Functional Programming Concept with Haskell

Dr. Samit Bhattacharya

Computer Science & Engineering

Indian Institute of Technology Guwahati

Recap

- Programming paradigms – broadly two types
  - Imperative
  - Declarative

- Imperative
  - Main concern – HOW to do things

- Declarative
  - Main concern – WHAT to do, not HOW to do things

Recap

- ➢ Declarative paradigm
  - Logic programming – we have already seen it (Prolog); centers around RELATIONS
  - Functional programming – going to discuss in this tutorial

Functional Programming

- A programming paradigm which models computations as the evaluation of expressions
  - Key Idea - computation as 'evaluation of mathematical functions'
- It does not contain any assignment statements
- Idea originated from the Lambda Calculus formalism

Functional Programming

- Languages that follow functional programming paradigm
  - Haskell
  - LISP
  - Python
  - Erlang
  - Racket
  - F#
  - Clojure

  - JavaScript (more familiar) – a multi paradigm language (procedural, OO and FP)

Functional Programming

- Languages that follow functional programming paradigm
  - Haskell
  - LISP
  - Python
  - Erlang
  - Racket
  - F#
  - Clojure

Haskell

- Standardized *purely* functional programming language

- Named after logician and mathematician Haskell Brooks Curry

- History
  - First version ("Haskell 1.0") introduced in 1990
  - Latest standard is "Haskell 2010"

Application

- Some examples
  - Darcs is a version control system written in Haskell, with several innovative features, such as more precise control of the patches to be applied
  - Xmonad is a window manager for the X Window System, written fully in Haskell
  - Facebook implements its anti-spam programs in Haskell, as open-source software

Haskell - Features

- Purely functional
- Statically typed
- Type inference
- Lazy
- Concurrent
- Packages

Purely Functional

- Every function in Haskell is a function in the mathematical sense (i.e., "pure")
  - The pure function returns the same output every time for the same input
  - In a *pure* functional language, you can't do anything that has a *side effect*
- If evaluating an expression changes some internal state, it is called a *side effect* since that may give a different result if the same expression is evaluated again

Purely Functional

```
function impure(str: string){
        str = str + "Post";
        print(str);
        return(str);
}
```

State of function
gets changed

Ex. Impure function

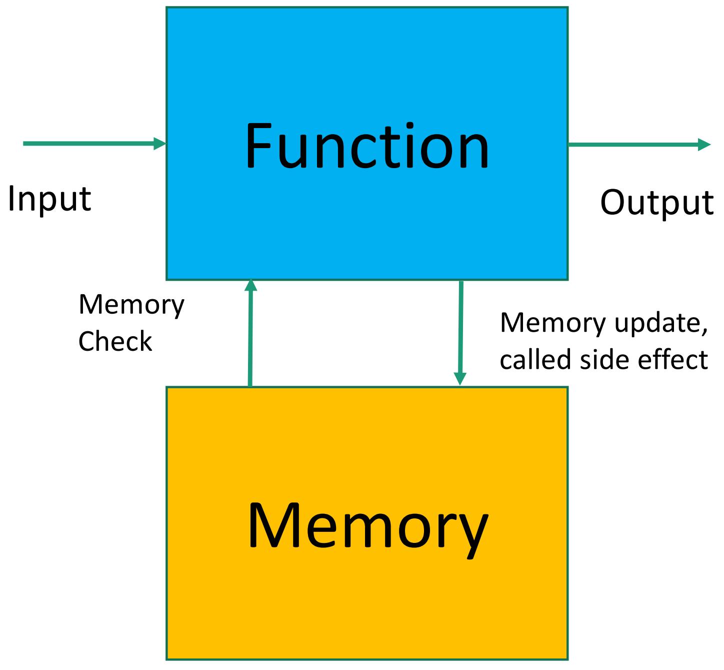
```
function impure(str: string){
            return(str + "Post");
}
```

No change in state $\boxed{\text{Immutable}}$

Ex. Pure function

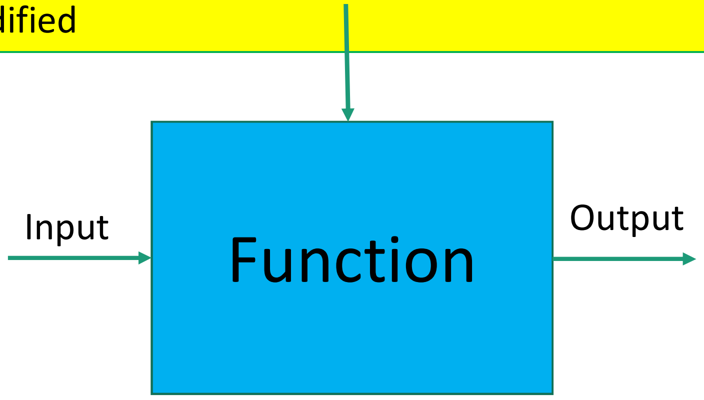
Purely Functional

- Purely functional



In other programming languages

It doesn’t have any side effect as state is not modified

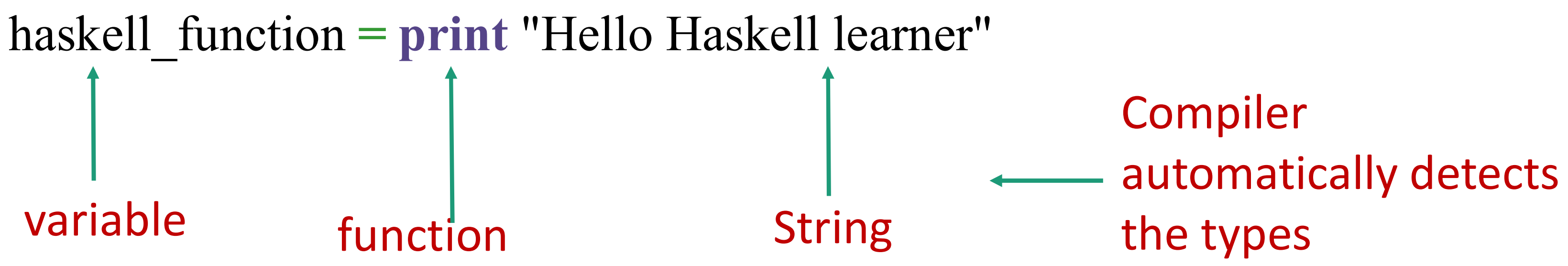


In Haskell

Haskell is stateless

Statically Typed

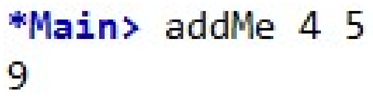
- Every expression in Haskell has a type which is determined at compile time
  - The compiler knows which piece of code is a number, which is a string and so on



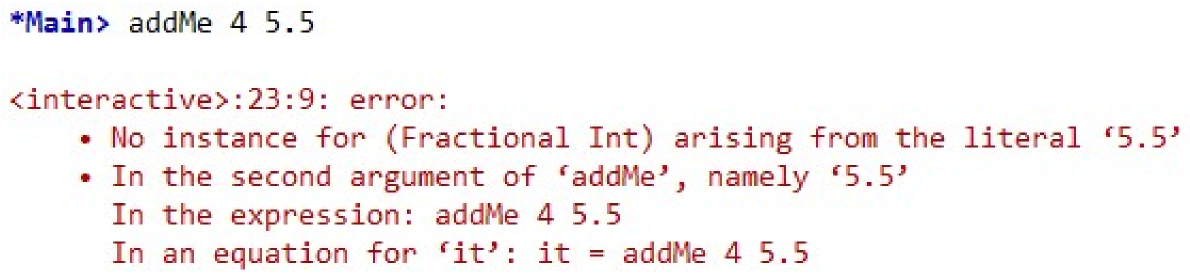
```
haskell_function = print "Hello Haskell learner"
```

Statically Typed

- All the types composed together by function application have to match up. If they don't, the program will be rejected by the compiler
  - Ex. addMe :: Int -> Int -> Int $\longleftarrow$ type signature or function declaration
  - addMe x y = x+y $\longleftarrow$ function definition



```
*Main> addMe 4 5
9
```



```
*Main> addMe 4 5.5

<interactive>:23:9: error:
    • No instance for (Fractional Int) arising from the literal ‘5.5’
    • In the second argument of ‘addMe’, namely ‘5.5’
      In the expression: addMe 4 5.5
      In an equation for ‘it’: it = addMe 4 5.5
```

Type Inference

- You don't have to explicitly label every piece of code with a type because the type system can intelligently figure out a lot about it;
  - Eg. if you say a = 5 + 4

Haskell automatically infer that ‘a’ is a number

- However, you can write out types if you choose, or ask the compiler to write them for you for handy documentation

Lazy

- Nothing's evaluated unless it has to be

Ex. f x y = x + 2

Function call : f 5 (29^35792)

Both the x and y values are evaluated and passed to function f

Non lazy languages like C or Java

Haskell pass the arguments value as it is without doing any actual computation of 29^35792

Haskell

Saves on CPU usage and user's time!

Concurrency

- Functional programming, by its nature (lack of side effect), is suitable for parallelism

- Concurrency in Haskell is mostly done with Haskell threads

- The Glasgow Haskell Compiler (GHC), comes with concurrency library containing a number of useful concurrency primitives and abstractions technique called Software Transactional Memory (STM)

- STM is an alternative to the lock based synchronization, whose basic objective is to evaluate a set of expression in isolated manner

Packages

- Open source contribution to Haskell is very active with a wide range of packages available on the public package servers
- There are 6,954 packages freely available; for instances

bytestring	Binary data	base	Prelude, IO, threads
network	Networking	text	Unicode text
parsec	Parser library	directory	File/directory
hspec	RSpec-like tests	attoparsec	Fast parser
monad-logger	Logging	persistent	Database ORM
template-haskell	Meta-programming	tar	Tar archives

Lets Start

Lets try to understand basic program of Haskell

Run your First Haskell Program

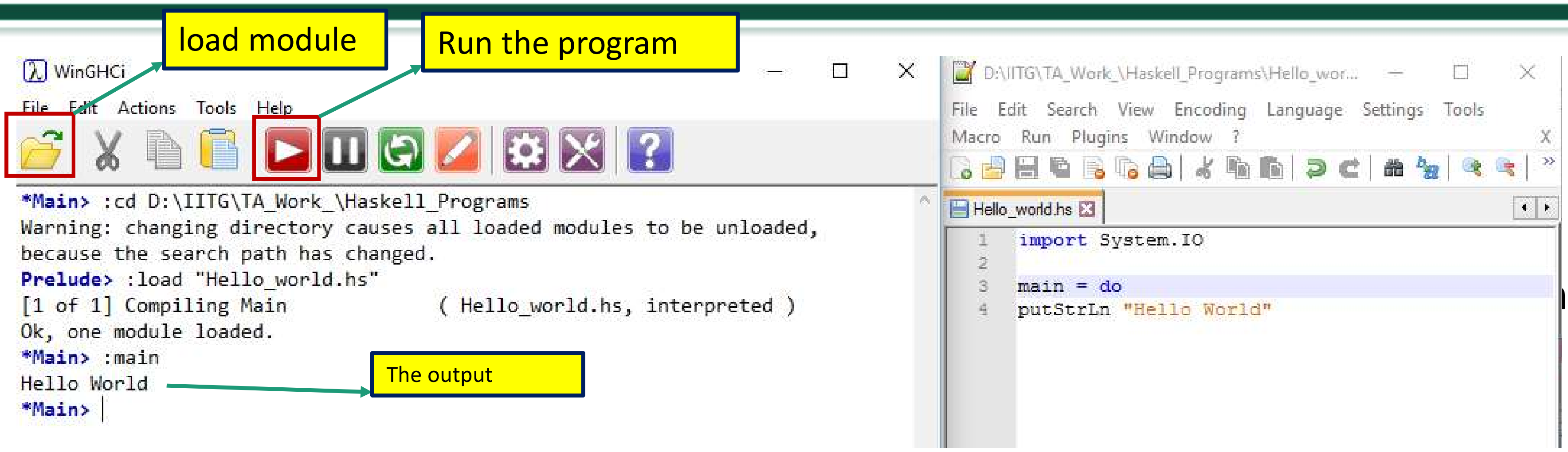
- Download and Install Haskell
  - Download link https://www.haskell.org/downloads

- File extension .hs
  - Open text editor, write your program, save your program with .hs extension (e.g., haskell-tutorial.hs)

Run your First Haskell Program

- Compilation and Run
  - For Windows OS
    - ✓ Open WinGHCi from start menu
    - ✓ Load your program (File -> Load..)
    - ✓ Run the function you want

Run your First Haskell Program


load module
Run the program
The output
WinGHCi
File Edit Actions Tools Help
D:\IITG\TA_Work_\Haskell_Programs\Hello_wor...
File Edit Search View Encoding Language Settings Tools
Macro Run Plugins Window ?
Hello_world.hs

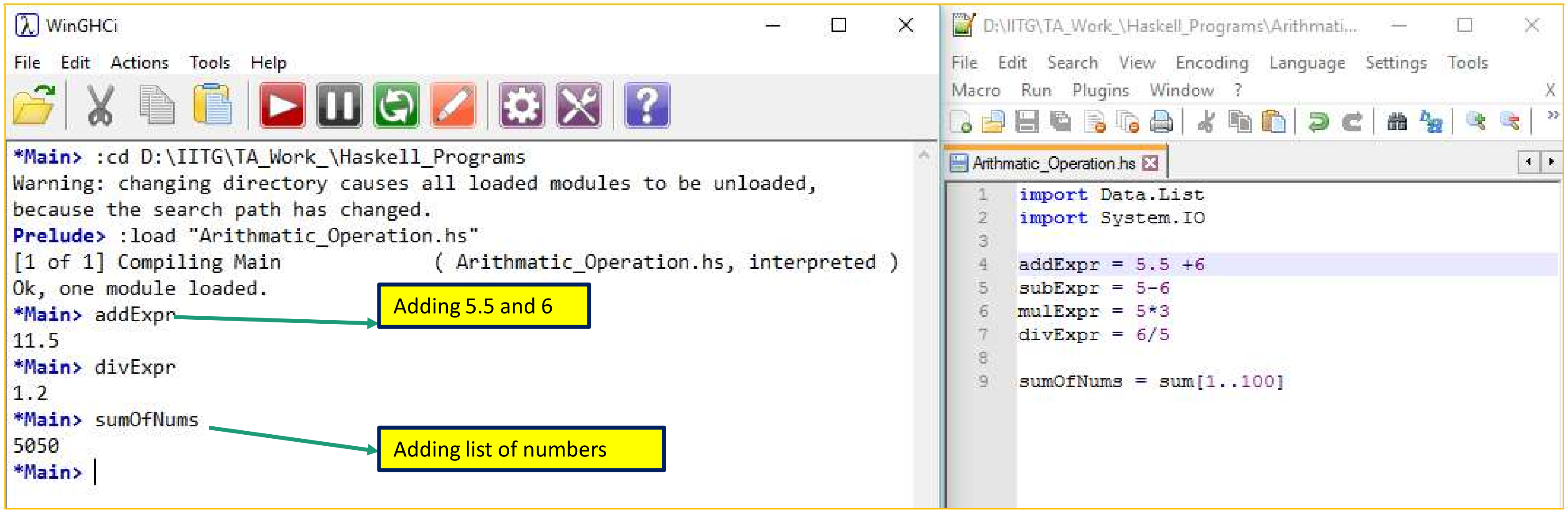
```
*Main> :cd D:\IITG\TA_Work_\Haskell_Programs
Warning: changing directory causes all loaded modules to be unloaded,
because the search path has changed.
Prelude> :load "Hello_world.hs"
[1 of 1] Compiling Main             ( Hello_world.hs, interpreted )
Ok, one module loaded.
*Main> :main
Hello World
*Main>
```

```haskell
import System.IO

main = do
putStrLn "Hello World"
```

- For Windows OS
  - Open WinGHCi from start menu
  - Load your program (File -> Load..)
  - Run the function you want

Haskell Program for arithmetic operation



WinGHCi
File Edit Actions Tools Help
Adding 5.5 and 6
Adding list of numbers
D:\IITG\TA_Work_\Haskell_Programs\Arithmati...
File Edit Search View Encoding Language Settings Tools
Macro Run Plugins Window ?
Arithmatic_Operation.hs

```
*Main> :cd D:\IITG\TA_Work_\Haskell_Programs
Warning: changing directory causes all loaded modules to be unloaded,
because the search path has changed.
Prelude> :load "Arithmatic_Operation.hs"
[1 of 1] Compiling Main             ( Arithmatic_Operation.hs, interpreted )
Ok, one module loaded.
*Main> addExpr
11.5
*Main> divExpr
1.2
*Main> sumOfNums
5050
*Main>
```

```haskell
import Data.List
import System.IO

addExpr = 5.5 +6
subExpr = 5-6
mulExpr = 5*3
divExpr = 6/5

sumOfNums = sum[1..100]
```

Key Points

- Functional programming is stateless

- Functional programs contain no assignment statements, so variables, once given a value, never change

- A function call can have no effect other than to compute its result

YOU MAY EXPLORE

http://www.learnyouahaskell.com

FOR MORE DETAIL

Announcements

- There will be one more tutorial on Haskell (by TA) – details will be intimated later
- Assignment 3: to be uploaded by Sunday (28th Oct)
  - Submission deadline will be 30th November. However, you may complete and get it evaluated early, if you wish
- Unit test 3 will take place in the last week before the end sem
- We will also have the viva during that week (likely to spread over the whole week, considering the large number) – details to be announced shortly [try not to miss it with excuses!]