'')and(atributo<>'')then
    begin
        while (S<>0)do
        begin
            i:=S;
            if (nome=copy(ABtabtokens[S].nome,1,tamanho)) then
            begin
                igual:=true;
                S:=0;
                FBAviso2.caption:='ERRO! Este item ja foi
                    inserido'
            end
            else begin
                compara:=ABtabtokens[S].nome[1];
                if (ord(nome[1])< ord(compara)) then
                    S:=ABtabtokens[S].esquerda

                else S:=ABtabtokens[S].direita;
            end;
        end;
        if (ABL=ABmax+1) then FBAviso2.caption:=' ERRO! A tabela
            ja esta cheia';
        if (not igual)then
        begin
            ABtabtokens[ABL].nome:=nome;

```

```

        ABtabtokens[ABL].nivel:=ABnivel;
        ABtabtokens[ABL].atributo:=atributo;
        ABtabtokens[ABL].esquerda:=0;
        ABtabtokens[ABL].direita:=0;
        if (ABEscopo[ABnivel]=0) then ABEscopo[ABnivel]:=ABL
        else if
            (ord(nome[1])<ord(ABtabtokens[i].nome[1])) then
            ABtabtokens[i].esquerda:=ABL
            else ABtabtokens[i].direita:=ABL;
            inc(ABL);
        end;
    end
    else FBaviso2.caption:='Os campos nome e atributo de vem ser
        preenchidos.';
    FBItemInst.text:=' ';
    FBItemInstAtrib.text:=' ';
end;

procedure TFrmFloArvBin.FBFecharClick(Sender: TObject);
begin
    FrmFloArvBin.visible:=false;
    FrmPrincipal.visible:=true;
end;

procedure TFrmFloArvBin.FBVisualizaClick(Sender: TObject);
var f:textfile;
    i,arquivo:integer;
begin
    FBVolta.enabled:=true;
    assignfile(f,'tabela.txt');
    rewrite(f);
    i:=1;
    while i <= ABL-1 do
    begin
        write(f,'Nome : ',ABtabtokens[i].nome,' ');
        write(f,'Atributo : ',ABtabtokens[i].atributo,' ');
        write(f,'Nivel : ',ABtabtokens[i].nivel,' ');
        write(f,'Esquerda : ',ABtabtokens[i].esquerda,' ');
        writeln(f,'Direita : ',ABtabtokens[i].direita,' ');
        inc(i);
    end;
    closefile(f);
    FBtabela.lines.loadfromfile('tabela.txt');
    FBtabela.visible:=true;
end;

procedure TFrmFloArvBin.FBVoltaClick(Sender: TObject);

```

```

begin
    FBTabela.visible:=false;
    FBVolta.enabled:=false;
end;

procedure TFrmFloArvBin.FBItemPesquClick(Sender: TObject);
begin
    FBAviso1.caption:='';
    FBAviso2.caption:='';
end;

procedure TFrmFloArvBin.FBItemInstClick(Sender: TObject);
begin
    FBAviso1.caption:='';
    FBAviso2.caption:='';
end;

procedure TFrmFloArvBin.FBItemInstAtribClick(Sender: TObject);
begin
    FBAviso1.caption:='';
    FBAviso2.caption:='';
end;
end.

```

A.4 Abordagem *Hash* na Linguagem Delphi

```

unit Hash;
interface
uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
    Dialogs,
    StdCtrls;
type
    TFrmHash = class(TForm)
        HEntra: TButton;
        HSai: TButton;
        HPesquisa: TButton;
        HItemPesqu: TEdit;
        HLabItemPesq: TLabel;
        HInstala: TButton;
        HLabItemInst: TLabel;
        HItemInst: TEdit;
        HLabAtrib: TLabel;
        HItemAtrib: TEdit;
        HFecha: TButton;
        HVolta: TButton;
        HVisualiza: TButton;
    end;

```



```

HAviso1: TLabel;
HAviso2: TLabel;
HTabela: TMemo;
HAviso3: TLabel;
procedure HEntraClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure HSaiClick(Sender: TObject);
procedure TiraEspaco(var palavra: string);
procedure HPesquisaClick(Sender: TObject);
procedure HInstalaClick(Sender: TObject);
procedure HFechaClick(Sender: TObject);
procedure HVisualizaClick(Sender: TObject);
procedure HVoltaClick(Sender: TObject);
procedure HItemInstClick(Sender: TObject);
procedure HItemAtribClick(Sender: TObject);
procedure HItemPesquClick(Sender: TObject);

private
  { Private declarations }
public
  { Public declarations }
end;
const max=10000; {Define o tamanho maximo da tabela de smmbolos}
const mhash = 509;
type
  token=record {Registro que forma a tabela de smmbolos}
    nome:string[10];{Contim o nome do token}
    nivel:integer;{Contim o nmvel em que o token se
                  encontra}
    atributo:string[15];{Contim o atributo do token}
    col: integer;
  end;
  Indice = 0..mhash-1 ; {Indice da tabela Hash}

var
  FrmHash: TFrmHash;
  TabTokens:array [1..max] of token;{Representa a tabela de smmbolos}
  TabHash: array [1..mhash] of integer; {Representa a tabela hash}
  escopo:array [1..max]of integer;{Vetor que contim os mndices dos
    primeiros tokens de cada nmvel}
  nivel: integer;{Contim o nzmero do nmvel que o token esta}
  raiz:integer;{Contim o mndice da tabela que contim a raiz}
  L:integer;{Contim o mndice da tabela referente a prsxima posigco da
    tabela a ser preenchida}
  entroubloco:boolean;{Diz se entrou em um bloco ou nco}

implementation

```

```

uses Principal;
{$R *.DFM}

Function Hashing(Chave: string): Indice;
var i, Soma, tamanho: Integer;
begin
    tamanho:=length(Chave);
    Soma:=0;
    For i:=1 to tamanho do
        Soma:=Soma + ord(Chave[i]);
    Hashing:=Soma mod mhash;
end;

procedure TFrmHash.TiraEspaco(var palavra: string);
var caracter:string;
    i,tamanho:integer;
begin
    i:=1;
    caracter:=copy(palavra,i,1);
    while caracter=' ' do
        begin
            inc(i);
            caracter:=copy(palavra,i,1);
        end;
    tamanho:=length(palavra)-i+1;
    palavra:=copy(palavra,i,tamanho);
end;

procedure TFrmHash.HEntraClick(Sender: TObject);
begin
    inc(nivel);
    entroubloco:=true;
    if (nivel > max) then HAviso1.caption:=' ERRO!!! A tabela ja
        completa.'
    else escopo[nivel]:=L;
    HAviso1.caption:='Entrada no bloco ' +inttostr(nivel);
end;

procedure TFrmHash.FormCreate(Sender: TObject);
begin
    nivel:=1;{Nivel 1 i considerado o das variaveis globais}
    L:=1;
    raiz:=0;
    entroubloco:=false;
    escopo[nivel]:=1;
end;

```

```

procedure TFrmHash.HSaiClick(Sender: TObject);
var S, B,i:integer;
    K:Indice;
begin
    S:=L;
    B:=escopo[nivel];
    While S > B do
    begin
        dec(S);
        K:=Hashing(tabtokens[S].nome);
        TabHash[K]:=Tabtokens[S].col;
    end;
    dec(nivel);
    L:=B;
end;

procedure TFrmHash.HPesquisaClick(Sender: TObject);
var S,K,tamanho,i:integer;
    nome:string;
    achou:boolean;
    n:Indice;
begin
    HAviso1.caption:='';
    HAviso2.caption:='';
    achou:=false;
    nome:=HItemPesqu.text;
    TiraEspaco(nome);
    tamanho:=length(nome);
    n:=hashing(nome);
    K:=TabHash[n];
    if (HItemPesqu.text<>'') then
    begin
        while ((K <> 0) and (not achou)) do
        begin
            if (nome=copy(tabtokens[K].nome,1,tamanho))
            then achou:=true;
            if (achou) then HAviso2.caption:= ' 0 item esta no
nível '+ inttostr(tabtokens[K].nivel) + ', no mndice '+inttostr(K)+' e
possui o atributo '+ tabtokens[K].atributo+'.'
            else begin
                HAviso2.caption:= ' 0 item pesquisado nao
foi encontrado!';
                k:=tabtokens[K].col;    {Verificar essa
sentença...}
            end;
        end;
    end;
end
end

```

```

                else HAViso2.caption:= '0 campo item deve ser
                    preenchido.';
            HItemPesqu.text:=' ';
end;

procedure TFrmHash.HInstalaClick(Sender: TObject);
var K,tamanho,i:integer;
    nome,atrib:string;
    igual:boolean;
    n:Indice;
begin
    HAViso1.caption:='';
    HAViso2.caption:='';
    igual:=false;
    nome:= HItemInst.text;
    TiraEspaco(nome);
    tamanho:=length(nome);
    atrib:=HItemAtrib.text;
    TiraEspaco(atrib);
    n:= Hashing(nome);
    K:=TabHash[n];
    if (nome<>'')and(atrib<>'')then
    begin
        while K >= escopo[nivel] do
        begin
            if nome=copy(tabtokens[K].nome,1,tamanho)
            then begin
                HAViso2.caption:='ERRO! Este item ja foi
inserido';

                igual:=true;
            end;
            k:=tabtokens[k].col;
        end;
        if (L=max+1) then HAViso2.caption:='A tabela ja se
encontra cheia'
        else if (not igual)
        then begin
            tabtokens[L].nome:=nome;
            tabtokens[L].nivel:=nivel;
            tabtokens[L].atributo:=atrib;
            tabtokens[L].col:=tabhash[n];
            tabhash[n]:=L;
            inc(L);
        end;
    end
    else HAViso2.caption:='0s campos nome e atributo de vem ser
preenchidos.';

```

```

        HItemInst.text:='';
        HItemAtrib.text:='';
end;

procedure TFrmHash.HFecharClick(Sender: TObject);
begin
    FrmHash.visible:=false;
    FrmPrincipal.visible:=true;
end;

procedure TFrmHash.HVisualizaClick(Sender: TObject);
var f:textfile;
    i,arquivo:integer;
begin
    HVolta.enabled:=true;
    assignfile(f,'tabela.txt');
    rewrite(f);
    i:=1;
    while i <= L-1 do
    begin
        write(f,'Nome : ',tabtokens[i].nome,' ');
        write(f,'Atributo : ',tabtokens[i].atributo,' ');
        write(f,'Nivel : ',tabtokens[i].nivel,' ');
        writeln(f,'Col : ',tabtokens[i].col,' ');
        inc(i);
    end;
    closefile(f);
    HTabela.lines.loadfromfile('tabela.txt');
    HTabela.visible:=true;
end;

procedure TFrmHash.HVoltaClick(Sender: TObject);
begin
    HTabela.visible:=false;
    HVolta.enabled:=false;
end;

procedure TFrmHash.HItemInstClick(Sender: TObject);
begin
    HAviso1.caption:=' ';
end;

procedure TFrmHash.HItemAtribClick(Sender: TObject);
begin
    HAviso1.caption:=' ';
end;

```

```
procedure TFrmHash.HItemPesquClick(Sender: TObject);  
begin  
    HAviso1.caption:=' ';  
end;  
  
end.
```

B Fontes dos Programas na Linguagem C

B.1 Abordagem Linear na Linguagem C

B.1.1 Lista Não Ordenada

```
#include <string.h>
#include <conio.h>
#include <stdio.h>
#include <process.h>
#include <dos.h>
#include <time.h>
#define NMax 1000      /* Tamanho maximo da Tabela */
int escopo[10];
int nivel;      /* inteiro que contem o numero do nivel atual */
int L;          /* inteiro que contem o indice do ultimo elemento da
                Tabela Simbolos */
int Raiz;       /* inteiro que contem o indice do primeiro elemento da
                Tabela Simbolos */

struct {
    char nome[10];      /* Contem o nome do Simbolo */
    int nivel;          /* Contem o nivel do Simbolo relacionado */
    char atributo[10];  /* Contem o atributo do relacionado */
} TabelaS[100]; /* Vetor de struct que contem a
                tabela de simbolos */

void Entrada_Bloco(void);
void Erro(int numero);
void Saida_Bloco(void);
void Get_Entry(char name[10]);
void Instala(char name[10], char atributo[10]);
void imprimir(void);

void main(void)
{
    L = 0;          /* Considera-se que a primeira posicao da tabela
                    eh a de indice 0. L eh o final da arvore binaria */

    Raiz = -1;
    nivel = 1;      /* 0 primeiro nivel , o 1 */
    clrscr();
    escopo[nivel] = 0; /* escopo[1] contem o indice do primeiro elemento
                        */
}

/*****
Implementacoes
```

```
*****/
```

```
/* Funcao que define os erros provaveis de ocorrer  
*****/
```

```
void Erro(int num)  
{  
    char opcao;  
    switch (num) {  
printf("Tabela de Simbolos cheia\n"); break;  
printf("Elemento nao encontrado\n"); break;  
printf("Este item ja foi inserido\n"); break;  
    }  
}
```

```
/* Funcao de entrada num bloco  
*****/
```

```
void Entrada_Bloco()  
{  
    nivel++;  
    if (nivel > NMax)  
        Erro(1);  
    else escopo[nivel] = L;  
}
```

```
/* Funcao de saida de um bloco  
*****/
```

```
void Saida_Bloco()  
{  
    L= escopo[nivel];  
    nivel--;  
}
```

```
/* Funcao que pesquisa item na tabela  
*****/
```

```
void Get_Entry(char x[10]) /* Pesquisa o simbolo "x" e retorna o  
                           indice da Tabela de simbolos  
                           onde ele se encontra */  
{  
    int S, K, achou, aux;  
    K = L;
```



```

    achou = 0;
    while ((K > 0)&&(achou == 0))
    {
        K--;
        aux = strcmp(TabelaS[K].nome,x);
        if (aux == 0) achou = 1;
    }
    if (achou == 1)
    {
        printf("O item esta no nivel  %d", TabelaS[K].nivel);
        printf("                Indice %u",K);
    }
    else Erro(2);
}

/***** Funcao que instala o item na tabela *****/

void Instala(char X[10], char atribut[10]) /* Instala o simbolo "X" com
                                           atributo na Tabela de Simbolos */

{
    int igual, k, aux;
    k = L;
    igual = 0;
    while (k > escopo[nivel])
    {
        k--;
        aux = strcmp(TabelaS[k].nome,X);
        if (aux == 0)
        { Erro(3);
          igual = 1;
        }
    }
    if (L == NMax + 1)    Erro(1);
    else if (igual == 0)
    {
        TabelaS[L].nivel = nivel;
        aux = strlen(atribut);
        for (k = 0; k<= aux-1; k++)
            TabelaS[L].atributo[k] = atribut[k];
        aux = strlen(X);
        for (k = 0; k<=(aux-1); k++)
            TabelaS[L].nome[k] = X[k];
        L++;
    }
}

```

```

}

void imprimir()
{
    clrscr();
    int i;
    for (i=0; i<=L-1; i++)
    {
        printf("\n\nNome : ");
        printf("%s", TabelaS[i].nome);
        printf("\n");
        printf("Atributo : ");
        printf("%s", TabelaS[i].atributo);
        printf("\n");
        printf("Nivel : ");
        printf("%i", TabelaS[i].nivel);
        printf("\n");
        printf("\n"); }
}

```

B.1.2 Lista Ordenada

```

#include <string.h>
#include <conio.h>
#include <stdio.h>
#include <process.h>
#include <dos.h>
#define NMax 1000 /* Numero maximo de niveis possiveis */
int escopo[10];
int nivel; /* inteiro que contem o numero do nivel atual */
int L; /* inteiro que contem o indice do ultimo elemento da Tabela
        de Simbolos */
int Raiz; /* inteiro que contem o indice do primeiro elemento da
        Tabela de Simbolos */

struct {
    char nome[10]; /* Contem o nome do Simbolo */
    int nivel; /* Contem o nivel do Simbolo relacionado */
    char atributo[10]; /* Contem o atributo do relacionado */
} TabelaS[100]; /* Vetor de struct que contem a
        tabela de simbolos */

void Entrada_Bloco(void);
void Erro(int numero);
void Saida_Bloco(void);
void Get_Entry(char name[10]);
void Instala(char name[10], char atributo[10]);

```

```

void imprimir(void);

void main(void)
{
    L = 1;      /* Considera-se que a primeira posicao da tabela eh a de
                  indice 0. L eh o final da arvore binaria */

    Raiz = 0;
    nivel = 1;      /* 0 primeiro nivel , o 1 */
    escopo[nivel] = 1; /* escopo[1] contem o indice do primeiro elemento
                        */
}

/*****
                                Implementacoes
*****/

/***** Funcao que define os erros provaveis de ocorrer
*****/

void Erro(int num)
{
    char opcao;
    switch (num) {
        case 1:
            printf("Tabela de Simbolos cheia\n"); break;
        case 2:
            printf("Este item nao foi encontrado\n"); break;
        case 3:
            printf("Este item ja foi inserido\n"); break;
    }
}

/***** Funcao de entrada num bloco
*****/
void Entrada_Bloco()
{
    nivel++;
    if (nivel > NMax)
        Erro(1);
    else escopo[nivel] = L;
}

/***** Funcao de saida de um bloco
*****/
void Saida_Bloco()

```

```

{
    L= escopo[nivel];
    nivel--;
}

/***** Funcao que pesquisa item na tabela *****/
void Get_Entry(char x[10]) /* Pesquisa o simbolo "x" e retorna o
                           indice da Tabela de simbolos onde ele se encontra */

{
    int K, achou, aux, Esquerda, Direita;
    Esquerda = 1;
    Direita = L;
    achou = 0;
    while ((Esquerda <= Direita)&&(achou == 0))
    {
        K=((Esquerda+Direita)/2);
        aux = strcmpi(x,TabelaS[K].nome);
        if (aux == 0) achou = 1;
        else if (aux < 0) Direita = K-1;
        else Esquerda = K+1;
    }
    if (achou == 1)
    {
        printf("O item esta no nivel  %d", TabelaS[K].nivel);
        printf("                Indice %u",K);
    }
    else Erro(2);
}

/***** Funcao que instala o item na tabela *****/
void Instala(char X[10], char atribut[10]) /* Instala o simbolo "X" com
                                           o atributo
                                           de Simbolos */

{
    int i, j, inseriu, deuerro, k, aux;
    k = escopo[nivel];
    inseriu = 0;
    deuerro = 0;
    i = L;
    while ((inseriu == 0)&&(deuerro == 0))
    {
        if (L == escopo[nivel])
        {

```

```

        inseriu = 1;
        TabelaS[i].nivel = nivel;
        aux = strlen(atribut);
        for (j = 0; j<= aux-1; j++)
            TabelaS[i].atributo[j] = atribut[j];
        aux = strlen(X);
        for (j = 0; j<=(aux-1); j++)
            TabelaS[i].nome[j] = X[j];
    }
    else
    {
        while (k < L)
        {
            aux = strcmp(TabelaS[k].nome,X);
            if (aux == 0)
            {
                Erro(3);
                L--;
                deuerro = 1;
            }
            k++;
        }
        if (deuerro == 0)
        {
            while ((X < TabelaS[i-1].nome)&&(i >=
                escopo[nivel]+1))
            {
                TabelaS[i].nivel = TabelaS[i-1].nivel;
                aux = strlen(atribut);
                for (j = 0; j<= aux-1; j++)
                    TabelaS[i].atributo[j] =
                        TabelaS[i-1].atributo[j];
                aux = strlen(X);
                for (j = 0; j<=(aux-1); j++)
                    TabelaS[i].nome[j] =
                        TabelaS[i-1].nome[j];
                i--;
            }
            if (L == NMax + 1)
            {
                L--;
                Erro(1);
            }
            else
            {
                inseriu = 1;
                TabelaS[i].nivel = nivel;

```

```

        aux = strlen(atribut);
        for (j = 0; j<= aux-1; j++)
            TabelaS[i].atributo[j] =
                atribut[j];
        aux = strlen(X);
        for (j = 0; j<=(aux-1); j++)
            TabelaS[i].nome[j] = X[j];
    }
}
L++;
}
}

```

```

void imprimir()
{
    int i;
    for (i=0; i<=L-1; i++)
    {
        printf("\nNome : ");
        printf("%s", TabelaS[i].nome);
        printf("\n");
        printf("Atributo : ");
        printf("%s", TabelaS[i].atributo);
        printf("\n");
        printf("Nivel : ");
        printf("%i", TabelaS[i].nivel);
        printf("\n");
        // printf("\n");
    }
}

```

B.2 Abordagem Árvore Binária na Linguagem C

```

#include <string.h>
#include <conio.h>
#include <stdio.h>
#include <process.h>
#include <dos.h>

#define NMax 1000    /* Numero maximo de niveis possiveis */
int escopo[10];
int nivel;    /* inteiro que contem o numero do nivel atual */
int L;    /* inteiro que contem o indice do ultimo elemento da Tabela
            de Simbolos */
int Raiz;    /* inteiro que contem o indice do primeiro elemento da

```

```

        Tabela de Simbolos */

struct {
    char nome[10];    /* Contem o nome do Simbolo */
    int  nivel;       /* Contem o nivel do Simbolo relacionado */
    int  esq;         /* Filho da esquerda do simbolo relacionado
                        */
    int  dir;         /* Filho da direita do simbolo relacionado */
} TabelaS[100];      /* Vetor de struct que
                        contem a tabela de simbolos */

void Entrada_Bloco(void);
void Erro(int numero);
void Saida_Bloco(void);
void Get_Entry(char name[10]);
void Instala(char name[10], char atributo[10]);
void imprimir(void);

void main(void)
{
    L = 1;    /* Considera-se que a primeira posicao da tabela eh a de
               indice 0. L eh o final da arvore binaria */

    Raiz = 0;
    nivel = 1;    /* 0 primeiro nivel , o 1 */
    escopo[nivel] = 1; /* escopo[1] contem o indice do primeiro elemento
                        */
}

/*****
Implementacoes
*****/

/***** Funcao que define os erros provaveis de ocorrer
*****/

void Erro(int num)
{
    char opcao;
    switch (num) {
        case 1:
            printf("Tabela de Simbolos cheia\n"); break;
        case 2:
            printf("Este item nao foi encontrado\n"); break;
        case 3:

```

```

printf("Este item ja foi inserido\n"); break;
}

/***** Funcao de entrada num bloco *****/

void Entrada_Bloco()
{
    nivel++;
    if (nivel > NMax)
        Erro(1);
    else escopo[nivel] = L;
}

/***** Funcao de saida de um bloco *****/

void Saida_Bloco()
{
    L= escopo[nivel];
    nivel--;
    if (Raiz >= L)
        Raiz =0;
    else for (i=1;i<=(L-1); i++)
    {
        if (TabelaS[i].esq >= L)
            TabelaS[i].esq = 0;
        if (TabelaS[i].dir >= L)
            TabelaS[i].dir = 0;
    }
}

/***** Funcao que pesquisa item na tabela *****/

void Get_Entry(char x[10]) /* Pesquisa o simbolo "x" e retorna o
                           indice da Tabela de simbolos
                           onde ele se encontra */
{
    int S, K, aux;
    S = Raiz;
    K = 0;
    while (S != 0)
    {
        aux = strcmpi(x,TabelaS[S].nome);
        if (aux == 0)

```



```

        {
            K = S;
            S = TabelaS[S].dir;
        }
        else if (aux < 0)      S = TabelaS[S].esq;
            else S = TabelaS[S].dir;
    }
    if (K != 0) /*Verificar se eh -1 ou 0 */
    {
        printf("O item esta no nivel  %d", TabelaS[K].nivel);
        printf("                Indice %u",K);
    }
    else Erro(2);
}

/***** Funcao que instala o item na tabela *****/
void Instala(char X[10], char atribut[10]) /* Instala o simbolo "X" com
                                           o atributo
                                           de Simbolos */
{
    int S, i, k, aux, achou;
    S = Raiz;
    i = Raiz;
    k = 0;
    achou=0;
    while (S != 0)    /* Enquanto nao achar um nodo folha  */
    {
        i=S;
        aux = strcmpi(X,TabelaS[S].nome);
        if (aux == 0)
        { k = S;
          S = TabelaS[S].dir;
          achou=1;
          if (i >= escopo[nivel]) Erro(3);
        }
        else if (aux <0) S = TabelaS[S].esq;
            else S = TabelaS[S].dir;
    }
    if ((k <= escopo[nivel])&&(L < NMax + 1))
    {
        if (achou==0)
        {
            TabelaS[L].nivel = nivel;
            aux = strlen(atribut);
            for (k = 0; k<= aux-1; k++)
                TabelaS[L].atributo[k] = atribut[k];

```

```

        TabelaS[L].esq = TabelaS[L].dir = 0;
        aux = strlen(X);
        for (k = 0; k<=(aux-1); k++)
            TabelaS[L].nome[k] = X[k];
        if (Raiz == 0) Raiz= L;
        else if (X < TabelaS[i].nome)
            TabelaS[i].esq = L;
        else TabelaS[i].dir = L;
        L++;
    }
}

void imprimir()
{
    int i;
    clrscr();
    for (i=0; i<=L-1; i++)
    {
        printf("\n\nNome : ");
        printf("%s", TabelaS[i].nome);
        printf("\n");
        printf("Atributo : ");
        printf("%s", TabelaS[i].atributo);
        printf("\n");
        printf("Nivel : ");
        printf("%i", TabelaS[i].nivel);
        printf("\n");
        printf("Esquerdo : ");
        if (TabelaS[i].esq == -1) printf("0");
        else printf("%i", TabelaS[i].esq);
        printf("\n");
        printf("Direito : ");
        if (TabelaS[i].dir == -1) printf("0");
        else printf("%i", TabelaS[i].dir);
        printf("\n");
        printf("\n"); }
}

```

B.3 Abordagem Floresta de Árvore Binária na Linguagem C

```

#include <string.h>
#include <conio.h>
#include <stdio.h>
#include <process.h>
#include <dos.h>

```

```

#define NMax 1000      /* Numero maximo de niveis possiveis */

int escopo[10];
int nivel;      /* inteiro que contem o numero do nivel atual */
int L;      /* inteiro que contem o indice do ultimo elemento da Tabela
              de Simbolos */
int Raiz;      /* inteiro que contem o indice do primeiro elemento da
              Tabela de Simbolos */

struct {
    char nome[10];      /* Contem o nome do Simbolo */
    int nivel;      /* Contem o nivel do Simbolo relacionado */
    char atributo[10]; /* Contem o atributo do relacionado */
    int esq;      /* Filho da esquerda do simbolo relacionado
                  */
    int dir;      /* Filho da direita do simbolo relacionado */
} TabelaS[100];      /* Vetor de struct que
                    contem a tabela de simbolos */

void Entrada_Bloco(void);
void Erro(int numero);
void Saida_Bloco(void);
void Get_Entry(char name[10]);
void Instala(char name[10], char atributo[10]);
void imprimir(void);
void main(void)
{
    L = 1;      /* Considera-se
                que a primeira posicao da tabela eh a de indice 1.
                L eh o final da arvore binaria */
    nivel = 1;      /* 0 primeiro nivel , o 1 */
    escopo[nivel] = 0; /* escopo[1] contem o indice do primeiro elemento
                  */
}

/*****
Implementacoes
*****/

/***** Funcao que define os erros provaveis de ocorrer
*****/

void Erro(int num)
{
    char opcao;
    switch (num) {
        case 1:

```

```

printf("Tabela de Simbolos cheia\n"); break;

printf("Este item nao foi encontrado\n"); break;

printf("Este item ja foi inserido\n"); break;
}

/***** Funcao de entrada num bloco
*****/
void Entrada_Bloco()
{
    nivel++;
    if (nivel > NMax)
        Erro(1);
    else escopo[nivel] = 0;
}

/***** Funcao de saida de um bloco
*****/

void Saida_Bloco()
{
    if (escopo[nivel] != 0) L = escopo[nivel];
    nivel--;
}

/***** Funcao que pesquisa item na tabela
*****/

void Get_Entry(char x[10]) /* Pesquisa o simbolo "x" e retorna o
                           indice da Tabela de simbolos
                           onde ele se encontra */
{
    int n, k, aux, achou;
    achou = 0;
    n = nivel;
    while ((n > 0)&&(achou==0))
    {
        k = escopo[n];
        while ((k != 0)&&(achou == 0))
        {
            aux = strcmpi(x,TabelaS[k].nome);
            if (aux == 0) achou = 1;
            else if (aux < 0) k = TabelaS[k].esq;
            else k = TabelaS[k].dir;
        }
    }
}

```

```

        n--;
    }
    if (achou == 1)
    {
        printf("O item esta no nivel: %d", TabelaS[k].nivel);
        printf("          Indice: %u",k);
    }
    else Erro(2);
}

/***** Funcao que instala o item na tabela *****/

void Instala(char X[10], char atribut[10]) /* Instala o simbolo "X" com
                                          o atributo na Tabela
                                          de Simbolos */
{
    int S, i, k, aux;
    aux = 1;
    S = escopo[nivel];
    while (S != 0) /* Enquanto nao achar um nodo folha */
    {
        i=S;
        aux = strcmpi(X,TabelaS[S].nome);
        if (aux == 0)
        {
            S = 0;
            Erro(3);
        }
        else if (aux < 0) S = TabelaS[S].esq;
            else S = TabelaS[S].dir;
    }
    if ((L< NMax + 1)&&(aux!=0))
    {
        TabelaS[L].nivel = nivel;
        aux = strlen(atribut);
        for (k = 0; k<= aux-1; k++)
            TabelaS[L].atributo[k] = atribut[k];
        TabelaS[L].esq = TabelaS[L].dir = 0;
        aux = strlen(X);
        for (k = 0; k<=(aux-1); k++)
            TabelaS[L].nome[k] = X[k];
        if (escopo[nivel] == 0) escopo[nivel] = L;
        else if (X < TabelaS[i].nome)
            TabelaS[i].esq = L;
            else TabelaS[i].dir = L;
        L++;
    }
}

```

```

    }
}

void imprimir()
{
    int i;
    for (i=1; i<=L-1; i++)
    {
        printf("\n\nNome : ");
        printf("%s", TabelaS[i].nome);
        printf("\n");
        printf("Atributo : ");
        printf("%s", TabelaS[i].atributo);
        printf("\n");
        printf("Nivel : ");
        printf("%i", TabelaS[i].nivel);
        printf("\n");
        printf("Esquerdo : ");
        printf("%i", TabelaS[i].esq);
        printf("\n");
        printf("Direito : ");
        printf("%i", TabelaS[i].dir);
        printf("\n");
        printf("\n"); }
}

```

B.4 Abordagem *Hash* na Linguagem C

```

#include <string.h>
#include <conio.h>
#include <stdio.h>
#include <process.h>
#include <stdlib.h>
#include <dos.h>
#define NMax 1000      /* Numero maximo de niveis possiveis */
int escopo[10];
int TabHash[509];
int nivel;      /* inteiro que contem o numero do nivel atual */
int L;          /* inteiro que contem o indice do ultimo elemento da Tabela
                  de Simbolos */
int Raiz;       /* inteiro que contem o indice do primeiro elemento da
                  Tabela de Simbolos */

struct {
    char nome[10];      /* Contem o nome do Simbolo */
    int nivel;          /* Contem o nivel do Simbolo relacionado */
    char atributo[10];  /* Contem o atributo do relacionado */
}

```

```

        int col;

        } TabelaS[100]; /* Vetor de struct que contem a
                           tabela de  simbolos */

void Entrada_Bloco(void);
void Erro(int numero);
void Saida_Bloco(void);
void Get_Entry(char name[10]);
void Instala(char name[10], char atributo[10]);
void imprimir(void);

void main(void)
{
    L = 1;      /* Considera-se que a primeira posicao da tabela eh a de
                  indice 1. L eh o final da  arvore binaria */

    Raiz = 0;
    nivel = 1;      /* 0 primeiro nivel , o 1 */
    escopo[nivel] = 1;      /* escopo[1] contem o indice do primeiro
                              elemento */
}

/*****
                                Implementacoes
*****/

/*****  Funcao que define os erros provaveis de ocorrer
*****/

void Erro(int num)
{
    char opcao;
    switch (num) {

                                                                    case 1:
printf("Tabela de Sibolos cheia\n");    break;

                                                                    case 2:
printf("Este item nao foi encontrado\n");  break;

                                                                    case 3:
printf("Este item ja foi inserido\n"); break;

                                                                    }
}

/*****  Funcao de entrada num bloco
*****/

void Entrada_Bloco()
{
    nivel++;

```

```

    if (nivel > NMax)
        Erro(1);
    else escopo[nivel] = L;
}

/***** Funcao de saida de um bloco
*****/

void Saida_Bloco()
{
    int S, B, k;
    char ident[10];
    S = L;
    B = escopo[nivel];
    while (S > B)
    {
        S--;
        strcpy(ident,TabelaS[S].nome);
        k = ident[1];
        TabHash[k] = TabelaS[S].col;
    }
    nivel--;
    L = B;
}

/***** Funcao que pesquisa item na tabela
*****/

void Get_Entry(char x[10])    /* Pesquisa o simbolo "x" e retorna o
                               indice da Tabela de simbolos
                               onde ele se encontra */
{
    int n, aux, achou, k;
    achou = 0;
    n = x[0];
    k = TabHash[n];
    while ((k != 0)&&(achou == 0))
    {
        aux = strcmpi(TabelaS[k].nome,x);
        if (aux == 0) achou = 1;
        else k = TabelaS[k].col;
    }
    if (achou == 1)
    {
        printf("\0 item esta no nivel  %d", TabelaS[k].nivel);
        printf("                Indice %u",k);
    }
}

```



```

    else Erro(2);
}

/***** Funcao que instala o item na tabela
*****/

void Instala(char X[10], char atribut[10]) /* Instala o simbolo "X" com
                                         o atributo
                                         na Tabela de Simbolos */
{
    int n, igual, k, aux;
    igual = 0;
    n = X[0]; /* Calcula o hashing(X)*/
    k = TabHash[n];
    while (k >= escopo[nivel])
    {
        aux = strcmpi(TabelaS[k].nome,X);
        if (aux == 0)
        { Erro(3);
          igual = 1;
        }
        k = TabelaS[k].col;
    }
    if (L == NMax + 1) Erro(1);
    else if (igual == 0)
    {
        TabelaS[L].nivel = nivel;
        aux = strlen(atribut);
        for (k = 0; k<= aux-1; k++)
            TabelaS[L].atributo[k] = atribut[k];
        aux = strlen(X);
        for (k = 0; k<=(aux-1); k++)
            TabelaS[L].nome[k] = X[k];
        TabelaS[L].col = TabHash[n];
        TabHash[n] = L;
        L++;
    }
}

void imprimir()
{
    int i;
    for (i=1; i<=L-1; i++)
    {
        printf("\n\nNome : ");
        printf("%s", TabelaS[i].nome);
        printf("\n");
    }
}

```

```
printf("Atributo : ");  
printf("%s", TabelaS[i].atributo);  
printf("\n");  
printf("Nivel : ");  
printf("%i", TabelaS[i].nivel);  
printf("\n");  
printf("\n"); }  
}
```

C Fontes dos Programas na Linguagem JAVA

C.1 Abordagem Linear na Linguagem JAVA

C.1.1 Lista Não Ordenada

```
import java.io.*;
import java.util.*;

        public class ListaNO extends Thread {
            final static int NMax = 1000;      /*
            Numero maximo de niveis possiveis */
            int escopo[] = new int[10];
            int nivel;      /* inteiro que contem o
            numero do nivel atual */
            int L;          /* inteiro que contem o
            indice do ultimo elemento da Tabela de Simbolos */
            int Raiz;       /* inteiro que contem o
            indice do primeiro elemento da Tabela de Simbolos */
            TabelaS tb = new TabelaS();

public void run() {
    L = 0;      /* Considera-se que a primeira posicao da tabela eh a de
                indice 1. L eh o final da arvore binaria */
    Raiz = -1;
    nivel = 1;      /* 0 primeiro nivel , o 1 */
    escopo[nivel] = 0;      /* escopo[1] contem o indice do primeiro
                            elemento */
}

public static void main(String argv[]) {
    ListaNO lno = new ListaNO();
    lno.start();
}

/*****
                                Implementacoes
*****/

/***** Funcao que define os erros provaveis de ocorrer
*****/

void Erro(int num)
{
    switch (num) {
        case 1: System.out.print("Tabela de Sibolos
                                cheia\n"); break;
```

```

        case 2: System.out.print("Este item nao foi
                               encontrado\n"); break;
        case 3: System.out.print("Este item ja foi
                               inserido\n"); break;
        default: ;
    }

    /*****          Funcao de entrada num bloco
    *****/

    void Entrada_Bloco()
    {
        nivel++;
        if (nivel > NMax) Erro(1);
        else escopo[nivel] = L;
    }

    /*****          Funcao de saida de um bloco
    *****/

    void Saida_Bloco()
    {
        L = escopo[nivel];
        nivel--;
    }

    /*****          Funcao que pesquisa item na tabela
    *****/

    void Get_Entry(String x)    /* Pesquisa o simbolo "x" e retorna o indice
                                da Tabela de simbolos
                                onde ele se encontra */

    {
        int S, K, aux, achou;
        achou = 0;
        K = L;
        while ((K > 0)&&(achou == 0))
        {
            K--;
            if(x.equals(tb.nome[K])) achou = 1;
        }
        if (achou == 1)
        {
            System.out.print("O item esta no nivel  %d", TabelaS[k].nivel);
            System.out.print("          Indice %u",k);
        }
        else
        {

```

```

        Erro(2);
    }
}

/***** Funcao que instala o item na tabela
*****/

void Instala(String X, String atribut) /* Instala o simbolo "X" com o
                                     atributo na Tabela de Simbolos */
{
    int igual, k, aux;
    igual = 0;
    k = L;
    while (k > escopo[nivel])
    {
        k--;
        if(X.equals(tb.nome[k])) { igual = 1; }
    }
    if (L == NMax + 1)      Erro(1);
    else if (igual == 0)
    {
        tb.nivel[L] = nivel;
        tb.atributo[L] = atribut;
        tb.nome[L] = X;
        L++;
    }
}

}

}

class TabelaS {
    String nome[] = new String[100] ;
    String atributo[] = new String[100];
    int nivel[] = new int[100];
}

```

C.1.2 Lista Ordenada

```

import java.io.*;
import java.util.*;

public class Lista0 extends Thread {
    final static int NMax = 100000; /*
    Numero maximo de niveis possiveis */
    int escopo[] = new int[10];
    int nivel; /* inteiro que contem o
               numero do nivel atual */
    int L; /* inteiro que contem o

```

```

        indice do ultimo elemento da Tabela de Simbolos */
        int Raiz;    /* inteiro que contem o
            indice do primeiro elemento da Tabela de Simbolos */
        TabelaS tb = new TabelaS();

public void run() {
    L = 1;    /* Considera-se que a primeira posicao da tabela eh a de
                indice 1. L eh o final da  arvore binaria */

    Raiz = 0;
    nivel = 1;    /* 0 primeiro nivel , o 1 */
    escopo[nivel] = 1;    /* escopo[1] contem o indice do primeiro
                            elemento */
}

public static void main(String argv[]) {
    Lista0 lo = new Lista0();
    lo.start();
}

/*****
                                Implementacoes
*****/

/***** Funcao que define os erros provaveis de ocorrer
*****/

void Erro(int num)
{
    switch (num) {
        case 1: System.out.print("Tabela de Sibolos
                                cheia\n"); break;
        case 2: System.out.print("Este item nao foi
                                encontrado\n"); break;
        case 3: System.out.print("Este item ja foi
                                inserido\n"); break;
        default: ;
    }
}

/***** Funcao de entrada num bloco
*****/

void Entrada_Bloco()
{
    nivel++;
    if (nivel > NMax) Erro(1);
    else escopo[nivel] = L;
}

```

```

/***** Funcao de saida de um bloco
*****/

void Saida_Bloco()
{
    L = escopo[nivel];
    nivel--;
}

/***** Funcao que pesquisa item na tabela
*****/

void Get_Entry(String x)    /* Pesquisa o simbolo "x" e retorna o indice
                               da Tabela de simbolos
                               onde ele se encontra */

{
    int K, aux, achou, Esquerda, Direita;
    Esquerda = 1;
    Direita = L;
    achou = 0;
    while ((Esquerda <= Direita)&&(achou == 0))
    {
        aux = Esquerda + Direita;
        K = aux/2;
        int comp= x.compareTo(tb.nome[K]);
        if (comp == 0) achou = 1;
        else if (comp < 0) Direita = K-1;
            else Esquerda = K +1;
    }
    if (achou != 1) Erro(2);
}

/***** Funcao que instala o item na tabela
*****/

void Instala(String X, String atribut) /* Instala o simbolo "X" com o
                                         atributo na Tabela de Simbolos */

{
    int i, inseriu, deuerro, k, aux;
    k = escopo[nivel];
    inseriu = 0;
    deuerro = 0;
    i = L;
    while ((inseriu == 0)&&(deuerro == 0))
    {
        if (L == escopo[nivel])
        {

```

```

        inseriu = 1;
        tb.nivel[i] = nivel;
        tb.atributo[i] = atribut;
        tb.nome[i] = X;
    }
    else
    {
        while (k < L)
        {
            if(X.equals(tb.nome[k]))
            {
                deuerro = 1;
                L--;
            }
            k++;
        }
        if (deuerro == 0)
        {
            int comp= X.compareTo(tb.nome[i-1]);
            while ((comp < 0)&&(i >=
                (escopo[nivel]+1)))
            {
                tb.nivel[i] = tb.nivel[i-1];
                tb.atributo[i] = tb.atributo[i-1];
                tb.nome[i] = tb.nome[i];
                i--;
            }
            if (L == NMax +1)
            {
                L--;
                Erro(1);
            }
            else
            {
                inseriu = 1;
                tb.nivel[i] = nivel;
                tb.atributo[i] = atribut;
                tb.nome[i] = X;
            }
        }
    }
    L++;
}
}
}

```

```

class TabelaS {
    String nome[] = new String[100] ;

```



```
String atributo[] = new String[100];
int nivel[] = new int[100];
}
```

C.2 Abordagem Árvore Binária na Linguagem JAVA

```
import java.io.*;
import java.util.*;

public class Arvore extends Thread {
    final static int NMax = 100000;    /*
        Numero maximo de niveis possiveis */
    int escopo[] = new int[10];
    int nivel;    /* inteiro que contem o
        numero do nivel atual */
    int L;    /* inteiro que contem o
        indice do ultimo elemento da Tabela de Simbolos */
    int Raiz;    /* inteiro que contem o
        indice do primeiro elemento da Tabela de Simbolos */
    TabelaS tb = new TabelaS();

    public void run() {
        L = 1;    /* Considera-se que a primeira posicao da tabela eh a de
            indice 1. L eh o final da arvore binaria */
        Raiz = 0;
        nivel = 1;    /* 0 primeiro nivel eh o 1 */
        escopo[nivel] = 1;    /* Escopo[0] contem o indice do primeiro
            elemento */
    }

    public static void main(String argv[]) {
        Arvore ar = new Arvore();
        ar.start();
    }

    /*****
        Implementacoes
    *****/

    /***** Funcao que define os erros provaveis de ocorrer
    *****/

    void Erro(int num)
    {
        switch (num) {
            case 1: System.out.print("Tabela de Sibolos
                cheia\n"); break;
        }
    }
}
```

```

        case 2: System.out.print("Este item nao foi
                               encontrado\n"); break;
        case 3: System.out.print("Este item ja foi
                               inserido\n"); break;
        default: ;
    }

    /*****          Funcao de entrada num bloco
    *****/

    void Entrada_Bloco()
    {
        nivel++;
        if (nivel > NMax) Erro(1);
        else escopo[nivel] = L;
    }

    /*****          Funcao de saida de um bloco
    *****/

    void Saida_Bloco()
    {
        L = escopo[nivel];
        if (Raiz >= L) Raiz = 0;
        else
            for (int i=1; i <= (L-1); i++)
            {
                if (tb.esq[i] >= L) tb.esq[i] = 0;
                if (tb.dir[i] >= L) tb.dir[i] = 0;
            }
        nivel--;
    }

    /*****          Funcao que pesquisa item na tabela
    *****/

    void Get_Entry(String x)    /* Pesquisa o simbolo "x" e retorna o indice
                                da Tabela de simbolos
                                onde ele se encontra */

    {
        int S, aux, k;
        S = Raiz;
        k = 0;
        while (S != 0)
        {
            int compara = x.compareTo(tb.nome[S]);
            if(compara == 0)

```

```

    { k = S;
      S = tb.dir[S];
    }
    else if (compara < 0) S = tb.esq[S];
        else S = tb.dir[S];
    }
    if (k == 0) Erro(2);
}

/***** Funcao que instala o item na tabela
*****/

void Instala(String X, String atribut) /* Instala o simbolo "X" com
                                     o atributo na Tabela de Simbolos */
{
    int S, i, k, aux, achou;
    S = Raiz;
    i = Raiz;
    k = 0;
    achou = 0;
    while (S != 0)
    {
        i = S;
        int compara = X.compareTo(tb.nome[S]);
        if(compara == 0)
        {
            k = S;
            S = tb.dir[S];
            achou = 1;
        }
        else if (compara < 0) S = tb.esq[S];
            else S = tb.dir[S];
    }
    if ((L < NMax + 1)&&(k <= escopo[nivel]))
    {
        if (achou == 0)
        {
            tb.nivel[L] = nivel;
            tb.atributo[L] = atribut;
            tb.nome[L] = X;
            tb.esq[L] = 0;
            tb.dir[L] = 0;
            if (Raiz == 0) Raiz = L;
            else
            {
                int comp = X.compareTo(tb.nome[i]);
                if (comp < 0)tb.esq[i] = L;
            }
        }
    }
}

```

```

        else tb.dir[i] = L;
    }
    L++;
}
}
}
}

```

```

class TabelaS {
    String nome[] = new String[100] ;
    String atributo[] = new String[100];
    int nivel[] = new int[100];
    int esq[] = new int[100];
    int dir[] = new int[100];
}

```

C.3 Abordagem Floresta de Árvores Binárias na Linguagem JAVA

```

import java.io.*;
import java.util.*;

public class Floresta extends Thread {
    final static int NMax = 100000; /*
        Numero maximo de niveis possiveis */
    int escopo[] = new int[10];
    int nivel; /* inteiro que contem o
        numero do nivel atual */
    int L; /* inteiro que contem o
        indice do ultimo elemento da Tabela de Simbolos */
    int Raiz; /* inteiro que contem o
        indice do primeiro elemento da Tabela de Simbolos */
    TabelaS tb = new TabelaS();

    public void run() {
        L = 1; /* Considera-se que a primeira posicao da tabela eh a de
            indice 1. L eh o final da arvore binaria */

        Raiz = 0;
        nivel = 1; /* 0 primeiro nivel eh o 1 */
        escopo[nivel] = 0; /* Escopo[0] contem o indice do primeiro
            elemento */
    }

    public static void main(String argv[]) {
        Floresta f = new Floresta();
        f.start();
    }
}

```

```

/*****
                                Implementacoes
*****/

/***** Funcao que define os erros provaveis de ocorrer
*****/

void Erro(int num)
{
    switch (num) {
        case 1: System.out.print("Tabela de Sbolos
                                cheia\n"); break;
        case 2: System.out.print("Este item nao foi
                                encontrado\n"); break;
        case 3: System.out.print("Este item ja foi
                                inserido\n"); break;
        default: ;
    }
}

/***** Funcao de entrada num bloco
*****/

void Entrada_Bloco()
{
    nivel++;
    if (nivel > NMax) Erro(1);
    else escopo[nivel] = 0;
}

/***** Funcao de saida de um bloco
*****/

void Saida_Bloco()
{
    if (escopo[nivel] != 0) {L = escopo[nivel];}
    nivel--;
}

/***** Funcao que pesquisa item na tabela
*****/

void Get_Entry(String x) /* Pesquisa o simbolo "x" e retorna o indice
                        da Tabela de simbolos onde ele se encontra */
{
    int n, aux, achou, k, n1, n2;
    achou = 0;
    n = nivel;

```

```

while ((n > 0)&&(achou == 0))
{
    k = escopo[n];
    while ((k != 0)&&(achou == 0))
    {
        int compara = x.compareTo(tb.nome[k]);
        if(compara == 0) {achou = 1;}
        else if (compara < 0) {k = tb.esq[k];}
        else {k = tb.dir[k];}
    }
    n--;
}
if (achou == 1)
{
    System.out.print("O item esta no nivel  %d", TabelaS[k].nivel);
    System.out.print("          Indice %u",k);
}
else
{
    Erro(2);
}
}

/***** Funcao que instala o item na tabela
*****/

void Instala(String X, String atribut) /* Instala o simbolo "X" com
                                     o atributo na Tabela de Simbolos */
{
    int S, i, k, aux;
    aux = 1;
    S = escopo[nivel];
    i = S;
    while (S != 0)
    {
        i = S;
        int compara = X.compareTo(tb.nome[S]);
        if(compara == 0)
        {
            S = 0;
            aux = 0;
        }
        else if (compara < 0){S = tb.esq[S];}
        else {S = tb.dir[S];}
    }
    if ((L < NMax + 1)&&(aux != 0))
    {

```

```

        tb.nivel[L] = nivel;
        tb.atributo[L] = atributo;
    tb.nome[L] = X;
    tb.esq[L] = 0;
    tb.dir[L] = 0;
    if (escopo[nivel] == 0) escopo[nivel] = L;
    else {
        int comp = X.compareTo(tb.nome[i]);
        if (comp < 0) tb.esq[i] = L;
        else tb.dir[i] = L;
    }
    L++;
}
}
}

```

```

class TabelaS {
    String nome[] = new String[100] ;
    String atributo[] = new String[100];
    int nivel[] = new int[100];
    int esq[] = new int[100];
    int dir[] = new int[100];
}

```

C.4 Abordagem *Hash* na Linguagem JAVA

```

import java.io.*;
import java.util.*;

    public class Hash extends Thread {
        final static int NMax = 1000;      /*
            Numero maximo de niveis possiveis */
        int escopo[] = new int[10];
        int TabHash[] = new int[509];
        int nivel;      /* inteiro que contem o
            numero do nivel atual */
        int L;          /* inteiro que contem o
            indice do ultimo elemento da Tabela de Simbolos */
        int Raiz;       /* inteiro que contem o
            indice do primeiro elemento da Tabela de Simbolos */
        TabelaS tb = new TabelaS();

    public void run() {
        L = 1;      /* Considera-se que a primeira posicao da tabela eh a de
            indice 1.  L eh o final da  arvore binaria */
        Raiz = 0;
    }
}

```

```

    nivel = 1;          /* 0 primeiro nivel , o 1 */
    escopo[nivel] = 1;   /* escopo[1] contem o indice do primeiro
                           elemento */
}

public static void main(String argv[]) {
    Hash h = new Hash();
    h.start();
}

/*****
                                Implementacoes
*****/

/***** Funcao que define os erros provaveis de ocorrer
*****/

void Erro(int num)
{
    switch (num) {
        case 1: System.out.print("Tabela de Sıbolos
                                cheia\n"); break;
        case 2: System.out.print("Este item nao foi
                                encontrado\n"); break;
        case 3: System.out.print("Este item ja foi
                                inserido\n"); break;
        default: ;
    }
}

/***** Funcao de entrada num bloco
*****/

void Entrada_Bloco()
{
    nivel++;
    if (nivel > NMax) Erro(1);
    else escopo[nivel] = L;
}

/***** Funcao de saida de um bloco
*****/

void Saida_Bloco()
{
    int S, B, k;
    String ident;
    S = L;
}

```



```

    B = escopo[nivel];
    while (S > B)
    {
        S--;
        //strcpy(ident,TabelaS[S].nome);
        k = (int)tb.nome[S].charAt(0);
        TabHash[k] = tb.col[S];
    }
    L = escopo[nivel];
    nivel--;
}

/***** Funcao que pesquisa item na tabela
*****/

void Get_Entry(String x) /* Pesquisa o simbolo "x" e retorna o indice
                           da Tabela de simbolos
                           onde ele se encontra */
{
    int n, aux, achou, k;
    achou = 0;
    n = (int)x.charAt(0);
    k = TabHash[n];
    while ((k != 0)&&(achou == 0))
    {
        if(x.equals(tb.nome[k])) achou = 1;
        else k = tb.col[k];
    }
    if (achou == 1)
    {
        System.out.print("O item esta no nivel  %d", TabelaS[k].nivel);
        System.out.print("          Indice %u",k);
    }
    else
    {
        Erro(2);
    }
}

/***** Funcao que instala o item na tabela
*****/

void Instala(String X, String atribut) /* Instala o simbolo "X" com
                                         o atributo na Tabela de Simbolos */
{
    int n, igual, k, aux;
    igual = 0;

```

```

n = (int)X.charAt(0); /* Calcula o hashing(X)*/
k = TabHash[n];
while (k >= escopo[nivel])
{
    if(X.equals(tb.nome[k])) { igual = 1; }
    k = tb.col[k];
}
if (L == NMax + 1)      Erro(1);
else if (igual == 0)
{
    tb.nivel[L] = nivel;
    tb.atributo[L] = atribut;
tb.nome[L] = X;
    tb.col[L] = TabHash[n];
    TabHash[n] = L;
    L++;
}
}
}

class TabelaS {
    String nome[] = new String[100] ;
    String atributo[] = new String[100];
    int nivel[] = new int[100];
    int col[] = new int[100];
}

```

Belo Horizonte, 5 de março de 1999

Yêdda Aires Duarte Silva

Mariza Andrade da Silva Bigonha