Projeto de Compiladores 2023/24

Compilador para a linguagem UC

28 de setembro de 2023

Este projeto consiste no desenvolvimento de um compilador para a linguagem UC, que é um subconjunto da linguagem C (de acordo com o padrão C99).

Na linguagem UC é possível usar variáveis e literais do tipo `char`, `short`, `int`, e `double` (todos com sinal). A linguagem UC inclui expressões aritméticas e lógicas, instruções de atribuição, operadores relacionais, e instruções de controlo (`if-else` e `while`). Inclui também funções com os tipos de dados já referidos, sendo a passagem de parâmetros sempre feita por valor. A ausência de parâmetros de entrada ou de valor de retorno é identificada pela palavra-chave `void`.

A função invocada no início de cada programa chama-se `main`, tem valor de retorno do tipo `int` e não recebe parâmetros, sendo que o programa `int main(void) { return 0; }` é um dos mais pequenos possíveis na linguagem UC. Os programas podem ler e escrever carateres na consola através das funções pré-definidas `getchar()` e `putchar()`, respetivamente.

O significado de um programa na linguagem UC será o mesmo que em C99, assumindo a pré-definição das funções `getchar()` e `putchar()`. Por fim, são aceites comentários nas formas `/* ... */` e `// ...` que deverão ser ignorados. Assim, por exemplo, o programa que se segue imprime na consola os carateres de A a Z:

```
int main(void) {
  char i = 'A';
  while (i <= 'Z') {
    putchar(i);
    i = i + 1;
  }
  return 0;
}
```

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%) – 20 de outubro de 2023
2. Análise sintática (25%) – 10 de novembro de 2023 (meta de avaliação)
3. Análise semântica (25%) – 24 de novembro de 2023
4. Geração de código (25%) – 15 de dezembro de 2023 (meta de avaliação)

Um relatório com um peso de 6% na avaliação acompanhará a entrega final. 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. 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 "uc2020123456_uc2020654321" usando os números de estudante como identificação do grupo na página `http://mooshak.dei.uc.pt/~comp2023` na qual o MOOSHAK está acessível. Será tida em conta apenas a última versão apresentada ao problema A de cada concurso do MOOSHAK para efeitos de avaliação.

Defesa e grupos

O trabalho será realizado por grupos de dois alunos, preferencialmente inscritos na mesma turma prática. Em casos excecionais, a confirmar com o docente, admite-se trabalhos individuais. A defesa oral do trabalho será realizada em grupo após a entrega da Meta 4. A nota final do projeto é limitada pela soma ponderada das classificações obtidas no MOOSHAK em cada uma das metas e diz respeito à prestação individual na defesa. Assim, a classificação final nunca poderá exceder a classificação obtida no MOOSHAK acrescida da classificação do relatório. Os testes colocados no repositório `https://git.dei.uc.pt/rbarbosa/Comp/tree/master/c` por cada estudante serão contabilizados na avaliação.

1 Meta 1 – Analisador lexical

Nesta primeira meta deve ser programado um analisador lexical para a linguagem UC. A programação deve ser feita em linguagem C utilizando a ferramenta *lex*. Os "tokens" a ser considerados pelo compilador deverão estar de acordo com o padrão C99[1] e são apresentados de seguida.

1.1 Tokens da linguagem UC

IDENTIFIER: sequências alfanuméricas começadas por uma letra, onde o símbolo "_" conta como uma letra. Letras maiúsculas e minúsculas são consideradas letras diferentes.

NATURAL: sequências de dígitos de base dez (0–9).

DECIMAL: 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 de base dez (0–9).

CHRLIT: um único caráter (excepto *newline* ou aspa simples) ou uma "sequência de escape" entre aspas simples. Apenas as sequências de escape \n, \t \\, \', \" e \ooo são especificadas pela linguagem, onde ooo representa uma sequência de 1 a 3 dígitos entre 0 e 7. A ocorrência de uma sequência de escape inválida ou de mais do que um caráter ou sequência de escape entre aspas simples deve dar origem a um erro lexical.

[1] ISO C 1999 Standard – `https://tinyurl.com/c1999standard`

CHAR = char

ELSE = else

WHILE = while

IF = if

INT = int

SHORT = short

DOUBLE = double

RETURN = return

VOID = void

BITWISEAND = “&”

BITWISEOR = “|”

BITWISEXOR = “^”

AND = “&&”

ASSIGN = “=”

MUL = “*”

COMMA = “,”

DIV = “/”

EQ = “==”

GE = “>=”

GT = “>”

LBRACE = “{”

LE = “<=”

LPAR = “(”

LT = “<”

MINUS = “-”

MOD = “%”

NE = “!=”

NOT = “!”

OR = “||”

PLUS = “+”

RBRACE = “}”

RPAR = “)”

SEMI = “;”

RESERVED: palavras reservadas da linguagem C não utilizadas em UC, bem como os símbolos “[”, “]”, o operador de incremento (“++”) e o operador de decremento (“−−”).

1.2 Programação do analisador

O analisador deverá chamar-se `uccompiler`, 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 deve escrever no *stdout* apenas as mensagens de erro. Caso o ficheiro `first.c` contenha o programa de exemplo apresentado anteriormente, que imprime os carateres de A a Z, a invocação

```
./uccompiler -l < first.c
```

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

```
INT
IDENTIFIER(main)
LPAR
VOID
RPAR
LBRACE
CHAR
IDENTIFIER(i)
ASSIGN
CHRLIT('A')
SEMI
WHILE
LPAR
...
```

Figura 1: Exemplo de resultado do analisador lexical. O resultado completo está disponível em: `https://git.dei.uc.pt/rbarbosa/Comp/blob/master/c/meta1/first.out`

O analisador deve aceitar (e ignorar) como separador de tokens o espaço em branco (espaços, tabs e mudanças de linha), bem como comentários do tipo /* ... */ 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 à categoria do token, como exemplificado acima para IDENTIFIER e CHRLIT.

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>, column <num coluna>: unrecognized character (<c>)\n"
"Line <num linha>, column <num coluna>: invalid char constant (<c>)\n"
"Line <num linha>, column <num coluna>: unterminated comment\n"
"Line <num linha>, column <num coluna>: unterminated char constant\n"

onde <num linha> e <num coluna> devem ser substituídos pelos valores correspondentes ao *início* do token que originou o erro, e <c> deve ser substituído por esse token. O analisador deve recuperar da ocorrência de erros lexicais a partir do *fim* desse token.

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 `uccompiler.l` e ser enviado num arquivo de nome `uccompiler.zip` que não deverá ter quaisquer diretorias.

O trabalho deverá ser verificado no MOOSHAK usando o concurso criado para o efeito. Será tida em conta apenas a última versão apresentada 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/Comp/tree/master/c` contendo casos de teste. A página do MOOSHAK está indicada no início deste enunciado.