Projeto de Compilador
E4 para **Análise Semântica**

Prof. Lucas Mello Schnorr
schnorr@inf.ufrgs.br

1 Introdução

A quarta etapa do trabalho de implementação de um compilador para a linguagem consiste em verificações semânticas. Elas fazem parte do sistema de tipos da linguagem com um conjunto de regras detalhado a seguir. A verificação de tipos é feita em tempo de compilação. Todos os nós da Árvore Sintática Abstrata (AST), gerada na E3, terão agora um novo campo que indica o seu tipo (se é inteiro, ponto-flutuante, etc). O tipo de um determinado nó da AST pode, em algumas situações, não ser definido diretamente, como para os comandos de fluxo de controle, por exemplo. Na maioria dos casos, no entanto, o tipo de um nó da AST é definido seguindo as regras de inferência da linguagem.

2 Funcionalidades Necessárias

2.1 Implementar uma tabela de símbolos

A tabela de símbolos guarda informações a respeito dos símbolos (identificadores e literais) encontrados na entrada. Cada entrada na tabela de símbolos tem uma chave e um conteúdo. A chave única identifica o símbolo, e o conteúdo deve ter os campos:

- localização (linha e coluna, esta opcional)
- natureza (literal, identificador ou função)
- tipo (qual o tipo do dado deste símbolo)
- dados do valor do token pelo `yylval` (veja E3)

A implementação deve prever que várias tabelas podem coexistir ao mesmo tempo, uma para cada escopo. As regras de escopo são delineadas a no anexo.

2.2 Verificação de declarações

Todos os identificadores devem ter sido declarados no momento do seu uso, seja como variável, seja como função. Todas as entradas na tabela de símbolos devem ter um tipo associado conforme a declaração, verificando-se se não houve dupla declaração ou se o símbolo não foi declarado. Caso o identificador já tenha sido declarado, deve-se lançar o erro `ERR_DECLARED`. Caso o identificador não tenha sido declarado no seu uso, deve-se lançar o erro `ERR_UNDECLARED`. Variáveis com o mesmo nome podem co-existir em escopos diferentes, efetivamente mascarando as variáveis que estão em escopos superiores.

2.3 Uso correto de identificadores

O uso de identificadores deve ser compatível com sua declaração e com seu tipo. Variáveis somente podem ser usadas em expressões e funções apenas devem ser usadas como chamada de função, isto é, seguidas da lista de argumentos possivelmente vazia. Caso o identificador dito variável seja usado como uma função, deve-se lançar o erro `ERR_VARIABLE`. Enfim, caso o identificador dito função seja utilizado como variável, deve-se lançar o erro `ERR_FUNCTION`.

2.4 Verificação de tipos

Inferência de tipos na AST. Uma declaração de variável permite ao compilador definir o seu tipo. O espaço ocupado em memória desta variável, seja ela simples ou arranjo, deve ser calculado baseado no tipo, sendo então registrado na tabela de símbolos. Os tipos de dados corretos devem ser inferidos onde forem usados, em expressões aritméticas, lógicas e relacionais. Para simplificar esse procedimento, os nós da AST devem ser anotados com um tipo definido de acordo com as regras de inferência de tipos. Um nó da AST deve ter portanto um novo campo que registra o seu tipo de dado. O processo de inferência de tipos está descrito abaixo (veja a seção "Sistema de tipos da Linguagem").

2.5 Mensagens de erro

Mensagens de erro significativas devem ser fornecidas. Elas devem descrever em linguagem natural o erro semântico, as linhas envolvidas, os identificadores e a natureza destes de uma maneira que o usuário do seu compilador compreenda o erro semântico.

A Sistema de tipos da Linguagem

A.1 Regras de Escopo

A verificação de declaração de tipos deve considerar o escopo da linguagem. O escopo pode ser global, local da função e local de um bloco, sendo que este pode ser recursivamente aninhado. Uma forma de se implementar estas regras de escopo é através de uma pilha de tabelas de símbolos. Para verificar se uma variável foi declarada, verifica-se primeiramente no escopo atual (topo da pilha) e enquanto não encontrar, deve-se descer na pilha (sem desempilhar) até chegar no escopo global (base da pilha, sempre presente). Caso o identificador não seja encontrado após este procedimento, temos a evidência que ele não foi declarado e portanto emitimos um erro semântico. Para a declaração de um símbolo, basta inseri-lo na tabela de símbolos do escopo que encontra-se no topo da pilha. O grupo deve identificar os locais adequados para inserir a criação, empilhamento, desempilhamento e destruição de uma tabela de símbolos. Não há necessidade de manter as tabelas em memória, na E4, até o final do processo de compilação.

A.2 Conversão implícita

As regras de coerção de tipos da inguagem são as seguintes. Um tipo `int` pode ser convertido implicitamente para `float` e para `bool`. Um tipo `bool` pode ser convertido implicitamente para `float` e para `int`. Um tipo `float` pode ser convertido implicitamente para `int` e para `bool`, perdendo precisão. Concretamente, isso implica que qualquer combinação de variáveis desses tipos em operações aritméticas, lógicas e relacionais é válida.

A.3 Inferência

As regras de inferência de tipos da linguagem são as seguintes. A partir de `int` e `int`, infere-se `int`. A partir de `float` e `float`, infere-se `float`. A partir de `bool` e `bool`, infere-se `bool`. A partir de `float` e `int`, infere-se `float`. A partir de `bool` e `int`, infere-se `int`. A partir de `bool` e `float`, infere-se `float`. Nos comandos de fluxo de controle, o tipo de dado da atribuição e do comando de inicialização de variável local é determinado pelo tipo de quem recebe o valor atribuído; o tipo de dado de um `if` (com else opcional) e de um `while` é o tipo de dado da expressão de teste. O tipo de dado de uma chamada de função é o tipo da função sendo chamada. O tipo de dado de um comando de retorno é o tipo do valor de retorno.

B Códigos de retorno

Os seguintes códigos de retorno devem ser utilizados quando o compilador encontrar erros semânticos. O programa deve chamar `exit` utilizando esses códigos imediamente após a impressão da linha que descreve o erro. Na ausência de qualquer erro, o programa deve retornar o valor zero.

```
#define ERR_UNDECLARED       10 //2.2
#define ERR_DECLARED         11 //2.2
#define ERR_VARIABLE         20 //2.3
#define ERR_FUNCTION         21 //2.3
```

Estes valores são utilizados na avaliação objetiva.

C Arquivo `main.c`

Utilize o mesmo `main.c` da E3.
Cuide da alocação dinâmica das tabelas.