Manual de utilização

Como obter e usar a linguagem quiz

(1234) Author One some@mail
(4321) Author Two another@mail

June 20, 2021

Contents

1 Apresentação

A linguagem quiz é uma ferramenta que permite criar questionários ricos com esforço mínimo.

Com esta ferramenta pode especificar perguntas, controlar a sua apresentação, ordem e cotação, sem grande esforço com a interface com o utilizador.

2 Obter as ferramentas

2.1 Compilador

2.1.1 Download

COLOCAR NOS RELEASES

2.1.2 Copiar do código fonte

INDICAR DEPENDÊNCIAS E INSTRUÇÕES DE COMPILAÇÃO

2.2 Relatórios e manuais

2.2.1 Download

ATUALIZAR

O relatório em pdf pode ser encontrado em doc/Relatório.

O manual de utilização pode ser visto em markdown ou em pdf em doc/manual

2.2.2 Compilar do código fonte

Para o relatório é necessário `pdflatex`, o manual de utilizador requer ainda `pandoc`

Para instalar as dependencias basta correr os comandos seguintes:

Em ubuntu:

```
sudo apt install pandoc texlive
```

Em arch:

`sudo pacman -S pandoc texlive-core`

Para compilar o relatório deve clonar ou baixar o repositório, dirigir-se a `doc/Relatório` e correr **COMANDOS**

Para compilar o manual basta ir a `doc/manual` e correr `sh compile.sh`, ou em alternativa executar diretamente o comando no ficheiro `pandoc --template=template.tex --toc user_manual.md -o manual.pdf`

3 Linguagem principal

3.1 Variáveis

Todas as variáveis têm tipo, nome e valor. Na sua declaração todos têm de ser explicitados.

```
# Declaração geral de uma variável
type name = value;
```

onde `name` é o nome da variável, podendo ser constituido por números, letras, '_' ou '-' e que começa com uma letra.

~~Sendo boas variáves escritas assim "x_1" mas **nunca** desta forma "_1x" ou desta "1_x".~~

A convensão da linguagem sugere utilizar *camelCase* ou *snake_case*.

Variáveis podem ter os seus valores alterados a qualquer momento por uma atribuição com a seguinte sintaxe:

```
name = newValue;
```

Onde `name` é o nome da variável a que se quer mudar o valor e `newValue` é o novo valor.

Importante destacar que `newValue` tem de ser do mesmo tipo da variável, ou de um tipo que nele possa ser convertido.

3.1.1 Tipos

Os tipos das variáveis indicam ao compilador como lidar com elas, ie. como operar entre elas.

Para maior facilidade de compreensão do código e de criação do compilador, esta linguagem é **fortemente** e **estaticamente tipada**. Significa que as variáveis têm tipos definidos e não podem ser mutados.

tipo	dados que guarda
num	números de vírgula flutuante
text	string, texto
time	tempo em segundos, minutos ou horas

tipo	dados que guarda
list	conjunto de variáveis de quaisquer tipos
question	pergunta

3.1.1.1 Num

Números são valores reais.

São internamente guardados como *double* do java, ou seja, valores reais com **15 casas decimais**.

3.1.1.2 Text

Text guarda uma *string*, ou seja, qualquer conjunto de carateres.

São especificadas entre aspas duplas (““).

3.1.1.3 Time

Time guarda tempo em segundos, munutos ou horas.

É identificado por `<num><s|m|h>`, onde `num` é um número que precede o prefixo de tempo, `s` para segundos, `m` para minutos e `h` para horas.

Por exemplo, `2h` representa 2 horas.

É ainda possível em vez de escrever um número escrever uma expressão, desde que ela seja do tipo `num`, como `time t = (1+2)s`.

3.1.1.4 List

As listas conservam a ordem de entrada, os elementos estão dispostos na ordem pela qual foram inseridos.

Listas podem guardar qualquer tipo de dados, é importante ter isso em conta pois manipular tipos de dados diferentes da mesma maneira pode dar resultados indesejados.

É ainda possível iterar por todos os elementos de uma lista, mais detalhes em for each.

3.1.1.5 Question

Question guarda perguntas, bem como a respetiva resposta do utilizador.

Parâmetro	tipo	explicação
id	text	O identificador único da pergunta. Apenas é definido nas perguntas vindas de uma base de dados
question	text	O texto da pergunta
type	text	O tipo da pergunta (`match`, `multiple`, `short`, `long`, `numeric`)
dependencies	list	As dependências da pergunta
groups	list	Os grupos a que a pergunta pertence
possibleAns	list	As respostas possíveis (perguntas do tipo `short` e `long` não têm este campo definido (lista vazia))

Parâmetro	tipo	explicação
correct	list/text	Validação das respostas corretas
ans	list/text	Resposta dada pelo utilizador

Em perguntas do tipo `long` a propriedade `possibleAns` é do tipo `text`, interpretado como regex para validar a pergunta.

Se as combinações de `possibleAns` e `correct` seguintes não forem respeitadas ocorrerá um erro.

tipo	possibleAns	correct
match	lista com duas listas	lista de listas de dois elementos
multiple	lista	lista sem listas
short	irrelevante	lista com um número (limite de palavras) e expressão regular que valida a pergunta
long	irrelevante	expressão regular
numeric	irrelevante	lista com números (se só o número for correto) ou de listas com dois números (intervalo)

3.2 Operações aritméticas

As próximas secções listam as possíveis combinações de tipos de operando e no que resultam as operações entre eles.

Todas as combinações não listadas não estão defenidas e retornam erro.

3.2.1 Adição

primeiro operando	segundo operando	resultado
num	num	Soma algébrica dos números
num	list	Adiciona o número à lista
num	text	Se o texto puder ser convertido em número adiciona os números, se não dá erro
num	list	Adiciona o número à lista
text	text	Adiciona os textos pela ordem indicada
text	list	**Apenas para listas de texto!** Adiciona todos os elementos numa única string
list	list	Retorna uma lista com os elementos da primeira seguidos dos da segunda

3.2.2 Subtração

primeiro operando	segundo operando	resultado
num	num	Subtração algébrica

3.2.3 Divisão

primeiro operando	segundo operando	resultado
num	num	Subtração algébrica

3.2.4 Multiplicação

primeiro operando	segundo operando	resultado
num	num	Multiplicação algébrica
num	text	Um texto com o texto original `num` vezes seguidas

3.2.5 Inversão

A inversão é unária. A sintaxe é `-<exp>`, onde `exp` é uma expressão dos tipos abaixo indicados.

operando	resultado
num	O número negado (por exemplo, 2 → -2)
text	O texto invertido (“teste” → “etset”)
list	A lista pela ordem inversa

3.2.6 Incremento e Decremento

Estas operações apenas se aplicam a números.

Aumenta/diminui em 1 o valor da variável.

```
a++; # pós-incremento
++b; # pré-incremento

a--; # pós-decremento
--b; # pré-decremento
```

Quando numa expressão, a posição dos sinais importa. Se os sinais **precedem** a variável, ela é incrementada **antes** de ser usada na operação. Se **sucedem** a variável, esta só é incrementada **após** a operação.

```
num a = 1;
```

```
num b = 2;

num c = a++ + b; # 1 + 2
# c = 3
# a = 2
# b = 2

num d = a + ++b; # 2 + 3
# d = 5
# a = 2
# b = 3
```

De notar que, no **pós-incremento** o valor da variável incrementada/decrementada é atualizado **logo após** ser usado.

```
num a = 1;

#       1   + 5 + 2 + 3   - 3
num b = a++ + 5 + a + ++a - a;
```

3.3 Operações lógicas

3.3.1 Comparações

Comparações dependem de relações de ordem entre os elementos a comparar.

A linguagem quiz permite todas as comparações standart da matemática, igual (`==`), menor (`<`), menor ou igual (`<=`), maior (`>`) e maior ou igual (`>=`).

tipo 1	tipo 2	resultado
num	num	Compara os números
num	text	Compara o número com o comprimento do texto
num	list	Compara o número com o comprimento da lista
text	text	Compara o comprimento dos textos

A ordem pela qual os tipos aparecem não é relevante.

3.3.2 E e Ou

Os operandos são, respetivamente, `&&` e `||`.

Se os operandos forem ambos números, operam com os valores lógicos deles (1 - verdadeiro, outro - falso).

Se não, a operação **e** executa a adição dos operandos.

3.4 Comentários

Comentário é texto que o programa ignora, isto é, não tem qualquer impacto no código.

~~No entanto é das ferramentas mais poderosas e fortemente encorajadas a serem usadas~~ No entanto é uma ferramente extremamente útil, tornando o código não só mais legivel mas também mais apto a modificações por alguém que não o autor.

Quer em quiz quer na linguagem auxiliar, qualquer texto após um `#` até ao fim da linha é considereado comentário

```
# Adds a and b and stores result in variable c
c = a + b;
```

A primeira linha não é executada.

3.5 Controlo de fluxo

Controlo de fluxo enriquece muito o programa, dado que permite que certas instruções sejam executadas mediante certas condições.

Em linguagens comuns há um tipo lógico, no entanto em quiz usa-se o tipo `num`, onde 1 é verdadeiro e qualquer outro valor é falso.

Caso a expressão a ser avaliada como lógica não seja um número será sempre falsa.

3.5.1 if, else if, else

Esta estrutura permite executar código uma vez se uma condição se verificar.

Na sua forma mais simples pode ser apenas usado um `if`

```
if (a == 1) {
    print "a is equal to 1";
    # ...
}
```

A instrução entre `()` a seguir ao `if` é avaliada como expressão lógica, se fôr verdadeira, o código dentro das `{}` será executado.

Quando há apenas uma instrução dentro do `if`, os `{}` podem ser omitidos

```
if (a == 1)
    print "a is equal to 1";
```

Mas por vezes quer-se ainda executar código caso a expressão não seja verdadeira. Para isso usa-se o bloco opcional `else`.

```
if (a == 1) {
    print "a is equal to 1";
```

```
} else {
    print "a is not equal to 1";
}
```

Há ainda casos onde se quer executar código mediante o valor de uma variável. Quando há várias hipóteses de valor, podem-se usar blocos `else if`, opcionais e de quantidade arbitrária.

```
if (a == 1)
    print "a is equal to 1";
else if (a == 2)
    print "a is equal to 2";
else if (a == 3)
    print "a is equal to 3";
else
    print "a is not 1, 2 or 3";
```

De notar que a haver um bloco `else`, este tem de ser o último da cadeia.

3.5.2 while e do while

Quando se quer que certo código seja executado enquanto uma condição se verifica deve-se recorrer ao `while` e `do while`.

```
while a < 10 {
    print "a is still less than 10";
    a++;
}
```

Mais uma vez, caso apenas haja uma linha a ser executada no loop, os `{}` podem ser omitidos.

```
while (a++ < 10)
    print "a is still less than 10"
```

O fluxo deste loop é:

1. Avaliar a expressão
2. Se verdadeira:
    1. Executa as instruções
    2. Retorna a **1.**
3. Se falsa:
    1. Sai do loop e continua o fluxo normal do programa

Se for pretendido que a avaliação ocorra depois de as instruções serem corridas (quando se quer que o código corra pelo menos uma vez) pode-se usar o `do while`

```
do
    print "a is still less than 10"
while (a < 10)
```

3.5.3 for

Há 3 instruções: a primeira é corrida uma vez antes de tudo, a segunda antes de cada iteração e a terceira no fim de cada.

```
for (num i = 0; i < 10; i++) {
    print "i is " + i;
}
```

Isto pode ser traduzido num ciclo `while` em:

```
num i = 0;

while (i < 10) {
    print "i is " + i;
    i++;
}
```

A primeira e terceira instruções são opcionais. QUando omitidas, o ciclo passa a ser um `while` simples

```
for (; i < 10; )
    print "i is " + i++;

while (i < 10)
    print "i is " + i++;
```

3.5.4 for each

Este loop permite iterar por todos os elementos de qualquer tipo de dados enumerável.

É criada uma variável de iteração que assumirá em cada iteração um valor diferente da estrutura enumerável.

```
for each v in lst {
    print v + " belongs to the list"
}

foreach v in lst
    print v + " belongs to the list"
```

Os tipos de dados enumeráveis são `num`, `list` e `text`.

Ao iterar por uma `list` a variável de iteração assumirá cada um dos valores da lista.

NOTA: listas podem conter dados de tipos diferentes, isto pode requerer alguns cuidados extra em loops sobre listas.

Ao usar um `num`, a variável de iteração assumirá valores numéricos de 0 ao número, se ele for positivo, ou do número a 0 caso contrário.

Ao iterar por um `text` a variável de iteração assumirá o valor de cada caracter na variável.

3.6 Funções

Funções permitem executar sem grande esforço tarefas mais complexas e de baixo nível, como imprimir texto no terminal ou avaliar uma resposta.

Funções têm um nome (o seu identificador) e parâmetros (os dados sobre os quais elas devem operar).

Há ainda o retorno, que é o resultado da função.

Os parâmetros são passados como uma lista, ou seja, as seguintes sintaxes são todas válidas:

```
# A list of questions
list quests = #...

# Calling the traditional way
list t1 = byTheme quests, "t1";

# Passing a list with the parameters
list l1 = quests, "t2";
list t2 = byTheme l1;
```

3.6.1 Lista de funções

A linguagem quiz suporta atualmente as seguintes funções:

nome	parâmetros	descrição
ask	question q	apresenta a pergunta q ao utilizador e aguarda a sua resposta. A resposta é garantidamente válida
ans	question q	retorna a resposta dada pelo utilizador à pergunta q, se a deu. Caso ainda não tenha sido respondida retorna uma lista vazia
byID	list db, text id	retorna a pergunta na lista db cujo ID é id, ou uma lista vazia caso contrário
byTheme	list db, text theme	retorna uma lista com todas as perguntas de db com o tema theme. A lista pode ser vazia se nenhuma for encontrada.
shuffle	list l	faz uma permutação aleatória na lista l. A lista é embaralhada, não é retornada uma cópia embaralhada.

nome	parâmetros	descrição
validate	question q	avalia a resposta dada pelo utilizador a q com o avaliador da pergunta. Se algum não existir a cotação é 0. Também retorna esse valor
print	text t	mostra no terminal o texto t

3.7 Palavras reservadas

Em quiz as palavras reservadas são os nomes das funções, as palavras das estruturas de controlo de fluxo (if, else, do, while, for, each, foreach) e os tipos de dados (list, num, text e question).

4 Exemplos

4.1 Hello World

Pergunta com report simples

4.2 Hello World CQuiz

Questionário com CQuiz