

Trabalho Prático - Especificação da Etapa 2: Análise Sintática e Preenchimento da Tabela de Símbolos

Resumo:

O trabalho consiste na implementação de um compilador para a linguagem que chamaremos a partir de agora de **abcde**. Na segunda etapa do trabalho é preciso fazer um analisador sintático utilizando a ferramenta de geração de reconhecedores *yacc* (ou *bison*) e completar o preenchimento da tabela de símbolos encontrados, associando os valores e tipos corretos aos *tokens*.

Funcionalidades necessárias:

A sua análise sintática deve fazer as seguintes tarefas:

- o programa principal deve chamar uma única vez a rotina *yyparse* para reconhecer programas que fazem parte da linguagem, e se concluída com sucesso, a análise deve retornar o valor 0 com *exit(0)*;
- imprimir uma mensagem de erro sintático para os programas não reconhecidos, informando a linha onde o erro ocorreu, e retornar o valor 3 como código genérico de erro sintático, chamando *exit(3)*;
- os nodos armazenados na tabela hash devem distinguir entre os tipos de símbolos armazenados, os valores devem ser convertidos do lexema para um campo de tipo apropriado, e o nodo deve ser associado ao token retornado através da atribuição para *yylval.symbol*;

Descrição Geral da Linguagem

Um programa na linguagem **abcde** é composto por um conjunto de declarações de variáveis globais e um conjunto de funções, que podem aparecer intercaladamente e em qualquer ordem. Não há variáveis locais ou tipos definidos pelo usuário. Cada função é descrita por um cabeçalho seguido de seu corpo. Os comandos podem ser de atribuição, controle de fluxo ou os comandos *read*, *print* e *return*.

Declarações de variáveis globais

As variáveis são declaradas escrevendo seu tipo e seu nome, e cada declaração é **terminada** por ponto-e-vírgula. A linguagem inclui também a declaração de vetores, feita pela definição de seu tamanho inteiro positivo entre colchetes, colocada à direita do nome do vetor. Todas as variáveis globais serão inicializadas com zero. Variáveis podem ser dos tipos `char` e `int`.

Definição de funções

Cada função é definida por um cabeçalho e um corpo. O cabeçalho consiste no tipo do valor de retorno seguido do nome da função e de uma lista, possivelmente vazia, entre parênteses, de parâmetros de entrada, separados por vírgula, onde cada parâmetro é definido pelo tipo seguido do nome, e não podem ser do tipo vetor. O corpo da função, que segue o cabeçalho, é composto somente por um comando, como definido a seguir.

Bloco de Comandos

Um bloco de comandos é definido entre chaves, e consiste em uma sequência de comandos simples, **separados** por ponto-e-vírgula. Um bloco de comandos é considerado como um comando único simples, recursivamente, e pode ser utilizado em qualquer construção que aceite um comando simples.

Comandos simples

Os comandos da linguagem podem ser: atribuição, construções de controle de fluxo, *read*, *print*, *return* e comando vazio. Na atribuição usa-se uma das seguintes formas:

```
variável = expressão  
vetor[ expressão ] = expressão
```

Os tipos corretos para o assinalamento e para o índice serão verificados somente na análise semântica. O comando *read* é identificado pela palavra reservada *read*, seguida de um nome de variável, na qual o valor lido da entrada padrão, se disponível e compatível, será colocado. O comando *print* é identificado pela palavra reservada *print*, seguida ou de um *string* ou de uma expressão aritmética a ser impressa. O comando *return* é identificado pela palavra reservada *return* seguida de uma expressão aritmética que dá o valor de retorno. Os comandos de controle de fluxo são descritos a seguir. Para facilitar a escrita de programas aceitando o caractere de ponto-e-vírgula como terminador, e não apenas separador, a linguagem deve aceitar também o comando vazio.

Comandos de Controle de Fluxo

Para controle de fluxo, a linguagem possui as seguintes construções:

```
if (expressão) then comando  
if (expressão) then comando else comando  
while (expressão) comando
```

Expressões Aritméticas ou Lógicas

As expressões aritméticas têm como folhas identificadores, opcionalmente seguidos de expressão inteira entre colchetes, para acesso a vetores, ou podem ser literais numéricos e *ascii*. As expressões aritméticas podem ser formadas recursivamente com operadores aritméticos, assim como permitem o uso de parênteses para associatividade. Expressões Lógicas (Booleanas) podem ser formadas através dos operadores relacionais aplicados a expressões aritméticas, ou de operadores lógicos aplicados a expressões

lógicas, recursivamente. Nesta etapa do trabalho, porém, não haverá distinção alguma entre expressões aritméticas, inteiras ou lógicas. A descrição sintática deve aceitar qualquer operador e sub-expressão de um desses tipos como válidos, deixando para a análise semântica das próximas etapas verificar a validade dos operandos e operadores. Finalmente, um operando possível de expressão é uma chamada de função, feita pelo seu nome, seguido de argumentos entre parênteses.

Tipos e Valores na tabela de Símbolos

A tabela de símbolos até aqui poderia representar o tipo do símbolo usando os mesmos **#defines** criados para os *tokens* (agora gerados pelo *yacc*). Mas logo será necessário fazer mais distinções, principalmente pelo tipo dos identificadores. Assim, é preferível criar um código especial para símbolos, através da série de definições abaixo:

```
#define      SYMBOL_UNDEFINED      0
#define      SYMBOL_LIT_INTEGER    1
#define      SYMBOL_LIT_TRUE       3
#define      SYMBOL_LIT_FALSE      4
#define      SYMBOL_LIT_CHAR        5
#define      SYMBOL_LIT_STRING      6
#define      SYMBOL_IDENTIFIER      7
```

Controle e organização do seu código fonte

O arquivo `tokens.h` usado na etapa1 não é mais necessário. Você deve seguir as demais regras especificadas na etapa1, entretanto. A função *main* escrita por você será usada sem alterações para os testes da etapa2. Você deve utilizar um *Makefile* para que seu programa seja completamente compilado com o comando *make*. Utilize o exemplo disponibilizado na página como referência. O formato de entrega será o mesmo da etapa1, e todas as regras devem ser observadas, apenas alterando o nome do arquivo executável e do arquivo `.tgz` para “etapa2”.

Atualizações e Dicas

Verifique regularmente a página da disciplina e o final desse documento para informar-se de alguma eventual atualização que se faça necessária ou dicas sobre estratégias que o ajudem a resolver problemas particulares. Em caso de dúvida, consulte o professor.

Porto Alegre, 14 de Setembro de 2011