Compiladores

Tabela de Símbolos

Introdução

Este material contém as notas de aula sobre *Tabela de Símbolos*, da disciplina *Compiladores*. O objetivo é apresentar estruturas de dados possíveis de implementar a tabela, bem como exemplos.

Tabela de Símbolos

A tabela de símbolos é um componente utilizado em todas as fases do compilador. As entradas da tabela são criadas durante a análise, pelo *scanner*, pelo *parser* e pelo analisador semântico.

A tabela é um tipo abstrato de dados, implementado utilizando alguma estrutura de dados[1]. Ela deve funcionar como um dicionário, associando uma chave a um registro, e contém dados sobre os identificadores usados no programa sendo traduzido.

CURIOSIDADE

Algumas implementações podem trazer palavras reservadas da linguagem na tabela de símbolos. Em outras implementações, usa-se uma tabela separada para as palavras reservadas.

[1] Nesse contexto, a expressão *estrutura de dados* pode ser compreendida como qualquer coisa que armazene símbolos, que, por sua vez, podem ser chamados de *nomes*.

Uma estrutura possível para a tabela, em um formato resumido, está presente na Figura 1. Nela um código composto por expressões que utilizam operações e identificadores, tais como

```
a = b + c
d = a - b
c = a + d
```

Fazem com que uma tabela seja gerada com quatro registros, um para cada identificador (`a`, `b`, `c`, `d`) e seu respectivo endereço de memória. Assim, toda vez que um identificador for usado, consultando a tabela é possível saber em que endereço ele se encontra.

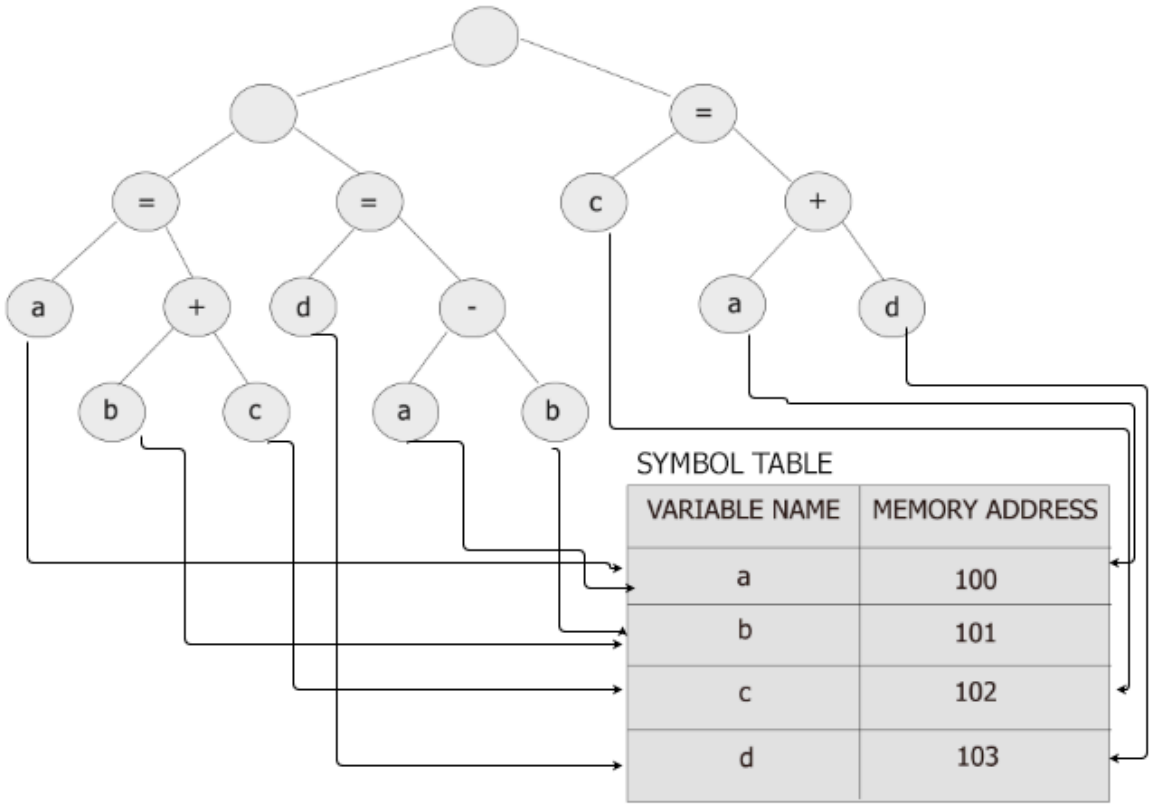


Figura 1. Tabela de símbolos resumida.

Os dados da tabela de símbolos são aqueles que o compilador não conhece. Linguagens possuem conjuntos de lexemas predefinidos, mas os identificadores são nomes não conhecidos no momento da compilação. Por esse motivo, caso algum lexema utilizado não contenha erros e não represente um símbolo conhecido, é necessário consultar a tabela de símbolos.

Exemplos de dados a serem armazenados na tabela de símbolos:

- lexema("tmp")
- tipo
- precisão
- classificação (variável, classe, parâmetro, procedimento etc.)
- se é uma função/método
- quais são seus argumentos e tipos
- tipo de passagem de parâmetro (valor ou referência)
- área de memória estática
- escopo
- (...)

Entre as possíveis estruturas para implementar a tabela de símbolos estão:

- **tabela de dispersão** (**tabela *hash***), onde é feito um *hash* na *string* que representa o identificador.
- **lista**, utilizando ***array*** para armazenar registro por registro e armazenando a próxima posição livre, ou **encadeada** (***linked list***), ligando os registros um a um.
- **árvore de busca binária**, estrutura que permite que cada nó tenha somente dois filhos.
- **múltiplas tabelas**, uma para cada escopo, em estrutura hierárquica (Figuras 2 e 3). Isso pode ser feito, por exemplo, em **árvores** ou **pilhas**.
    - Vantagem: é fácil saber quando uma tabela de símbolos não é mais necessária para poder encerrar seu escopo.
    - Desvantagens: dificuldade em abrir escopos, mais complexidade, perda de espaço (se cada tabela for implementada com *array* ou *hash*)

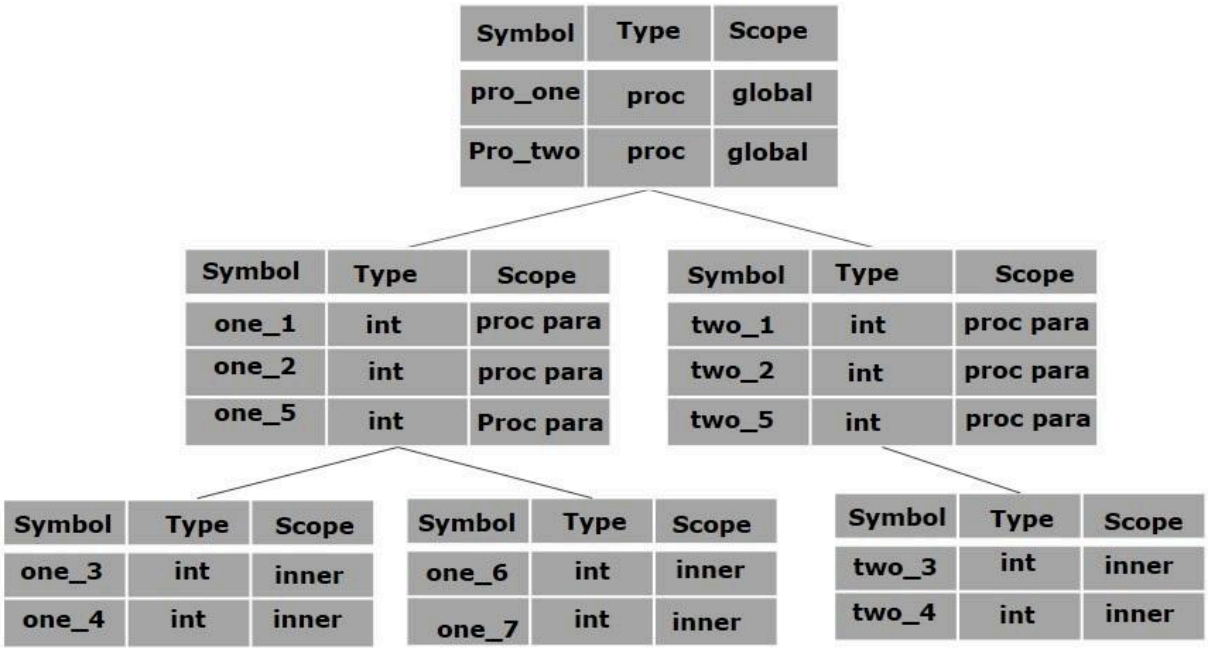
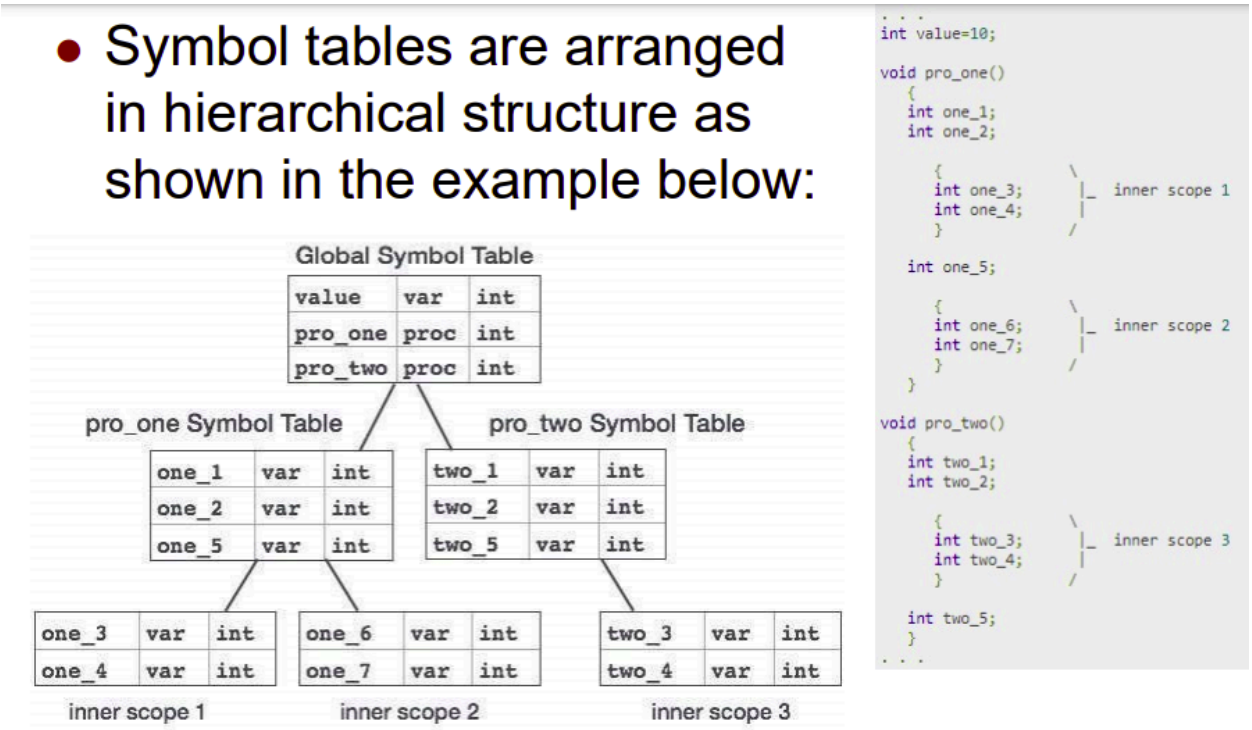


Figura 2. Estrutura hierárquica de tabelas de símbolos. Fonte: Tutorialspoint, s.d.

Figura 3. Escopos das tabelas de símbolos em estrutura hierárquica. Fonte: ALI, s.d.



A escolha da estrutura de dados deve ser cuidadosa, pois influencia a eficiência. A estrutura

escolhida também é influenciada pela linguagem que está sendo implementada, e, em particular, por suas regras de escopo. Cada vez que o programa se depara com um identificador, a tabela é acessada para encontrá-lo e buscar seus dados. Caso existam novos dados no programa, a tabela é atualizada.

Para variáveis, costuma-se registrar:

- classe(var)
- tipo
- endereço no texto
- precisão
- tamanho

Para parâmetros, costuma-se registrar:

- classe (par)
- tipo
- mecanismo de passagem

Para procedimentos, costuma-se registrar:

- classe (proc)
- número de parâmetros.

Um exemplo a partir do código do quadro a seguir:

```
int soma(int x, int y) {
    int resultado;
    resultado = x + y;
    return resultado;
}
```

Neste exemplo, a tabela poderia ser da seguinte maneira:

ID	**Nome**	**Classe**	**Tipo**
1	soma	proc	int

2	x	par	int
3	y	par	int
4	resultado	var	int

Outra tabela possível seria:

ID	**Nome**	**Classe**	**Tipo**	**Nível (de bloco)**
1	soma	proc	int	1
2	x	par	int	2
3	y	par	int	2
4	resultado	var	int	2

Na segunda tabela, o nível de bloco indicaria que a entrada faz parte de um escopo interno. Porém, esse exemplo ainda não resolve um problema: se existir também a função `subtrai`, que também recebe dois parâmetros `int x` e `int y`. Estes parâmetros também seriam de nível 2 e os registros seriam exatamente iguais aos de ID 2 e 3.

Nesse caso, uma possível solução seria identificar a linha da tabela de símbolos que contém o seu escopo. Por exemplo, no caso da função `soma`, seria necessário que todos os símbolos internos indicassem referência à linha 1, assim sabe-se que todos são parte daquela função. Dessa forma, uma outra versão da tabela ficaria assim:

ID	**Nome**	**Classe**	**Tipo**	**Nível**	**Referência**
1	soma	proc	int	1	(null)
2	x	par	int	2	1
3	y	par	int	2	1
4	resultado	var	int	2	1

Se a implementação fosse hierárquica, seria necessário utilizar aqui duas tabelas, uma contendo o escopo externo e outra para o interno. A tabela global seria:

ID	**Nome**	**Classe**	**Tipo**

1	soma	proc	int

A tabela local relacionada à função `soma` seria:

ID	Nome	Classe	Tipo
1	x	par	int
2	y	par	int
3	resultado	var	int

Novamente, assim como ocorre com outros conceitos, pode-se encontrar na literatura várias formas diferentes de se representar a tabela de símbolos.

Independentemente da estrutura escolhida para implementar a tabela de símbolos, devem ser implementadas pelo menos duas operações: busca e inserção.

Tabela de símbolos real:

Compila o código

`gcc hello-world.c`

Mostra as seções do código objeto e a tabela de símbolos

`objdump -ht a.out`

(disponível no arquivo objdump-hrt-saida-hello-world-tabela-de-simbolos.dat)

Filtra por termos específicos

`objdump -ht a.out | grep main`

`objdump -ht a.out | grep puts`

Outra forma

`nm a.out`

Mas por que aparece `puts` se a função de escrita utilizada foi `printf`? `puts` é uma função da `libc`, a biblioteca padrão da linguagem C. Esse símbolo foi gerado como resultado da invocação de `printf`. Observando o comando abaixo isso fica claro:

`gcc -S hello-world.c -o -`

Também pode ser visto compilando normalmente e desmontando o binário:

```
gcc hello-world.c; objdump -S a.out | grep puts
```

A resposta deve mostrar algo com `puts@GLIBC_versão`.

Leituras Sugeridas

- Artigo *Symbol Table in Compiler*, de Harshita Singh
- Artigo *A Study on Verification and Analysis of Symbol Tables for Development of the C++ Compiler*, de YangSun Lee e YunSik Son

Conclusão

A tabela de símbolos é um componente essencial em todas as fases de um compilador, funcionando como um dicionário que associa chaves a registros. Ela armazena dados sobre os identificadores usados no programa, como tipo, precisão, escopo e localização na memória. A escolha da estrutura de dados para a tabela de símbolos, seja uma tabela *hash*, lista encadeada ou árvore de busca binária, impacta diretamente na eficiência do compilador.

A tabela de símbolos deve ser capaz de realizar operações básicas de busca e inserção de maneira eficiente. Exemplos práticos mostram como diferentes implementações lidam com questões como escopo e hierarquia.

Exercícios

1) Há dois tipos de atividades básicas em um compilador: análise e síntese. A tabela de símbolos é criada em qual dessas atividades?
2) Analise a seguinte afirmação: *A tabela de símbolos é uma estrutura utilizada para armazenar principalmente os nomes de funções, variáveis, classes etc. Entre os atributos presentes na tabela, normalmente estão a tipagem, o escopo, os argumentos, os tipos de retornos, etc.* Verdadeiro ou falso?
3) Analise a seguinte afirmação: *Os tipos de dados dos identificadores são adicionados na*

tabela de símbolos ainda na fase de análise léxica. Verdadeiro ou falso?

4) Para cada aplicação listada abaixo, à esquerda, se for necessário armazenar os dados em uma variável que é a parametrização de uma tabela de símbolos genérica (TS), encontre a parametrização no grupo da direita que fará o trabalho corretamente. Assuma que tipos com colchetes são listas daquele tipo.

1. Para cada ano de calendário desde 1989, o número de estudantes que se graduou no IFC naquele ano.
2. Um dicionário de todos os sinônimos de todas as palavras em português.
3. Contagem de frequência de palavras em um livro.
4. Armazenar uma lista de pessoas vivendo em cada endereço e rastrear a idade de cada pessoa lá.
5. Rastrear de que jeito cada senador eleito votou (sim ou não) em alguma moção.
6. Tabela de símbolos de todos os nomes e valores de variáveis locais em uma máquina virtual.

a. TS<Literal, Inteiro>
b. TS<Literal, Lógico>
c. TS<Inteiro, Inteiro>
d. TS<Literal, Literal[]>
e. TS<Literal, TS<Literal, Inteiro>>
f. TS<Literal, Objeto>

Respostas

1) Análise.
2) Verdadeiro.
3) Falso.
4) 1c; 2d; 3a; 4e; 5b; 6f (`Objeto` é como se fosse uma superclasse genérica, como ocorre em Java. Ela pode referenciar qualquer tipo não primitivo.

Referências

AHO, Alfred V. et al. Compilers: principles, techniques, & tools. Pearson Education India, 2007.

ALI, Shaukat. Symbol Table. Disponível em: http://www.uop.edu.pk/ocontents/Chapter%2013%20-%20Symbol%20Table.pdf. Acesso em: 22 mai. 2024.

DU BOIS, André Rauber. Notas de Aula sobre Compiladores. 18 ago. 2011.

RICARTE, Ivan Luiz Marques. 2003. Compiladores. https://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node37.html. Acesso em: 22 mai. 2024.