

PROGRAMAÇÃO FUNCIONAL

Códigos: GSI004 e GBC033

Cursos: Sistemas de Informação e Ciência da Computação – Sta. Mônica

Professora: MSc. Júlia Tannús de Souza

Período Letivo: 2021/2 – junho a agosto de 2022

Aula Prática 1 - Roteiro

Para “rodar” um programa que você escreveu, você precisará de um compilador ou de um interpretador. Um **compilador** é um programa de sistema que traduz um programa descrito em uma linguagem de alto nível para um programa equivalente em código de máquina para um processador (Fig. 1), gerando um arquivo executável (.exe).



Fig.1 – O que faz um compilador.

Fonte: <https://blog.betrybe.com/tecnologia/compilador-o-que-e/>

Por sua vez, um **interpretador** faz essa tradução linha a linha, em tempo real, sem gerar um arquivo. Ele serve para você testar o seu código antes de compilar a versão final.

Uma forma de você interpretar/compilar seu código sem instalar nada é usando o <https://replit.com/new/haskell>, de forma *online*.

Outra forma é instalar em seu computador, para usar *offline*. Há vários compiladores para a linguagem Haskell. Usaremos o GHC (*Glasgow Haskell Compiler*). Em junho de 2022, ele pode ser instalado da seguinte forma:

- 1) Acesse o site: <https://www.haskell.org/ghcup/>.

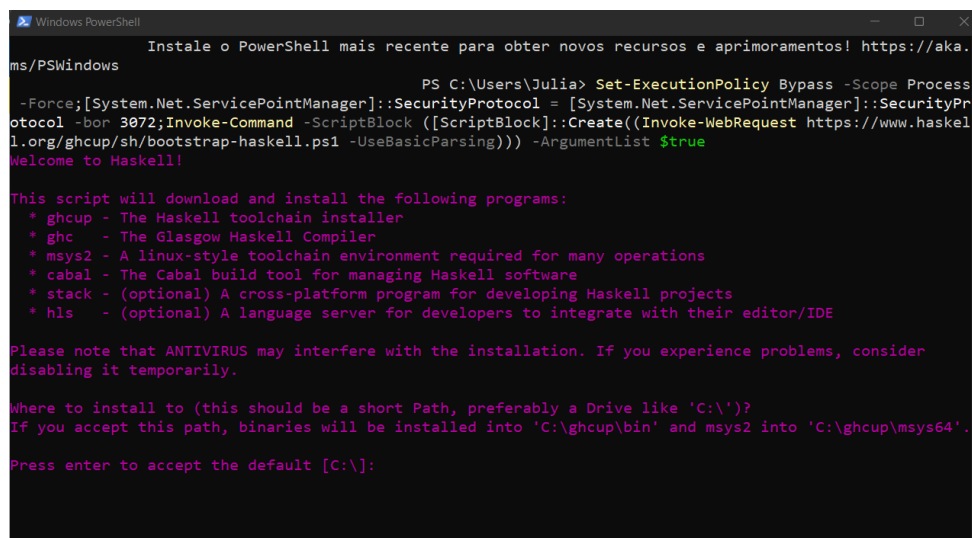


- 2) Clique no botão para copiar o comando.

- 3) Abra o Windows PowerShell (pesquisando “PowerShell” na barra de pesquisa).

4) Cole o comando no Windows PowerShell e aperte “Enter”.

5) Agora, o compilador e o interpretador Haskell serão instalados no seu computador. Esse processo pode demorar vários minutos. Aguarde. O *prompt* te fará algumas perguntas durante o processo. Aperte “Enter” após as perguntas para proceder com a instalação *default* (Fig. 2).



```
Windows PowerShell
Instale o PowerShell mais recente para obter novos recursos e aprimoramentos! https://aka.ms/PSWindows

PS C:\Users\Julia> Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; Invoke-Command -ScriptBlock ([ScriptBlock]::Create((Invoke-WebRequest https://www.haskell.org/ghcup/sh/bootstrap-haskell.ps1 -UseBasicParsing))) -ArgumentList $true
Welcome to Haskell!

This script will download and install the following programs:
* ghcup - The Haskell toolchain installer
* ghc   - The Glasgow Haskell Compiler
* msys2 - A linux-style toolchain environment required for many operations
* cabal - The Cabal build tool for managing Haskell software
* stack - (optional) A cross-platform program for developing Haskell projects
* hls   - (optional) A language server for developers to integrate with their editor/IDE

Please note that ANTIVIRUS may interfere with the installation. If you experience problems, consider disabling it temporarily.

Where to install to (this should be a short Path, preferably a Drive like 'C:\')?
If you accept this path, binaries will be installed into 'C:\ghcup\bin' and msys2 into 'C:\ghcup\msys64'.
Press enter to accept the default [C:\]:
```

Fig.2 – Instalando o GHC.

6) Ao finalizar o processo, digite “ghci” no Windows PowerShell ou no Prompt de Comando. Caso entre no “Prelude>”, a instalação foi bem sucedida (Fig. 3) .



```
C:\Users\Julia>ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude>
```

Fig.3 – GHC instalado.

Prelude é a biblioteca padrão. É um conjunto de arquivos onde estão definidas várias funções e operadores básicos.

Você pode fazer várias operações no Prelude. Porém, ao escrever programas maiores, é útil ter um **editor de texto**. Dentre vários editores, recomendo o Sublime Text (outros exemplos: Emacs, Vim, Atom, VS Code, Bloco de Notas).

O editor de texto pode reconhecer várias linguagens, e te ajuda a escrever códigos maiores, ao verificar sua sintaxe, destacar com cores diferentes os tipos, nomes de variáveis e funções, fazer sugestões de texto, indentar automaticamente o texto, etc. Siga os passos a seguir para editar um arquivo Haskell:

- 1) Baixe o Sublime Text em: <https://www.sublimetext.com/>.
- 2) Abra o Sublime Text.
- 3) Configure o corretor de texto para verificar Haskell no menu: View > Syntax > Haskell.
- 4) Crie um novo arquivo no menu File > New File.
- 5) Salve esse arquivo com a extensão “.hs” em File > Save as... > nome_do_arquivo.hs
- 6) Pronto, você já pode fazer definições de funções, operações, etc.

Para editar um arquivo já salvo anteriormente, basta você abri-lo novamente com o editor de texto (em File > Open).

Para testar se o seu código funciona, você pode carregar o arquivo no ghci, usando o comando **:load**. Após realizar alterações no arquivo, lembre-se de salvar as alterações antes de testar o código novamente, com Ctrl-S, e recarregar o arquivo no ghci, com o comando **:reload**. Para usar o comando **:load**, você deve especificar o caminho onde está salvo seu arquivo, da seguinte forma:

```
Prelude> :load "C:\\UFU\\PF\\Aulas\\aula01.hs"
```

Comando	Significado
:load	Carrega um arquivo
:reload	Recarrega o arquivo
:edit	Edita o arquivo atual
:type <expr>	Mostra o tipo de uma expressão
:help	Obtém ajuda
:quit	Termina a sessão
:browse <module>	Mostra todas as funções de um módulo

- 1) Experimente testar os comandos acima no ghci. Observe os resultados.

Outra forma de carregar um arquivo é você associar a extensão .hs com o ghci.exe. Você pode fazer isso clicando com o botão direito em um arquivo .hs, depois clicar em “Abrir com...” > Escolher outro aplicativo > ghci.exe. Geralmente, o ghci.exe fica na pasta C:\ghcup\bin. Dessa forma, para abrir um arquivo .hs com o ghci, bastará você dar dois cliques no mesmo.

Ao finalizar de escrever e testar um programa, você pode compilá-lo, construindo sua versão final executável (.exe). Para isso, siga os seguintes passos:

1) Crie e salve um arquivo .hs com o seguinte texto:

```
main :: IO ()
main = do
  putStrLn "Hello, World!"
  x <- getChar
  putStrLn ("")
```

2) Abra o PowerShell.

3) Digite o comando ghc seguido do caminho do seu arquivo .hs. Pressione Enter.

4) Será gerado um executável na mesma pasta do arquivo fonte. Execute-o.

Obs: Para compilar o seu arquivo, ele deve ter definida uma função *Main*.

Exercícios propostos (agradecimento: Profª Maria Adriana)

1. Teste as expressões abaixo e explique cada resultado:

Prelude> 2 + 15

Prelude> 49 * 100

Prelude> 1892 - 1472

Prelude> 5 / 2

Prelude> (50 * 100) - 4999

Prelude> 50 * 100 - 4999

Prelude> 50 * (100 - 4999)

Prelude> -1 + 5

Prelude> (-1) + 5

Prelude> 5 + "cinco"

Prelude> 5 + True

Prelude> succ 8

Prelude> min 9 10

Prelude> min 3.4 3.2

Prelude> max 100 101

Prelude> succ -4

```
Prelude> succ (-4)
Prelude> succ 9 + max 5 4 + 1
Prelude> (succ 9) + (max 5 4) + 1
Prelude> 5 == 5
Prelude> 1 == 0
Prelude> 5 /= 5
Prelude> 5 /= 4
Prelude> "hello" == "hello"
Prelude> 'a' == 'b'
Prelude> "elegante" < "elefante"
Prelude> True && False
Prelude> True && True
Prelude> False || True
Prelude> not False
Prelude> not (True && True)
Prelude> if (3 > 4) then "maior" else "menor"
-- Os proximos comandos devem ser executados em sequência
Prelude> let i = -- complete aqui com sua idade
Prelude> i
Prelude> if (i < 18) then "menor de idade" else "maior de idade"
```

2. Crie um arquivo denominado lab01.hs. Nesse arquivo você implementará todas as funções desta aula prática.
3. Escreva uma função para calcular o dobro de um número.
4. Escreva uma função para quadruplicar um número, utilizando a função definida no item anterior.
5. Escreva uma função soma2 x y que realiza a soma de dois números x e y.
6. Com base na função soma2, implemente a função soma4 que calcula a soma de quatro números.
7. Implemente a seguinte função:

misterio x y z w = soma2 (soma2 x y) (soma2 z w)

Compare a saída da função misterio com a saída da função soma4 e entenda o que aconteceu. Dica: Execute um caso passo-a-passo.

8. Implemente a função hipotenusa que, a partir dos dois catetos de um triângulo retângulo, fornece o valor da hipotenusa desse triângulo. Dica: utilize a função sqrt, que calcula a raiz quadrada de um número.