Projeto de Compiladores 2022/23

Compilador para a linguagem Juc

26 de setembro de 2022

Este projeto consiste no desenvolvimento de um compilador para a linguagem Juc, que é um subconjunto da linguagem Java de acordo com a especificação Java SE 9 (disponível na página `https://docs.oracle.com/javase/specs/jls/se9/html/index.html`).

Na linguagem Juc é possível usar variáveis e literais dos tipos `boolean`, `int` e `double` (estes dois últimos com sinal). É também possível usar literais do tipo `String`, apenas para efeito de escrita no *stdout*. A linguagem Juc inclui expressões aritméticas e lógicas, instruções de atribuição, operadores relacionais e instruções de controlo (`while` e `if-else`). Inclui também métodos estáticos com os tipos de dados já referidos e ainda o tipo especial `String[]`, sendo a passagem de parâmetros sempre feita por valor e podendo ou não ter valor de retorno. A ausência de valor de retorno é identificada pela palavra-chave `void`.

Os programas da linguagem Juc são compostos por uma única classe (principal) contendo métodos e atributos, todos eles estáticos. O método `main(...)` invocado no início de cada programa pode receber parâmetros, que deverão ser literais inteiros, através da linha de comandos. Supondo que o parâmetro formal do método `main(...)` é `args`, os respetivos valores podem ser obtidos através do método pré-definido `Integer.parseInt(args[...])` e a expressão `args.length` dá o número de parâmetros. O método pré-definido `System.out.print(...)` permite escrever na consola valores lógicos, inteiros, reais e strings.

O significado de um programa na linguagem Juc será o mesmo que na linguagem Java, assumindo a pré-definição dos métodos `Integer.parseInt(...)` e `System.out.print(...)`, bem como da construção `.length`. Por fim, são aceites comentários nas formas `/* ... */` e `// ...` que deverão ser ignorados. Assim, por exemplo, o programa que se segue calcula o fatorial de um número passado como argumento:

```
class Factorial {
    public static int factorial(int n) {
        if (n == 0)
            return 1;
        return n * factorial(n-1);
    }

    public static void main(String[] args) {
        int argument;
        argument = Integer.parseInt(args[0]);
        System.out.print(factorial(argument));
    }
}
```

O programa anterior declara uma variável `argument` do tipo `int` e atribui-lhe o valor inteiro do argumento passado ao programa, usando o método `parseInt(...)` para realizar a conversão. De seguida, calcula o fatorial desse valor e invoca o método `print(...)` para escrever o resultado na consola.

Metas e avaliação

O projeto está estruturado em quatro metas encadeadas, nas quais o resultado de cada meta é o ponto de partida para a meta seguinte. As datas e as ponderações são as seguintes:

1. Análise lexical (19%) – 14 de outubro de 2022
2. Análise sintática (25%) – 2 de novembro de 2022 (meta de avaliação)
3. Análise semântica (25%) – 18 de novembro de 2022
4. Geração de código (25%) – 9 de dezembro de 2022 (meta de avaliação)

A entrega final será acompanhada de um relatório que tem um peso de 6% na avaliação. Para além disso, a entrega final do trabalho deverá ser feita através do Inforestudante, até ao dia seguinte ao da Meta 4, e incluir todo o código-fonte produzido no âmbito do projeto (exatamente os mesmos arquivos `.zip` que tiverem sido colocados no MOOSHAK em cada meta).

O trabalho será verificado no MOOSHAK em cada uma das metas usando um concurso criado para o efeito. A classificação final da Meta 1 é obtida em conjunto com a Meta 2 e a classificação final da Meta 3 é obtida em conjunto com a Meta 4. O nome do grupo a registar no MOOSHAK é obrigatoriamente da forma "uc2019123456_uc2019654321" usando os números de estudante como identificação do grupo na página `https://mooshak.dei.uc.pt/~comp2022` na qual o MOOSHAK está acessível. Será tida em conta apenas a última submissão ao problema A de cada concurso do MOOSHAK para efeitos de avaliação.

Defesa e grupos

O trabalho será realizado por grupos de dois alunos inscritos em turmas práticas do mesmo docente. Em casos excecionais, a confirmar com o docente, admite-se trabalhos individuais. A defesa oral do trabalho será realizada em grupo na semana seguinte à entrega da Meta 4. A nota final do projeto é limitada pela soma ponderada das pontuações obtidas no MOOSHAK em cada uma das metas e diz respeito à prestação individual na defesa. Assim, a classificação final nunca poderá exceder a pontuação obtida no MOOSHAK acrescida da classificação do relatório final. Aplica-se mínimos de 40% à nota final após a defesa. Os programas de teste colocados no repositório `https://git.dei.uc.pt/rbarbosa/Comp2022/tree/master` por cada estudante serão contabilizados na avaliação.

1 Meta 1 – Analisador lexical

Nesta primeira meta deve ser programado um analisador lexical para a linguagem Juc. A programação deve ser feita em linguagem C utilizando a ferramenta *lex*. Os "tokens" a ser considerados são apresentados de seguida e deverão estar de acordo com a especificação da linguagem Java, disponível em `https://docs.oracle.com/javase/specs/jls/se9/html/jls-3.html` na sua versão original.

1.1 Tokens da linguagem Juc

ID: sequências alfanuméricas começadas por uma letra, onde os símbolos "_" e "$" contam como letras. Letras maiúsculas e minúsculas são consideradas letras diferentes.

INTLIT: representa uma constante inteira composta pelo dígito zero, ou sequências de dígitos decimais ou "_", começadas por um dígito diferente de zero e terminadas num dígito.

REALLIT: uma parte inteira seguida de um ponto, opcionalmente seguido de uma parte fracionária e/ou de um expoente; ou um ponto seguido de uma parte fracionária, opcionalmente seguida de um expoente; ou uma parte inteira seguida de um expoente. O expoente consiste numa das letras "e" ou "E" seguida de um número opcionalmente precedido de um dos sinais "+" ou "-". Tanto a parte inteira como a parte fracionária e o número do expoente consistem em sequências de dígitos decimais ou "_" começadas e terminadas por um dígito.

STRLIT: uma sequência de carateres (exceto "carriage return", "newline" e aspas duplas) e/ou "sequências de escape" entre aspas duplas. Apenas as sequências de escape \f, \n, \r, \t, \\ e \" são especificadas pela linguagem. Sequências de escape não especificadas devem dar origem a erros lexicais, tal como se detalha mais adiante.

BOOLLIT = "true" | "false"

AND = "&&"

ASSIGN = "="

STAR = "*"

COMMA = ","

DIV = "/"

EQ = "=="

GE = ">="

GT = ">"

LBRACE = "{"

LE = "<="

LPAR = "("

LSQ = "["

LT = "<"

MINUS = "-"

MOD = “%”

NE = “!=”

NOT = “!”

OR = “||”

PLUS = “+”

RBRACE = “}”

RPAR = “)”

RSQ = “]”

SEMICOLON = “;”

ARROW = “->”

LSHIFT = “<<”

RSHIFT = “>>”

XOR = “^”

BOOL = “boolean”

CLASS = “class”

DOTLENGTH = “.length”

DOUBLE = “double”

ELSE = “else”

IF = “if”

INT = “int”

PRINT = “System.out.print”

PARSEINT = “Integer.parseInt”

PUBLIC = “public”

RETURN = “return”

STATIC = “static”

STRING = “String”

VOID = “void”

WHILE = “while”

RESERVED: palavras reservadas da linguagem Java não utilizadas em Juc bem como o operador de incremento (“++”), o operador de decremento (“−−”), o literal “null” e os identificadores “Integer” e “System”.

1.2 Programação do analisador

O analisador deverá chamar-se `jucompiler`, ler o ficheiro a processar através do *stdin* e, quando invocado com a opção `-l`, deve emitir os tokens e as mensagens de erro para o *stdout* e terminar. Na ausência de qualquer opção, ou se invocado com a opção `-e1`, deve escrever no *stdout* apenas as mensagens de erro. Por exemplo, caso o ficheiro `Factorial.java` contenha o programa de exemplo dado anteriormente, que calcula o fatorial de números, a invocação:

```
jucompiler -l < Factorial.java
```

deverá imprimir a correspondente sequência de tokens no ecrã. Neste caso:

```
CLASS
ID(Factorial)
LBRACE
PUBLIC
STATIC
INT
ID(factorial)
LPAR
INT
ID(n)
RPAR
LBRACE
...
```

Figura 1: Exemplo de output do analisador lexical. O output completo está disponível em: `https://git.dei.uc.pt/rbarbosa/Comp2022/blob/master/meta1/Factorial.out`

O analisador deve aceitar (e ignorar) como separador de tokens o espaço em branco (espaços, tabs e mudanças de linha), bem como comentários dos tipos `// ...` e `/* ... */`. Deve ainda detetar a existência de quaisquer erros lexicais no ficheiro de entrada. Sempre que um token possa admitir mais do que um valor semântico, o valor encontrado deve ser impresso entre parêntesis logo a seguir ao nome do token, como exemplificado na figura acima para ID.

1.3 Tratamento de erros

Caso o ficheiro contenha erros lexicais, o programa deverá imprimir exatamente uma das seguintes mensagens no *stdout*, consoante o caso:

```
Line <num linha>, col <num coluna>: illegal character (<c>)\n
Line <num linha>, col <num coluna>: invalid escape sequence (<c>)\n
Line <num linha>, col <num coluna>: unterminated comment\n
Line <num linha>, col <num coluna>: unterminated string literal\n
```

onde `<num linha>` e `<num coluna>` devem ser substituídos pelos valores correspondentes ao *início* do token que originou o erro, e `<c>` devem ser substituídos por esse token. Tanto as linhas como as colunas são numeradas a partir de 1. O analisador deve recuperar da ocorrência de erros lexicais a partir do *fim* do respetivo token. No caso de uma string não terminada que inclua sequências de escape inválidas, o erro de string não terminada deve ser apresentado após os erros de sequência inválida.

1.4 Entrega da Meta 1

O ficheiro *lex* a entregar deverá obrigatoriamente identificar os autores num comentário no topo desse ficheiro, contendo o nome e o número de estudante de cada elemento do grupo. Esse ficheiro deverá chamar-se `jucompiler.l` e ser enviado num arquivo de nome `jucompiler.zip` que não deverá ter quaisquer diretorias.

O trabalho deverá ser verificado no MOOSHAK usando o concurso criado especificamente para o efeito. Será tida em conta apenas a última submissão ao problema A desse concurso. Os restantes problemas destinam-se a ajudar na verificação do analisador. No entanto, o MOOSHAK não deve ser utilizado como ferramenta de depuração. Os estudantes devem usar e contribuir para o repositório disponível em `https://git.dei.uc.pt/rbarbosa/Comp2022/tree/master` contendo casos de teste. A página do MOOSHAK está indicada no início do presente documento.