Universidade de Évora

Compiladores

1ª Frequência - Correcção

(1) 1. Considere o seguinte conjunto de números de vírgula flutuante:

`{1.0, .365, 0.234E25, 0.123e--10, .23e+15}`

Qual das seguintes expressões regulares identifica números (literais) do tipo dos que pertencem ao conjunto? (considere a notação do `flex`)

A. `[0-9]\.[0-9]*[Ee]?[0-9]+`

B. `[0-9]?\.[0-9]+([Ee]?[0-9]+)*`

C. `[0-9]*\.[0-9]+([Ee][+-]?[0-9]+)?`

D. `[0-9]+(\.)?[0-9]*[Ee+-]?[0-9]?`

2. Usando a questão anterior como exemplo, explique como se processam, em termos de compilação, os casos em que:

(1) (a) o número é negativo (e.g., `{-1.0, -.365, -0.234E25, -0.123e-10, -.23e+15}`).

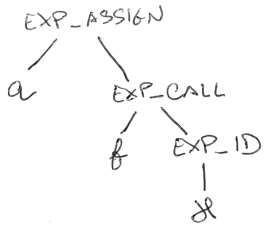
> **Solução:** Os números são *tokens* sem sinal. O sinal de menos é um token. Na gramática há uma regra para o menos (unário), que determina que um sinal de menos antes de algo é uma operação unária. Resumindo, os números negativos são tratados como operações unárias cujo operador é `-`.
>
> (Nota: geralmente, é feito o mesmo para o sinal de `+`, mas este pode ser ignorado quando se gera a APT).

(0.75) (b) aparecem vários sinais (e.g., `3.5 - -2.1`).

> **Solução:** Neste caso temos uma operação unária de `-` aplicada ao segundo número e uma operação binária de `-` entre dois argumentos. Resumindo, temos uma subtracção (normal) entre uma expressão que é um literal float e outra expressão que é uma operação unária de `-` aplicada a outra expressão que é outro literal float.

3. Para as seguintes linhas de código, proponha uma representação para a sintaxe abstracta e desenhe as APTs respectivas:

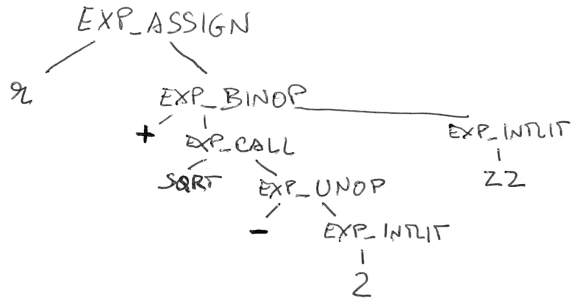
(1.5) (a) `a = f(x);`

> **Solução:** Para este exemplo, precisamos de 3 tipos de nós na sintaxe abstracta:
>
> - `exp_assign(ID, EXP)`
> - `exp_call(ID, ARG)` (simplificamos e assumimos que só há 1 argumento)
> - `exp_id(ID)`
>
> 
>

(1.5) (b) `r = sqrt(-2) + 22;`

Solução: Acrescentamos alguns nós à sintaxe abstracta:

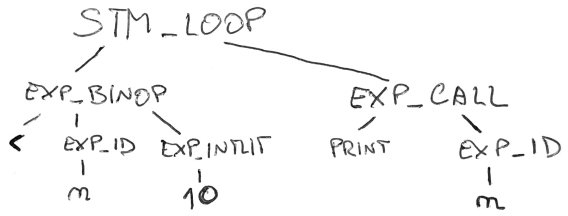
- `exp_binop(OP, EXP, EXP)`
- `exp_unop(OP, EXP)`
- `exp_intlit(NUM)`



(1.5) (c) `while (n<10) do { print(n); };`

Solução: Para este excerto, já temos quase tudo, apenas nos falta um nó para o ciclo:

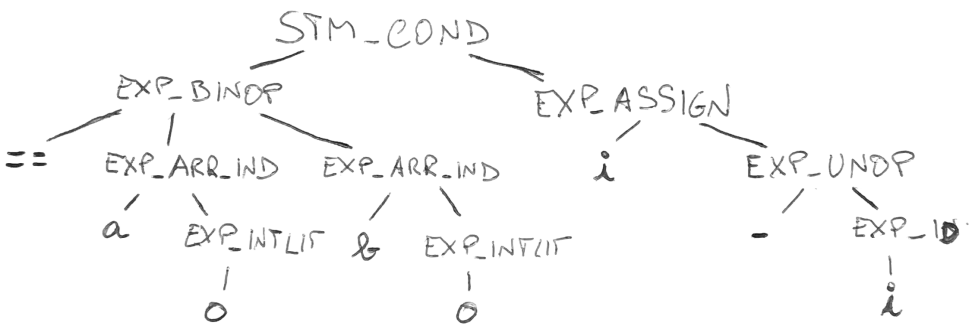
- `stm_loop(COND, BODY)` (vamos assumir que o body tem só uma instrução)



(1.5) (d) `if a[0] == b[0] then { i = -i };`

Solução: Agora temos um condicional (igual ao ciclo, para este exemplo até podemos ignorar o *iffalse branch*), mas a maior novidade é o acesso a índices de array:

- `stm_cond(COND, IFTRUE)`
- `exp_arr_ind(ID, EXP)`



4. Considere o processo de compilação de um determinado programa. Na fase da análise semântica, obteve-se a seguinte evolução da *Symbol Table*:

1.

id	type	args
f	float	int, int
a	int(arg)	
b	int(arg)	
i	int	
r	float	

ENV

2.

id	type	args
f	float	int, int
g	int	float
a	float(arg)	
i	int	

ENV

3.

id	type	args
f	float	int, int
g	int	float
o	int	
main	void	void
t	string	

ENV

(1) (a) A *Symbol Table* apresentada ainda sofrerá alterações até ao fim da análise semântica? Justifique.

Solução: Sofrerá pelo menos uma alteração: ao chegar ao fim da sub-árvore da função `main()`, será chamada a função `drop_environment()`, que remove o scope de nível 2 da Symbol Table (ou seja, a variável `t` vai desaparecer).

Por outro lado, nada sabemos sobre o código do programa nem sobre a linguagem, consoante os quais ainda podemos ter:

- mais variáveis na função `main()`
- mais variáveis globais depois da função `main()`
- mais funções declaradas após a função `main()`

(1.5) (b) Proponha um excerto de programa que possa ter dado origem à *Symbol Table* apresentada.

Solução: Basta um simples exemplo, mesmo sem código "executável":

```
f(a:int, b:int) : float {
  i: int;
  r: float;
}
g(a:float) : int {
  i: int;
}
o: int;
main() : void {
  t: string;
}

```

Note-se que a variável `o` é global, pois não aparece dentro de nenhum contexto local na Symbol Table.

5. Considere o seguinte programa em *Ya!*:

```
1  factrec (n: int) : int {
2    a,b: float;
3
4    if n == 1 then { b = 1; }
5    else {
6      a = n - 1;
7      b = n * factrec(a);
8    };
9    return b;
10 };
11
12 main () : void {
13   a: int[10];
14   print(factrec(3));
15 };
```

(1.5) (a) Suponha que se introduz uma nova linha de código entre as linhas 7 e 8: `a[1] = 0`. Que tipo de erro estamos a introduzir? Justifique em 10 palavras ou menos.

Solução: Estamos a introduzir um erro semântico, pois apesar de a sintaxe estar correcta, a variável `a` não é de um tipo indexável.

(1.5) (b) Mostre uma representação da *Symbol Table*, aquando da análise semântica do ramo da APT correspondente ao código da linha 8.

Solução:

id	type	args
factrec	int	int
n	int(arg)	
a	float	
b	float	

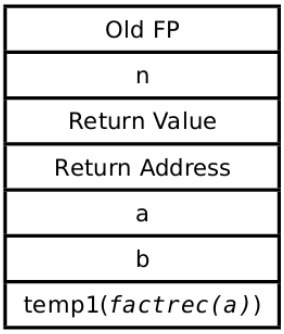
ENV

(2) (c) Explique, de forma sucinta, como é feita a distinção entre a variável `a` da função `factrec` e a variável `a` da função `main`, durante o processo de compilação.

Solução: Na análise semântica, as variáveis ficam em contextos diferentes (nunca coexistem na Symbol Table. Durante a execução, cada uma está num registo de activação diferente.

(1.25) (d) Proponha um desenho para o *Registo de Activação* da função `factrec()`.

Solução: Uma hipótese, assumindo que o valor da chamada recursiva necessita de um temporário:



(1.25) (e) Considerando o desenho proposto por si na alínea anterior, dê uma estimativa do espaço máximo ocupado na *stack* pela função `factrec()`, assumindo a execução do código da linha `14`.

Solução: Aqui basta uma estimativa. Assumindo que a arquitectura de destino é de 32 bits, o RA apresentado (7 words) ocuparia $7 \times 4 = 28$ bytes. Como a chamada vai ser recursiva 2 vezes, o tamanho máximo seriam 84 bytes.

Se não considerarmos a chamada com o argumento 3, então a stack poderá crescer infinitamente (característica das funções recursivas).

(1.25) (f) Suponha agora que alguém implementou um compilador para a linguagem *Ya!*, do qual não sabemos nada sobre a sua implementação. Ao correr esse compilador sobre o código anterior, obtém-se a seguinte mensagem de erro:

```
Erro de tipos na linha 7.
```

Resumidamente, e assumindo que o compilador está bem implementado (com excepção das mensagens de erro não estarem detalhadas), dê uma ideia sobre o que se passa para o erro ter ocorrido.

Solução: Claramente, este compilador não faz a conversão implícita entre `int` e `float`... O erro tanto pode ser por estarmos a passar um argumento `float` à função `factrec()`, como por estarmos a afectar o resultado de `factrec()` (`int`) à variável `b` (`float`).