COMPILADORES

Prova 1 – 07/03/2017

Prof. Marcus Ramos

1. (1,0 ponto) O estudo dos processadores de linguagens é justificado pela importância das linguagens de programação de alto-nível no desenvolvimento de programas. Que importância é essa?

Linguagens de programação de alto-nível são fundamentais para o desenvolvimento de programas. Elas garantem legibilidade, produtividade, facilidade de manutenção, segurança e portabilidade, entre suas principais vantagens quando comparadas com as linguagens de baixo-nível. Linguagens de baixo-nível são utilizadas atualmente apenas em situações muito especiais, e por isso temos uma dependência muito grande de bons processadores de linguagens (compiladores, interpretadores etc) em diversas áreas de aplicação.

2. (1,0 ponto) Qual a relação que existe entre programa-fonte/programa-objeto e programa de alto-nível/programa de baixo-nível?

Programa-fonte é aquele que é fornecido como entrada para um processador de linguagem. Programa-objeto é aquele que é gerado como saída. O programa-fonte pode ser tanto de alto quanto de baixo-nível. O mesmo vale para o programa-objeto. Programa de alto-nível é aquele que é escrito em alguma linguagem de alto-nível. Programa de baixo-nível é aquele que é escrito em alguma linguagem de baixo-nível (como linguagem de montagem ou linguagem de máquina).

3. (1,0 ponto) Desceva, usando a notação dos Diagramas-T, os passos necessários para a obtenção de uma versão executável de um compilador auto-compilável Pascal/x86. Considere que você dispõe de um compilador C/x86.

a. Escrever um compilador Pascal/x86 em C;
b. Compilar (a) em C/x86 e obter Pascal/x86;
c. Traduzir manualmente (a) para Pascal;
d. Compilar (c) em (b).

4. (1,0 ponto) Defina o que é um passo e fase de compilação. Dê exemplos.

Um passo de compilação corresponde à uma leitura completa do programa-fonte na sua versão original ou em alguma representação (como por exemplo uma árvore de sintaxe). Um compilador pode ser constituído em um único passo ou em múltiplos passos. Uma fase de compilação é uma tarefa bem determinada que é executada pelo compilador. É o caso, por exemplo, da análise sintática, da análise de contexto e da geração de código.

5. (1,0 ponto) Porque hoje em dia é mais vantajoso organizar um compilador em vários passos ao invés de um único passo?

A compilação em múltiplos passos traz como principal vantagem uma melhor organização do código do compilador do ponto de vista da Engenharia de Software. Os princípios da alta coesão e do baixo acoplamento entre os módulos são melhor obedecidos, o que favorece a construção, o teste e a manutenção do compilador. As desvantagens, relacionadas com o

maior uso de memória e maior necessidade processamento, não são consideradas relavantes face à atual tecnologia.

6. (1,0 ponto) Em que consiste o front-end e o back-end de um compilador? Qual a vantagem de organizá-lo desta forma?

O front-end é composto pelas fases de análise sintática e de contexto, e compreende tudo o que se refere às características da linguagem-fonte. O back-end, por sua vez, compreende o gerador de código e fase relacionadas (otimização de código dependente de máquina, por exemplo). Um compilador organizado desta forma permite a combinação de diversos front-end com diversos back-end (desde que a interface seja compatível), garantindo uma multiplicidade de processadores distintos a partir de alguns poucos front-end e back-end.

7. (1,0 ponto) Verifique se a gramática a seguir é LL(1). Justifique a sua resposta. Em caso negativo, obtenha uma gramática equivalente que seja LL(1).

   S → aaXaa | abYab
   X → bX | ε
   Y → bY | ε

Não é LL(1) pois first1 (aaXaa)= first1 (abYab)={a}. Para obter uma gramática equivalente LL(1) é necessário fazer a seguinte fatoração:

S → a(aXaa|bYab)
X → bX | ε
Y → bY | ε

Desta maneira, temos:

- $first_1(aXaa)=\{a\}$, $first_1(bYab)=\{b\}$ e $\{a\}\cap\{b\}=\{\}$;
- $first_1(bX)=\{b\}$, $follow_1(X)=\{a\}$ e $\{b\}\cap\{a\}=\{\}$;
- $first_1(bY)=\{b\}$, $follow_1(Y)=\{a\}$ e $\{b\}\cap\{a\}=\{\}$.

e isso completa a prova de que a gramática manipulada é LL(1).

8. (1,0 ponto) A gramática abaixo representa árvores binárias em forma textual. Obtenha o esboço de um analisador sintático descendente (construído pelo método recursivo descendente) para a correspondente linguagem.

   T → b T T
   T → u T
   T → x
   T → y
   T → (T)

A gramática é LL(1), portanto a codificação pode ser feita diretamente.

```
private void parseT() {
switch currentToken {
        case “b”:
                acceptIt();
                parseT();
                parseT();
```

```
            break;
        case “u”:
            acceptIt();
            parseT();
            break;
        case “x”:
        case “y”:
            acceptIt();
            break;
        case “(“:
            acceptIt();
            parseT();
            accept(“)”);
            break;
        }
}
```

9. (1,0 ponto) Descreva o que acontece durante a subfase de identificação da análise de contexto.

Os identificadores usados no programa são vinculados com as respectivas declarações, conforme as regras de escopo da linguagem. A árvore de sintaxe abstrata é decorada com referências que conectam cada uso de um identificador com a respectiva declaração. Identificadores não declarados ou duplamente declarados no mesmo bloco geram mensagens de erro. Identificadores declarados porém não referenciados geram mensagens de alerta.

10. (1,0 ponto) Descreva o que acontece durante a subfase de verificação de tipos da análise de contexto.

É realizada inicialmente uma inferência dos tipos de expressões usadas no programa. Tais inferências são feitas percorrendo-se às árvores que representam as mesmas de baixo para cima, e verificando se as sub-expressões internas possuem tipos compatíveis com as operações empregadas. Depois, os tipos das expressões são confrontados com os tipos esperados nos pontos onde são usadas. É o caso, por exemplo, de expressões usadas em comandos condicionais, comandos iterativos, comandos de atribuição, passagem de parâmetros etc. Durante esta análise podem ser emitidos mensagens reportando erros de tipo, como é o caso de tipos incompatíveis ou tipos mal-formados.