

Compiladores - 1^o bimestre

1 Introdução

Compiladores caracteriza-se como uma disciplina cumulativa, não segmentável. O que é visto no primeiro bimestre é essencial no segundo bimestre.

Faltas prejudicam a continuidade e devem ser evitadas, caso inevitável, sugere-se que o aluno assista o vídeo da aula e tire eventuais dúvidas com o professor, pois qualquer descontinuidade compromete o aprendizado.

- Avaliações bimestrais:
 - Prova: 50%
 - Trabalho: 50% (Individual ou duplas)
 - Exercícios individuais: até 1 ponto na média.
- Os trabalhos deverão ser apresentados online ao professor por ambos os alunos do grupo quando for o caso.
- É responsabilidade do aluno, caso trabalho individual, ou de ambos os alunos do grupo, se for o caso, garantir que o trabalho seja apresentado até o máximo 4 (quatro) dias **antes** da data limite para inserção das notas no sistema segundo o calendário da UFAL.
- A não apresentação no prazo significa que a nota do trabalho será zero, a não existência de horário disponível para apresentação é responsabilidade do aluno, por isso sugere-se apresentação antecipada.
- Os horários disponíveis para agendamento serão disponibilizado pelo professor na agenda do Google.
- O não comparecimento no horário agendado, com tolerância de 10 min, implicará em reagendamento, com perda de uma oportunidade de apresentação prévia (ver observação ao final do documento).
- O não agendamento implica na dependência de existência de horário disponível.
- Apresentações de última hora não serão aceitas.

2 Especificação mínima da linguagem (40%)

- A linguagem deve poder ser analisada em passo único.
- Como regra geral para definição, seguir a forma de análise dos trabalhos de CLP (ver documento anexo).
- Especificar a estrutura geral de um programa, indicando onde podem ser declaradas funções, variáveis, instruções, e qual o ponto inicial de execução.
- A especificação de tipos deve ser estática, com no mínimo o seguinte conjunto de tipos de dados:

- inteiro;
- ponto flutuante;
- caractere;
- booleano;
- cadeia de caracteres (deve ser um tipo primitivo);
- arranjos unidimensionais (especificar como o tamanho é

- Especificar as constantes literais de cada tipo.
- Especificar as operações de cada tipo.
- Especificar coerções eventualmente suportadas.
- Conjunto mínimo de operadores

Especificar ordem de precedência e associatividade:

- Aritméticos: para tipos numéricos
 - * aditivos, multiplicativos;
 - * unário negativo (sugere-se o uso de unário positivo também);
- Relacionais:
 - * Caso sejam associativos, a semântica de dois operadores vizinhos deve ser especificada;
 - * para tipos numéricos, caracteres e cadeias de caracteres: todos;
 - * para tipos booleanos: igualdade e desigualdade;
- Lógicos: para tipo booleano
 - * negação, conjunção e disjunção;
- Concatenação: geram cadeias de caracteres
 - * tipos caracteres e cadeias de caracteres;
 - * tipos numéricos e booleanos: se concatenados a um caractere ou cadeia de caracteres, deve ser convertido para cadeia de caracteres;
 - * deve ser especificado se uma eventual conversão para cadeia de caracteres deve implícita ou explícita;
 - * deve permitir formatação como está definido para as saídas (ver tópico abaixo).

• Instruções

Especificar natureza, formas de controle e semântica:

- **Estrutura condicional:** de uma e duas vias.
- **Estrutura iterativa com controle lógico:**
 - * especificar se é pré ou pós-teste.
- **Estrutura iterativa controlada por contador:**
 - * deve permitir a omissão do valor do incremento do laço, devendo assumir neste caso o valor um;
 - * em C e Java o comando **for** é uma estrutura iterativa com controle lógico e portanto este tipo de comando não atende este quesito.
- **Entrada:**
 - * deve permitir entrada de mais de uma variável em uma única instrução;

- * se a entrada for formatada, o formato deve estar colocado imediatamente à esquerda da variável a ser lida (ver especificação de saída abaixo).
- **Saída:**
 - * deve permitir emissão de mais de uma variável/constante literal em uma única instrução;
 - * deve ser especificada uma forma para definir quebra de linha;
 - * deve permitir formatação de:
 - # tamanho total ocupado pelo elemento na saída;
 - # alinhamento à esquerda ou à direita no campo;
 - # adicionalmente no caso de valores de ponto flutuante, o número de casas decimais;
 - * quando presente o formato deve ser colocado imediatamente à esquerda do elemento a ser exibido.
- **Atribuição:**
 - * especificar se é comando ou operador;
 - * caso seja operador devem ser especificadas precedência e associatividade;
 - * caso seja comando, deve ser especificado se é permitido que algum resultado de expressão seja descartado.
- **Funções**
 - Especificar modelos semânticos de passagem de parâmetros suportados e forma de implementação (ver Sebesta para detalhes, ou com o professor)
- **Programs exemplos**
 - Incluir os seguintes programas exemplos, que deverão ser testados usando o analisador léxico:
 - * **Alô mundo;**
 - * **Série de Fibonacci:** implementar em uma função, que deve listar os elementos da série menores que um valor limite, separados por vírgula, usando pelo menos uma iteração com controle lógico; o limite deve ser lido no programa principal;
 - * **Shell sort:** implementado em uma função usando pelo menos uma iteração controlada por contador, em um arranjo cujos valores devem ser lidos e listados no programa principal, também no programa principal listar os valores ordenados.

3 Especificação dos tokens (20%)

- Especificar a linguagem de programação em que os analisadores léxico e sintático serão implementados.
- Especificar a enumeração com as categorias dos tokens a ser obrigatoriamente usada nos analisadores léxico e sintático, usando a sintaxe da linguagem escolhida para a implementação dos analisadores; nomes simbólicos de até 20 caracteres.
- Especificar em dois grupos distintos:
 - Tabela com as expressões regulares auxiliares, que não representam terminais da linguagem, caso existirem;
 - Tabela com as categorias simbólicas dos tokens, especificadas na enumeração de categorias, e as expressões regulares dos lexemas correspondentes, que representam os terminais da linguagem.
- A especificação das expressões regulares devem seguir os padrões do Flex (arquivo anexo).

4 Analisador Léxico (40%)

- Deve ser implementado para fazer a análise “*on the fly*”, o analisador léxico deve ler e listar cada linha do programa fonte de cada vez, apenas quando a linha anterior tiver sido totalmente analisada a próxima linha pode ser lida.
- Imprimir as linhas de código numerando-as no formato “%d4_” (4 dígitos, alinhados à direita com preenchimento de espaços em branco à esquerda, seguido de 2 espaços em branco).
- Cada chamada do analisador sintático, simulado pelo programa de teste nesta fase (ver abaixo), deve retornar um único *token*. O método/função a ser usado pelo analisador sintático deve ter a assinatura:

```
Token nextToken();
```

ou similar, dependendo da linguagem de programação usada.

O arquivo fonte não deve ser lido inteiro em memória.

O tipo de dado do **Token** deverá ser um registro ou classe com os dados do *token*.

- Erros léxicos devem ser enviados para o analisador sintático tratar.
- **Programa para testar o analisador léxico:**
 - O nome do programa analisado deve ser passado na linha de comando do programa de teste.
 - O programa deverá acessar nextToken() listando para cada *token*, nesta ordem:
 - * posição: linha e coluna no programa exemplo;
 - * categoria: o número e nome associados na enumeração;
 - * lexema (valor léxico);
 - * o formato de impressão deve seguir o modelo a seguir (em C):
“`_____ [%04d, _%04d] _ (%04d, _%20s) _ { %s }`”
Obs.: indentação de 14 casas; caso o código numérico das categorias ultrapassar 4 dígitos, usar o número par superior mais próximo, garantindo o alinhamento dos campos.
- Como o analisador léxico imprime as linhas à medida que as lê, os tokens deverão estar abaixo da linha onde apareceram
- Devem ser apresentados os resultados dos testes para os três programas previamente solicitados.

AS ESPECIFICAÇÕES AQUI DEFINIDAS DEVEM SER SEGUÍDAS LITERALMENTE.

- Revise frequentemente estas especificações para garantir que seu trabalho está em conformidade com as mesmas.
- A aderência à uma especificação é uma necessidade de suma importância em um ambiente de produção, aproveitem esta oportunidade de aprendizagem.

SUGERE-SE FORTEMENTE QUE:

- Versões prévias sejam apresentadas ao professor para análise e discussão.
 - Cada trabalho pode ser apresentado previamente até duas vezes.
 - Apresentações prévias são permitidas até uma semana antes da data limite para digitação da nota no sistema de acordo com o calendário da UFAL.

- Seja usado o github para manter o trabalho, neste caso resultados poderão ser apresentados indicando o endereço relativo no repositório (o repositório será clonado pelo professor).
- Caso isso não seja feito, todos os fontes devem ser disponibilizados para serem copiados pelo professor pelo menos 1 hora **antes** da apresentação.
- Em qualquer dos casos, instruções para compilação/interpretação e execução dos programas devem ser fornecidos para execução a partir de terminal.