

# CSE216 Programming Abstractions

## Complex Number

YoungMin Kwon

# Using AWS (local machine)

- From your local machine
  - Go to the directory where sshaws, sftpaws are.
  - Create a directory: `mkdir <name of a directory>`
  - Change directory: `cd <name of the directory>`
    - .. is the name of the parent directory
    - E.g.) `cd ..`
  - Run `sshaws` to run a shell at AWS
  - Run `sftpaws` to upload/download files to/from AWS

# Using AWS (local machine)

- Windows cmd example

```
Command Prompt
D:\>mkdir reci
D:\>cd reci
D:\reci>code complex.ml
D:\reci>cd ..
D:\>sshaws
D:\>start putty -ssh -X -i "C:\Users\youngmin.kwon\
D:\>ftpaws
D:\>start psftp -i "C:\Users\youngmin.kwon\Desktop\
D:\>
```

# Using AWS (ssh)

- In your ssh shell
  - Create a directory: `mkdir <name of a directory>`
  - Change directory: `cd <name of the directory>`
  - To see the contents of the directory: `ls`
  - To run ocaml: `ocaml`

```
ubuntu@ip-172-31-44-244: ~/reci
ubuntu@ip-172-31-44-244:~$
ubuntu@ip-172-31-44-244:~$ mkdir reci
ubuntu@ip-172-31-44-244:~$ cd reci
ubuntu@ip-172-31-44-244:~/reci$ ls
complex.ml
ubuntu@ip-172-31-44-244:~/reci$ ocaml complex.ml
hello complex
ubuntu@ip-172-31-44-244:~/reci$
```

# Using AWS (sftp)

- In your sftp shell
  - Change AWS directory: **cd** <name of directory>
  - Change local directory: **lcd** <name of directory>
  - To see the contents of AWS directory: **dir**
  - To see the contents of local directory: **!dir**
    - To run any local command use **!command**
  - To upload file: **put** <file name> or **mput** \*.ml
  - To download file: **get** <file name> or **mget** \*.ml

# Using AWS (sftp)

- sftp

```
C:\Users\youngmin.kwon\Desktop\Tools\psftp.exe
Using username "ubuntu".
Remote working directory is /home/ubuntu
psftp> cd reci
Remote directory is now /home/ubuntu/reci
psftp> lcd reci
New local directory is D:\reci
psftp> dir
Listing directory /home/ubuntu/reci
drwxrwxr-x  2 ubuntu  ubuntu   4096 Mar 14 01:43 .
drwxr-xr-x  8 ubuntu  ubuntu   4096 Mar 14 01:39 ..
psftp> !dir
Volume in drive D is DATA
Volume Serial Number is 80E9-55BA

Directory of D:\reci

03/14/2022  10:39 AM  <DIR>      .
03/14/2022  10:39 AM  <DIR>      ..
03/14/2022  10:42 AM                31 complex.ml
                1 File(s)                31 bytes
                2 Dir(s)  362,399,760,384 bytes free
psftp> mput *.ml
local:complex.ml => remote:/home/ubuntu/reci/complex.ml
psftp> mget *.ml
remote:/home/ubuntu/reci/complex.ml => local:complex.ml
psftp> _
```

# Using Functions Like Objects

- Constructor
  - Takes **fields** as parameters
  - Return a function that takes a **select** parameter
- Accessor
  - Depending on the **select** value, return different values

# Using Functions Like Objects

```
(*pair.ml  
  In the toplevel run #use "pair.ml";;  
*)
```

```
(*pair takes f and s as its fields  
  depending on sel, return different values  
*)
```

```
let pair f s =  
  fun sel -> if      sel = "first"  then f  
             else if sel = "second" then s  
             else assert false
```

```
(*making an object*)
```

```
let a = pair 1 2
```

```
(*accessing its fields*)
```

```
let _ = a "first"  
let _ = a "second"
```



# Complex Number

- Constructor
  - `complex r i`: takes `real` and `imaginary` parts and returns a `function`
- Accessor
  - The `function` takes a parameter `sel` and returns its real, imag., mag., and ang. depending on `sel`.
  - i.e. if `c = complex r i`, then `c sel` returns
    - `r` if `sel` is `"real"`
    - `i` if `sel` is `"imag"`
    - `sqrt (r*r + i*i)` if `sel` is `"mag"`
    - `atan2 i r` if `sel` is `"ang"`

```
(*constructor and accessor*)
(*sel is one of "real", "imag", "mag", and "ang"*)
let complex r i = (*TODO: implement this function*)
  fun sel ->
```

```
(*test*)
let equ a b =
  let eps = 1e-10 in
  let abs x = if x < 0. then -. x else x in
  abs (a -. b) < eps
```

```
let a = complex 2. 3.
let b = complex 1. 1.
```

```
let test () =
  let ( == ) = equ in
  assert (a "real" == 2.      && a "imag" == 3.);
  assert (b "mag" == sqrt 2. && b "ang" == atan2 1. 1.);
  ()
let _ = test ()
```

```
(*
```

# Arithmetic Operations

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

$$(a + bi) - (c + di) = (a - c) + (b - d)i$$

$$(a + bi) * (c + di) = (ac - bd) + (ad + bc)i$$

$$\begin{aligned}(a + bi) \div (c + di) &= \frac{(a + bi) * (\overline{c + di})}{(c + di) * (\overline{c + di})} \\ &= \frac{ac + bd}{c^2 + d^2} + \frac{bc - ad}{c^2 + d^2}i\end{aligned}$$

$$\overline{(a + bi)} = (a - bi)$$

```

(*arithmetic operations*)
(*opr is one of "add", "sub", "mul", and "div"*)
let rec arith opr a b =
  (a "real", a "imag", b "real", b "imag") |> fun (ra, ia, rb, ib) ->
    if      opr = "add" then (*TODO: implement this part*)

    else if opr = "sub" then (*TODO: implement this part*)

    else if opr = "mul" then (*TODO: implement this part*)

    else if opr = "div" then (*TODO: implement this part*)
      let c = complex rb (-. ib) in (*conjugate of b*)
      let (rnum, inum) = arith "mul" a c
          |> fun num -> (num "real", num "imag") in
      let rden
          = arith "mul" b c
          |> fun den -> den "real" in

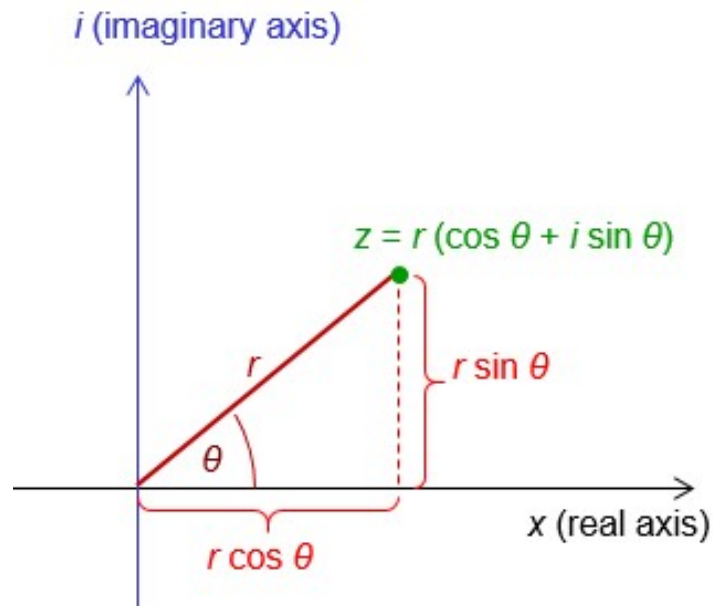
    else assert false

```

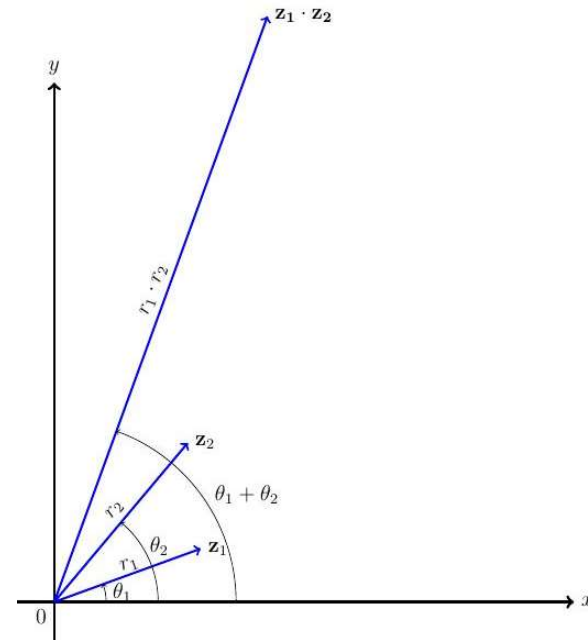
```
let test () =  
  let ( + ) = arith "add" in  
  let ( - ) = arith "sub" in  
  let ( * ) = arith "mul" in  
  let ( / ) = arith "div" in  
  let ( == ) = equ in  
  
  a + b |> fun c -> assert (c "real" == 3. && c "imag" == 4.);  
  a - b |> fun c -> assert (c "real" == 1. && c "imag" == 2.);  
  a * b |> fun c -> assert (c "real" == -1. && c "imag" == 5.);  
  a / b |> fun c -> assert (c "real" == 2.5 && c "imag" == 0.5);  
  ()  
  
let _ = test ()
```

# Polar Form

- Multiplication in Polar form



$$r \angle \theta = r \cdot \cos \theta + i \cdot r \cdot \sin \theta$$



$$r_1 \angle \theta_1 \cdot r_2 \angle \theta_2 = (r_1 \cdot r_2) \angle (\theta_1 + \theta_2)$$
$$r_1 \angle \theta_1 / r_2 \angle \theta_2 = (r_1 / r_2) \angle (\theta_1 - \theta_2)$$

```
(*ploar form*)
(*sel is one of "real", "imag", "mag", and "ang"*)
let polar m a =
  fun sel -> (*TODO: implement this function*)
```

```
let test () =
  let ( == ) = equ in
  let c = polar (a "mag") (a "ang") in
  assert (c "mag" == a "mag" && c "ang" == a "ang");
  assert (c "real" == 2. && c "imag" == 3.);
  ()
```

```
let _ = test ()
```

```
(*arithmetic operations*)
(*opr is one of "add", "sub", "mul", and "div"*)
let arith2 opr a b =
  (a "mag", a "ang", b "mag", b "ang") |> fun (ma, aa, mb, ab) ->
    (*TODO: implement this function*)
```

```
let test () =
  let ( + ) = arith2 "add" in
  let ( - ) = arith2 "sub" in
  let ( * ) = arith2 "mul" in
  let ( / ) = arith2 "div" in
  let ( == ) = equ in
  a + b |> fun c -> assert (c "real" == 3. && c "imag" == 4.);
  a - b |> fun c -> assert (c "real" == 1. && c "imag" == 2.);
  a * b |> fun c -> assert (c "real" == -1. && c "imag" == 5.);
  a / b |> fun c -> assert (c "real" == 2.5 && c "imag" == 0.5);
  ()
let _ = test ()
```