Trabalho #01 – `wc`

`wc`

`wc` é uma das ferramentas GNU da suíte `coreutils`. Ela conta o número de caracteres, palavras ou linhas da string dada na entrada. Ela possui um comportamento que não é completamente intuitivo no tratamento do número de linhas, o qual é descrito no seu manual. Passa acessar este manual utilize o comando

```
$ info wc
```

O trabalho consiste em portar a ferramenta `wc` para quatro linguagens distintas: Assembly x86, Smalltalk, Haskell e Prolog. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do `wc` pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de `wc`.

A cada implementação/linguagem serão atribuídos até 2,5 pontos da nota, mediante a seguinte expressão:

$$N_i = 2,5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #02 – `basename`

`basename`

`basename` é uma das ferramentas GNU da suíte `coreutils`. Ela renomeia qualquer prefixo que caracterize diretórios de um *path* passado como argumento. Opcionalmente ela também pode remover um sufixo indicado. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info basename
```

O trabalho consiste em portar a ferramenta **`basename`** para quatro linguagens distintas: Assembly x86, Smalltalk, Haskell e Prolog. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`basename`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`basename`**.

A cada implementação/linguagem serão atribuídos até 2,5 pontos da nota, mediante a seguinte expressão:

$$N_i = 2,5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #03 – `md5sum`

`md5sum`

`md5sum` é uma das ferramentas GNU da suíte `coreutils`. Ela computa ou verifica o MD5, algoritmo de *hash* de 128 *bits*. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info md5sum
```

O trabalho consiste em portar a ferramenta **`md5sum`** para duas linguagens distintas: Smalltalk e Haskell. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`md5sum`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`md5sum`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #04 – `md5sum`

`md5sum`

`md5sum` é uma das ferramentas GNU da suíte `coreutils`. Ela computa ou verifica o MD5, algoritmo de *hash* de 128 *bits*. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info md5sum
```

O trabalho consiste em portar a ferramenta **`md5sum`** para duas linguagens distintas: Assemlby x86 e Prolog. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`md5sum`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`md5sum`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #05 – `sum`

`sum`

`sum` é uma das ferramentas GNU da suíte `coreutils`. Ela computa o *checksum*, algoritmo de *hash* de 16 *bits*. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info sum
```

O trabalho consiste em portar a ferramenta **`sum`** para duas linguagens distintas: Smalltalk e Haskell. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`sum`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`sum`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #06 – `sum`

`sum`

`sum` é uma das ferramentas GNU da suíte `coreutils`. Ela computa o *checksum*, algoritmo de *hash* de 16 *bits*. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info sum
```

O trabalho consiste em portar a ferramenta **`sum`** para duas linguagens distintas: Assembly x86 e Prolog. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`sum`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`sum`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #07 – `seq`

`seq`

`seq` é uma das ferramentas GNU da suíte `coreutils`. Ela imprime uma sequência de números na saída padrão. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info seq
```

O trabalho consiste em portar a ferramenta **`seq`** para duas linguagens distintas: Smalltalk e Haskell. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`seq`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`seq`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #08 – `seq`

`seq`

`seq` é uma das ferramentas GNU da suíte `coreutils`. Ela imprime uma sequência de números na saída padrão. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info seq
```

O trabalho consiste em portar a ferramenta **`seq`** para duas linguagens distintas: Assembly x86 e Prolog. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`seq`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`seq`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #09 – `uniq`

`uniq`

`uniq` é uma das ferramentas GNU da suíte `coreutils`. Ela imprime as linhas passadas na entrada, exceto as repetições adjacentes. Opcionalmente pode-se ignorar as linhas que não tem repetições ou as que se repetiram ao menos uma vez. documentação desta ferramenta pode ser obtida por meio do comando

```
$ info uniq
```

O trabalho consiste em portar a ferramenta **`uniq`** para duas linguagens distintas: Smalltalk e Haskell. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`uniq`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`uniq`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #10 – `uniq`

`uniq`

`uniq` é uma das ferramentas GNU da suíte `coreutils`. Ela imprime as linhas passadas na entrada, exceto as repetições adjacentes. Opcionalmente pode-se ignorar as linhas que não tem repetições ou as que se repetiram ao menos uma vez. documentação desta ferramenta pode ser obtida por meio do comando

```
$ info uniq
```

O trabalho consiste em portar a ferramenta **`uniq`** para duas linguagens distintas: Assembly x86 e Prolog. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`uniq`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`uniq`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #11 – `base64`

`base64`

`base64` é uma das ferramentas GNU da suíte `coreutils`. Ela codifica a entrada em base 64. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info base64
```

O trabalho consiste em portar a ferramenta **`base64`** para duas linguagens distintas: Smalltalk e Haskell. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`base64`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`base64`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #12 – `base64`

`base64`

`base64` é uma das ferramentas GNU da suíte `coreutils`. Ela codifica a entrada em base 64. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info base64
```

O trabalho consiste em portar a ferramenta **`base64`** para duas linguagens distintas: Assembly x86 e Prolog. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`base64`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`base64`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #13 – `cksum`

`cksum`

`cksum` é uma das ferramentas GNU da suíte `coreutils`. Ela computa o CRC da entrada. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info cksum
```

O trabalho consiste em portar a ferramenta **`cksum`** para duas linguagens distintas: Smalltalk e Haskell. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`cksum`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`cksum`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #14 – `cksum`

`cksum`

`cksum` é uma das ferramentas GNU da suíte `coreutils`. Ela computa o CRC da entrada. A documentação desta ferramenta pode ser obtida por meio do comando

```
$ info cksum
```

O trabalho consiste em portar a ferramenta **`cksum`** para duas linguagens distintas: Assembly x86 e Prolog. Estas implementações não devem se valer de bibliotecas externas, se restringindo às instalações destas linguagens descritas nas instruções gerais dos trabalhos.

A implementação do **`cksum`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 8.30 de **`cksum`**.

A cada implementação/linguagem serão atribuídos até 5 pontos da nota, mediante a seguinte expressão:

$$N_i = 5 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação feita na i-ésima linguagem não compile, ou resulte num erro de execução, a nota N_i será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.

Trabalho #15 – `cowsay`

`cowsay`

`cowsay` é uma ferramentas que gera uma representação ASCII de uma vaca, que diz a mensagem passada como parâmetro. Para acessar este manual utilize o comando

```
$ info cowsay
```

O trabalho consiste em portar a ferramenta **`cowsay`** para Haskell. Esta implementação não deve se valer de bibliotecas externas, se restringindo à instalação desta linguagem descrita nas instruções gerais dos trabalhos.

A implementação do **`cowsay`** pode ser obtida neste link.

Avaliação

As implementações serão avaliadas por meio de uma suíte de T testes unitários, os quais exercitarão chamadas dos aplicativos com diferentes entradas e combinações de *flags*. As saídas dos aplicativos serão confrontadas por àquelas produzidas pela versão 1.12 de **`cowsay`**.

A implementação corresponde aos 10 pontos da nota, mediante a seguinte expressão:

$$N = 10 \times \frac{A}{T},$$

onde A é o número de testes bem sucedidos e N_i é a nota atribuída à i-ésima linguagem.

Caso a implementação não compile, ou resulte num erro de execução, a nota N será zero. O mesmo acontecerá se o código da implementação não for entregue no prazo estipulado.