COMPILADORES

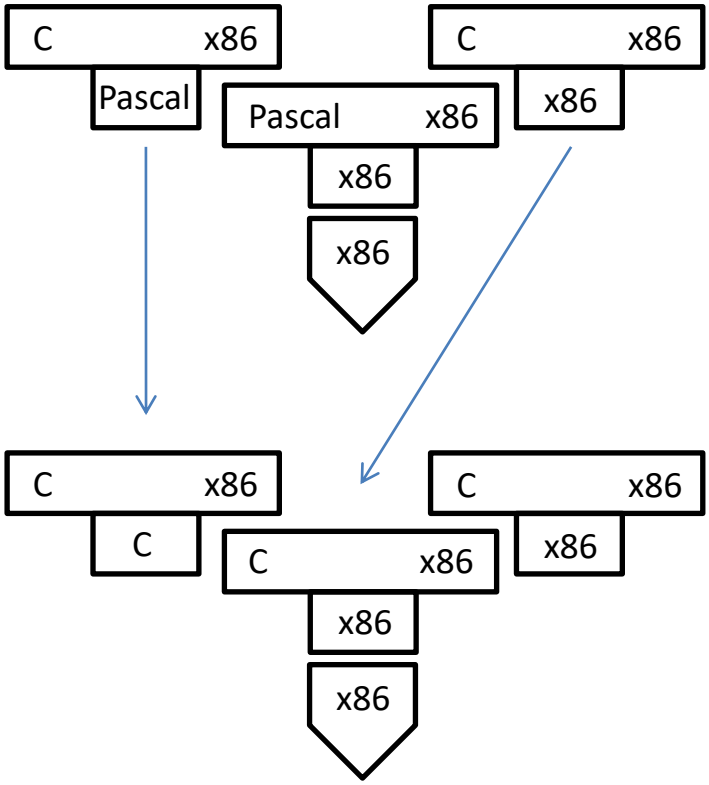
Prova 1 – 17/01/2019 – Prof. Marcus Ramos

Questão 1 (2 pontos): Justifique a importância do estudo de processadores de linguagens na formação do profissional de computação.

A importância decorre do fato de que linguagens de alto-nível fazem parte do dia-a-dia do profissional da área. Seja para programação, representação de dados, consultas diversas, interação homem-máquina, seja na forma compilada ou interpretada, as linguagens são parte indissociável da forma como trabalhamos e produzimos. Por isso, é fundamental conhecer e dominar os métodos para o processamento de tais linguagens. Legibilidade, portabilidade, segurança e produtividade são os principais fatores que motivam o uso de linguagens de alto-nível e, por conseguinte, justificam o estudo de processadores de linguagens. Assim, conhecer a teoria e dominar as técnicas usadas pelos processadores de linguagens é uma forma importante de garantir um uso cada vez melhor, mais consciente e mais frequente das linguagens de programação de alto-nível (e outras) tão presentes no nosso dia-a-dia. Além disso, o usuário se capacita a criar ferramentas computacionais para áreas de aplicação inovadoras ou já existentes.

Questão 2 (2 pontos): Mostre como a técnica de bootstrapping pode ser usada para implementar um compilador para a linguagem C usando um compilador já existente para a linguagem Pascal.

a) Escrever em Pascal um compilador que aceita como linguagem-fonte a linguagem C e compilar no compilador de Pascal;
b) Reescrever o compilador de Pascal em C;
c) Recompilar o compilador C.



Questão 3 (2 pontos): Justifique o maior interesse sobre compiladores organizados em vários passos em relação aos compiladores organizados em um único passo.

Compiladores de múltiplos passos favorecem a alta coesão e o baixo acoplamento entre os módulos componentes. Isto facilita a o desenvolvimento, o teste, a manutenção e a substituição de módulos, característica importante da Engenharia de Software para sistemas compostos por vários módulos. Os compiladores de um único passo não possuem estas características e eram assim construídos no passado por causa das limitações de velocidade e memória das máquinas antigas. Hoje em dia isso não se justifica mais.

Questão 4 (2 pontos): O que significa dizer que uma gramática é LL(1)? Como esta condição pode ser verificada?

Significa dizer que a correspondente linguagem pode ser reconhecida levando-se em conta apenas o não-terminal corrente e o símbolo de entrada corrente (token) usando derivações mais à esquerda. Ou seja, a regra escolhida em cada passo é determinada de maneira unívoca pelo não-terminal mais à esquerda na forma sentencial corrente e pelo símbolo corrente da entrada. A forma sentencial corrente pode eventualmente ser usada também no processo decisório, mas normalmente é desconsiderada a fim de simplificar o projeto do analisador sintático. Para verificar se uma gramática é LL(1), basta verificar se, para cada símbolo não-terminal da mesma, as respectivas regras iniciam com símbolos terminais distintos (conjuntos first distintos). Além disso, se alguma regra produzir a cadeia vazia, então o conjunto follow deve ser disjunto em relação ao conjunto first das regras não-vazias.

Questão 5 (2 pontos): Por que a gramática precisa ser LL(1) para que se possa usar o método recursivo descendente de análise sintática? O que acontece se o método for usado numa gramática que não é LL(1)?

O método recursivo descendente, utilizado na construção de analisadores sintáticos descendentes determinísticos exige, como pré-requisito, que a gramática de entrada seja LL(1) pois apenas assim é possível garantir a correção e a funcionalidade do programa resultante. Se a gramática não for LL(1), então alguns problemas podem acontecer com o código construído através do método recursivo descendente. Poderão haver problemas de compilação do código-fonte, se existirem duas ou mais regras para um mesmo símbolo não-terminal que comecem com o mesmo símbolo terminal (se for usado o comando switch-case). Se, de forma alternativa, for usado o comando if-else, poderão existir comandos que não são nunca executados. Além disso, se a gramática possuir recursão à esquerda, o analisador sintático poderá entrar em loop infinito. Ou seja, haverão problemas de compilação e/ou execução com o programa obtido, que dessa forma não implementa um analisador sintático para a linguagem gerada pela gramática em questão.