COMPILADORES

Prova 1 - 26/09/2011 - Prof. Marcus Ramos

1ª Questão (1 ponto) - O que são métodos formais e informais, quando se considera a especificação de linguagens de programação? Descreva em que situações, durante a especificação de uma linguagem de programação de alto-nível, são usados métodos formais e informais. Discorra sobre as conseqüências dessa estratégia para a organização e a implementação do compilador.

Métodos informais correspondem ao uso da linguagem natural (português, inflês etc) para representar alguma dimensão de uma linguagem de programação - sintaxe ou semântica. Método formal corresponde ao uso de alguma notação matemática para a mesma finalidade (gramáticas, autômatos etc). Normalmente, a sintaxe livre de contexto é representada formalmente e a sintaxe dependente de contexto e a semântica de maneira informal. Como conseqüência, a fase de análise sintática precisa ser complementada por uma fase de análise de contexto, sendo que essa deve ser construída manualmente, a partir de especificações informais. O mesmo vale para a fase de geração de código.

2ª Questão (1 ponto) - Considere o trecho de programa C abaixo.

```
#include <stdio.h>
int main () {
      int fat=1,i,n;
      scanf ("%d",&n);
      for (i=1;i<=n;i++) fat=fat*i;
      printf ("%d",fat);
      }
```

Descreva a atividade de um compilador típico, nas fases de análise léxica e sintática (livre e dependente de contexto), usando o programa acima como exemplo. Seja o mais detalhado e preciso possível.

Análise léxica:
Os tokens (identificadores, palavras-chave, operadores, sinais de pontuação etc) são reconhecidos e classificados. Os separadores (espaços em branco, mudanças de linha etc) são reconhecidos e ignorados. Exemplos de tokens no programa acima: int, main, fat, =, *, } etc.

Análise sintática livre de contexto:
A estrutura do programa, incluindo declarações, comandos e expressões, é confrontada com a gramática que especifica a linguagem para verificar se o mesmo não contém erros. No exemplo, o programa contém uma declaração e três comandos, sendo que o segundo comando ("for") controla um comando de atribuição. Expressões comparecem na declaração da variável "fat", no comando "for" e no "printf".

Análise das dependências de contexto:
Verifica se os tipos dos dados são compatíveis nas operações de atribuição, comparação, adição e multiplicação, e também se a expressão de controle do comando iterativo produz um valor lógico. Verifica se todos os nomes foram declarados e não há duplicação no mesmo bloco. No exemplo, há dependências de contexto na inicialização da variável "fat", no "scanf", no "for" (quatro ocorrências) e no "printf". Além disso, todos os nomes usados foram previamente declarados ("fat", "i" e "n").

3ª Questão (1 ponto) - Conceitue *front-end* e *back-end* de um compilador. Explique as vantagens de se organizar um compilador dessa forma. Qual a importância da árvore de sintaxe abstrata nessa organização?

Front-end corresponde aos analisadores sintático (léxico incluído) e de contexto. Back-end corresponde ao gerador de código. Esse tipo de organização é viabilizada pelo uso de uma representação de árvore de sintaxe abstrata que possa servir para várias linguagens-fonte diferentes, e que funcione como interface entre o front e o back-end. A vantagem é que com m front-ends e n back-ends é possível dispor de $m * n$ compiladores diferentes, construindo-se apenas $m + n$ componentes.

4ª Questão (1 ponto) - Justificar, do ponto de vista da engenharia de software, a vantagem que obtém na organização de um compilador em múltiplos passos em comparação com a organização em um único passo.

Alta coesão:
As fases executam apenas as tarefas que lhe são inerentes, sem superposição de atribuições. Cada fase é auto-contida e não depende das demais para cumprir a sua tarefa.

Baixo acomplamento:
As interfaces entre os componentes que implementam as várias fases de um compilador são simples e bem-estruturadas, permitindo que os mesmo possam ser intercambiados sem grandes dificuldades.

5ª Questão (1 ponto) - Descreva, usando a notação dos diagramas-T, uma estratégia para a construção de um compilador auto-compilável para a linguagem C# que gere código para a máquina x86. A linguagem de desenvolvimento disponível é Java. Mostre como acontece a compilação e a execução de um programa C# na máquina x86.

Dados: (Java→JVM)/x86 e JVM/x86
(C#→x86)/Java --- (Java→JVM)/x86 --- (C#→x86)/JVM
(C#→x86)/C# --- (C#→x86)/JVM --- JVM/x86 --- (C#→x86)/x86
P/C# --- (C#→x86)/x86 --- P/x86

6ª Questão (1 ponto) - Um λ-termo (do cálculo lambda) é definido como:

a) $a, b, c, \dots, z$, que representam variáveis, são λ-termos;
b) Se M e N são λ-termos, então (MN) é um λ-termo;
c) Se M é um λ-termo e x é uma variável qualquer, então $(\lambda x.M)$ é um λ-termo.

São exemplos de λ-termos: $(\lambda x.(xy))$, $((\lambda y.y)(\lambda x.(xy)))$, $(x\ (\lambda x.(\lambda x.x)))$ e $\big(\lambda x.(yz)\big)$. Obtenha uma gramática que gere o conjunto de todos os λ-termos assim definidos.

$V \to a|b|\dots|z$
$L \to V$
$L \to (LL)$
$L \to (\lambda V.L)$

Exemplo: $L \Rightarrow (\lambda V.L) \Rightarrow (\lambda x.L) \Rightarrow (\lambda x.(LL)) \Rightarrow (\lambda x.(VL)) \Rightarrow (\lambda x.(xL)) \Rightarrow (\lambda x.(xy))$

7ª Questão (1 ponto) - Determinar se a gramática abaixo é LL(1). Justificar a sua resposta.

$S \to aS \mid X \mid YZ$
$X \to bX \mid Z$
$Y \to eW \mid \varepsilon$
$Z \to cS \mid ddX$

Z:
$first_1(cS) = \{c\}$
$first_1(ddX = \{d\}$
Condição LL(1) satisfeita.

Y:
$first_1(eW) = \{e\}$
$follow_1(Y) = \{c, d\}$
Condição LL(1) satisfeita.

X:
$first_1(bX) = \{b\}$
$first_1(Z) = \{c, d\}$

Condição LL(1) satisfeita.

S:
$first_1(aS) = \{a\}$
$first_1(X) = \{b, c, d\}$
$first_1(YZ) = \{e, c, d\}$
Como $first_1(X) \cap first_1(YZ) \neq \emptyset$, segue que a gramática em questão não é LL(1).

8ª Questão (1,5 ponto) - Obter uma gramática LL(1) que seja equivalente à gramática abaixo.

$S \rightarrow aX \mid aaY \mid aaaZ$
$X \rightarrow bX \mid b$
$Y \rightarrow Yc \mid c$
$Z \rightarrow dZd \mid e$

$S \rightarrow a(X \mid a(Y \mid aZ))$
$first_1(X) = \{b, \vdash\}$
$first_1(a(Y \mid aZ)) = \{a\}$
Condição LL(1) satisfeita
$first_1(Y) = \{c, \vdash\}$
$first_1(aZ) = \{a\}$
Condição LL(1) satisfeita

$X \rightarrow b^+$
$first_1(b^+) = \{b\}$
Condição LL(1) satisfeita

$Y \rightarrow c^+$
$first_1(c^+) = \{c\}$
Condição LL(1) satisfeita

$Z \rightarrow dZd \mid e$
$first_1(dZd) = \{d\}$
$first_1(e) = \{e\}$
Condição LL(1) satisfeita

9ª Questão (1,5 ponto) - Obter um esboço de reconhecedor sintático Java para a linguagem da questão 8.

```
void parseS() {
      accept ("a");
      switch (cT) {
            case "b":
            case "⊢":  parseX();
                       break
            case "a":  acceptIt();
                       switch (cT) {
                             case "c":
                             case "⊢":  parseY();
                                        break;
                             case "a":  acceptIt();
                                        parseZ();
                                        break;
                             }
            }
}

void parseX() {
      accept ("b");
      while (cT=="b") acceptIt();
```

```
}

void parseY() {
     accept ("c");
     while (cT=="c") acceptIt();
}

voide parseZ() {
     switch (cT) {
          case "d":  accept ("d");
                     parseZ();
                     accept ("d");
                     break;
          case "e":  acceptIt();
          }
}
```