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. Estas verificações fazem parte do sistema de tipos da linguagem com um conjunto de regras detalhado a seguir. Toda 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 (para os comandos de fluxo de controle, por exemplo). Na maioria dos casos, no entanto, seu tipo 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, variável, função, etc)
- tipo (qual o tipo de dado deste símbolo)
- tamanho (derivado do tipo e se vetor)
- argumentos e seus tipos (no caso de funções)
- dados do valor do token pelo `yylval` (veja E3)

A implementação deve prever que várias tabelas de símbolos possam coexistir, 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, como vetor ou 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. As regras de escopo são delineadas no anexo.

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 sem indexação, vetores somente podem ser utilizados com indexação, e funções apenas devem ser usadas como chamada de função, isto é, seguidas da lista de argumentos, esta possivelmente vazia conforme a sintaxe da E2, entre parênteses. Caso o identificador dito variável seja usado como vetor ou como função, deve-se lançar o erro `ERR_VARIABLE`. Caso o identificador dito vetor seja usado como variável ou função, deve-se lançar o erro `ERR_VECTOR`. Enfim, caso o identificador dito função seja utilizado como variável ou vetor, deve-se lançar o erro `ERR_FUNCTION`.

2.4 Verificação de tipos e tamanho dos dados

Uma declaração de variável deve permitir ao compilador definir o tipo e a ocupação em memória da variável na sua entrada na tabela de símbolos. Com o auxílio desta informação, quando necessário, os tipos de dados corretos devem ser inferidos onde forem usados, em expressões aritméticas, relacionais, lógicas, ou para índices de vetores. Por simplificação, isso nos leva a situação onde todos 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. Como não temos coerção de variáveis do tipo string e char, o compilador deve lançar o erro `ERR_STRING_TO_X` quando a variável do tipo string estiver em uma situação onde ela deve ser convertida para qualquer outro tipo. De maneira análoga, o erro `ERR_CHAR_TO_X` deve ser lançado quando uma variável do tipo char deve ser convertida implicitamente. Enfim, vetores não podem ser do tipo `string`. Caso um vetor tenha sido declarado com o tipo `string`, o erro `ERR_STRING_VECTOR` deve ser lançado.

2.5 Retorno, argumentos e parâmetros de funções

A lista de argumentos fornecidos em uma chamada de função deve ser verificada contra a lista de parâmetros formais na declaração da mesma função. Cada chamada de função deve prover um argumento para cada parâmetro, e ter o seu tipo compatível. Tais verificações devem ser realizadas levando-se em conta as informações registradas na tabela de símbolos, registradas no momento da declaração/definição da função. Na hora da chamada da função, caso houver um número menor de argumentos que o necessário, deve-se lançar o erro `ERR_MISSING_ARGS`. Caso houver um número maior de argumentos que o necessário, deve-se lançar o erro `ERR_EXCESS_ARGS`. Enfim, quando o número de argumentos é correto, mas os tipos dos argumentos são incompatíveis com

os tipos registrados na tabela de símbolo, deve-se lançar o erro ERR_WRONG_TYPE_ARGS. Retorno, argumentos e parâmetros de funções não podem ser do tipo string. Quando estes casos acontecerem, lançar o erro ERR_FUNCTION_STRING.

2.6 Verificação de tipos em comandos

Prevalece o tipo do identificador que recebe um valor em um comando de atribuição. O erro ERR_WRONG_TYPE deve ser lançado quando o tipo do valor a ser atribuído a um identificador for incompatível com o tipo deste identificador. Os demais comandos simples da linguagem devem ser verificados semanticamente para obedecer as seguintes regras. O comando input deve ser seguido obrigatoriamente por um identificador do tipo int e float. Caso contrário, o compilador deve lançar o erro ERR_WRONG_PAR_INPUT. De maneira análoga, o comando output deve ser seguido por um identificador ou literal do tipo int e float. Caso contrário, deve ser lançado o erro ERR_WRONG_PAR_OUTPUT. O comando de retorno return deve ser seguido obrigatoriamente por uma expressão cujo tipo é compatível com o tipo de retorno da função. Caso não seja o caso, o erro ERR_WRONG_PAR_RETURN deve ser lançado pelo compilador. Nos comandos de shift (esquerda e direta), deve-se lançar o erro ERR_WRONG_PAR_SHIFT caso o parâmetro após o token de shift for um número maior que 16.

2.7 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.

A Sistema de tipos da Linguagem

Regras de Escopo. A verificação de declaração prévia 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 até chegar no escopo global (base da pilha, sempre presente). Caso o identificador não seja encontrado, isso indica que ele não foi declarado. Para se "declarar"um símbolo, basta inserí-lo na tabela de símbolos do escopo que encontra-se no topo da pilha.

Conversão implícita. As regras de coerção de tipos da Linguagem são as seguintes. Não há conversão implícita para os tipos string e char. 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.

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. A matriz abaixo resume:

Tamanho. O tamanho dos tipos da linguagem é definido da seguinte forma. Um char ocupa 1 byte. Um string ocupa 1 byte para cada caractere. O tamanho máximo de um string é definido na sua inicialização (com o operador de inicialização). Uma string não inicializada ocupa 0 bytes e não pode receber valores cujo tamanho excede àquele máximo da inicialização. Caso o tamanho de um string a ser atribuído exceder o máximo, deve-se emitir o erro ERR_STRING_MAX. Um int ocupa 4 bytes. Um float ocupa 8 bytes. Um bool ocupa 1 byte. Um vetor ocupa o seu tamanho vezes o seu tipo.

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
#define ERR_DECLARED         11
#define ERR_VARIABLE         20
#define ERR_VECTOR           21
#define ERR_FUNCTION         22
#define ERR_WRONG_TYPE       30
#define ERR_STRING_TO_X      31
#define ERR_CHAR_TO_X        32
#define ERR_STRING_MAX       33
#define ERR_STRING_VECTOR    34
#define ERR_MISSING_ARGS     40
#define ERR_EXCESS_ARGS      41
#define ERR_WRONG_TYPE_ARGS  42
#define ERR_FUNCTION_STRING  43
#define ERR_WRONG_PAR_INPUT  50
#define ERR_WRONG_PAR_OUTPUT 51
#define ERR_WRONG_PAR_RETURN 52
#define ERR_WRONG_PAR_SHIFT  53
```

Estes valores são utilizados na avaliação objetiva.

C Arquivo `main.c`

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