

# Projeto de Compiladores 2020/21

## Compilador para a linguagem UC

7 de outubro de 2020

Este projeto consiste no desenvolvimento de um compilador para a linguagem UC, que é um subconjunto da linguagem C (de acordo com o standard 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 caracteres 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 caracteres de A a Z:

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

## 1 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%) – 17 de outubro de 2020
2. Análise sintática (25%) – 14 de novembro de 2020 (meta de avaliação)
3. Análise semântica (25%) – 30 de novembro de 2020
4. Geração de código (25%) – 21 de dezembro de 2020 (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 “uc2020123456\_uc2020654321” usando os números de estudante como identificação do grupo na página <http://mooshak2.dei.uc.pt/~compiladores> 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.

## 1.1 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/Comp2020/tree/master> por cada estudante serão contabilizados na avaliação.

## 2 Meta 1 – Analisador lexical

Nesta primeira meta deve ser programado um analisador lexical para a linguagem UC. A programação deve ser feita recorrendo à linguagem de programação C utilizando a ferramenta *lex*. Os “tokens” a ser considerados pelo compilador deverão estar de acordo com o C99 standard<sup>1</sup> e são apresentados de seguida.

### 2.1 Tokens da linguagem UC

ID: sequências alfanuméricas começadas por uma letra, onde o símbolo “\_” conta como uma letra. Letras maiúsculas e minúsculas são consideradas letras diferentes.

INTLIT: sequências de dígitos decimais (0–9).

CHRLIT: um único carácter (excepto *newline* ou aspa simples) ou uma “sequência de escape” entre aspas simples. Apenas as sequências de escape `\n`, `\t`, `\\`, `\'`, `\"` e `\ooo` são definidas 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 carácter ou sequência de escape entre aspas simples deve dar origem a um erro lexical.

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

---

<sup>1</sup>ISO C 1999 Standard – <https://tinyurl.com/compiladores2020>

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.

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 (“--”).

## 2.2 Programação do analisador

O analisador deverá chamar-se *ucompiler*, 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. Caso o ficheiro *first.uc* contenha o programa de exemplo dado anteriormente, que imprime os caracteres de A a Z, a invocação:

```
./ucompiler -l < first.uc
```

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

```
INT
ID(main)
LPAR
VOID
RPAR
LBRACE
CHAR
ID(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/Comp2020/blob/master/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 ao nome do token, como exemplificado acima para ID e INTLIT.

## 2.3 Tratamento de erros

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

```
"Line <num linha>, col <num coluna>: invalid char constant (<c>)\n"
```

```
"Line <num linha>, col <num coluna>: unterminated comment\n"
```

```
"Line <num linha>, col <num coluna>: unterminated char constant\n"
```

```
"Line <num linha>, col <num coluna>: illegal character (<c>)\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. O analisador deve recuperar da ocorrência de erros lexicais a partir do *fim* desse token.

## 2.4 Submissão 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 *ucompiler.1* e ser enviado num arquivo de nome *ucompiler.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/Comp2020/tree/master> contendo casos de teste. A página do MOOSHAK está indicada na Secção 1.