**Gramática Personalizada C++**

A forma como a gramática funciona é que ela divide um programa em categorias sintáticas , que descrevem o que compõe as diferentes partes de um programa. Podemos usar a mesma ideia para descrever uma gramática para o inglês: por exemplo, uma *frase* é uma categoria sintática composta de outras categorias (menores), como *substantivos* , *verbos* e *adjetivos* . A gramática descreve o que há nessas categorias e como construir as estruturas maiores a partir das menores.

Outra maneira de usar a gramática é "diagramar" um programa - exatamente como [diagramar uma frase](http://en.wikipedia.org/wiki/Sentence_diagram) na escola primária. Você deve ser capaz de circular cada parte de seu programa e identificar a categoria sintática - é uma declaração? uma expressão? uma declaração? uma definição? - e explique por que ele pertence onde está no programa.

A notação para regras gramaticais usa vários operadores especiais (coloridos em azul):

* : = define uma regra e pode ser lido como "é composto por".
* | (barra vertical) separa as opções em uma regra.
* [] (colchetes) incluem partes de uma regra que são opcionais.
* {} (chaves) envolvem partes de uma regra que podem ser repetidas zero ou mais vezes.

**Programa**

A regra gramatical mais importante define o que compõe um programa, no nível mais alto. Nossos programas consistem em a #include, seguido por zero ou mais definições de estrutura, zero ou mais declarações de função e zero ou mais definições de função. Aqui está como escrevemos isso na gramática:

|  |  |  |
| --- | --- | --- |
| **programa** | **: =** | **#include "comp11io.h" { structdef } { functiondecl } { functiondef }** |

**Definições de estrutura**

Novo: as definições de struct introduzem novos tipos, então geralmente os colocamos no topo do arquivo de origem C ++. O "corpo" da estrutura consiste em um conjunto de campos ou membros, cada um dos quais possui um tipo e um nome. O nome que segue a structpalavra - chave será o nome do novo tipo.

|  |  |  |
| --- | --- | --- |
| **structudef** | **: =** | **struct nome {     { tipo de nome ; } }** |

**Declarações e definições de funções**

Ambas as declarações e definições de função têm um cabeçalho, então podemos fazer uma regra separada para isso e usá-la duas vezes. Observe a regra dos parâmetros: diz *"os parâmetros são compostos de um parâmetro, seguido por zero ou mais ocorrências de uma vírgula seguida por outro parâmetro."*(Um nome é qualquer palavra que comece com uma letra seguida por um certo número de letras e dígitos.)

|  |  |  |
| --- | --- | --- |
| **cabeça** | **: =** | **nome do tipo ( [ parâmetros ] )** |
| **parametros** | **: =** | **parâmetro { , parâmetro }** |
| **parâmetro** | **: =** | **Digite o nome** |
| **modelo** | **: =** | **int | double | bool | char | string | nome**  **// Observação: deve ser um nome de tipo de estrutura** |

Novo: Observe que, uma vez que podemos definir novos tipos usando structdef , a regra para tipos agora inclui a opção para um nome , que teria que ser o nome de um tipo de struct.

**Corpo da função**

O corpo consiste em uma mistura de definições e declarações de variáveis. Observe a maneira como defini essa regra: você pode escolher um variabldef ou uma instrução , e *tudo isso* tem uma "estrela", permitindo que você misture os dois tipos de construção.

|  |  |  |
| --- | --- | --- |
| corpo | : = | { variabledef   | declaração } |

Compare essa regra com uma regra ***alternativa*** que força você a ter todas as suas declarações de variáveis ​​primeiro e, em seguida, todas as suas declarações (é assim que as linguagens de programação C mais antigas funcionam):

|  |  |  |
| --- | --- | --- |
| corpo | : = | { variabledef }          // Observação: força todas as definições de variáveis ​​a virem primeiro { instrução } |

Novo: na última aula, dividimos uma definição de variável em duas coisas: uma declaração de variável (com uma definição opcional) e uma atribuição de variável. Deixaremos a regra de definição de variável existente, mas tornaremos a definição opcional. Para atribuição de variável, adicionaremos um novo tipo de instrução a seguir.

Uma definição de variável associa o valor de uma expressão a uma variável. Por enquanto, deixaremos a expressão indefinida - informalmente, é uma fórmula matemática de algum tipo. Pode ser um simples número ou variável, ou um grande e complexo cálculo. As expressões também incluem aplicativo de função (ou *chamada de* função ).

|  |  |  |
| --- | --- | --- |
| variableef | : = | nome do tipo [ = expressão ] ; |

**Afirmações**

Os tipos de instruções que vimos incluem a instrução return e a instrução if / else. Usarei "corpo" dentro da instrução if, para indicar que qualquer série de instruções é permitida lá. Observe os colchetes ao redor da parte "else", indicando que é opcional.

|  |  |  |
| --- | --- | --- |
| demonstração | : = | | returnstmt | ifstmt | tarefa |
| returnstmt | : = | return expressão; |
| ifstmt | : = | if () {     corpo de expressão } [ else {     corpo } ] |
| tarefa | : = | expressão do nome= ; |