Lec 7: List

CS220: Programming Principles

Sang Kil Cha

List

Motivation

Can we represent a sequence of values whose size is unknown at compile time?

List

List is a finite sequence of values.

Extremely useful data type for functional languages!

List by Examples

Example Lists.

```
[] // Empty list
[1; 2; 3] // A list of three integers.
["a"; "b"; "c"; "d"] // A list of four strings.
[1; 2; "abc"] // ?
```

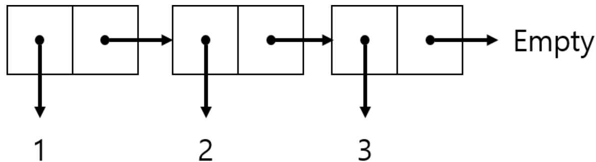
Cons

1. ***Cons*** constructs a value, which is often referred to as a ***cons cell***, holding a pair of values. We say, "cons x onto y" when we construct a new pair where y is followed by x.
2. The first element of a cons cell is ***car***.
3. The second element of a cons cell is ***cdr***[1].

[1] Why car and cdr? See: `https://en.wikipedia.org/wiki/CAR_and_CDR`

List = a Sequence of Cons Cells

- There is a value that represents an empty list (often called as `nil`).
- A singleton list is a cons cell of a value and an empty list.
- A list with two elements can be constructed by consing two cons cells.
- ...



List Consing Operator (: :)

```
let (::) a b = // Prepend a to the list b.

1 :: [2] // Returns [1; 2]
2 :: [4; 6] // Returns [2; 4; 6]
"abc" :: [] // Returns ["abc"]

1 :: 2 :: 3 :: 4 :: [] // [1; 2; 3; 4]
```

Note: `[1; 2; 3; 4]` is just syntactic sugar for `1 :: 2 :: 3 :: 4 :: []`.

List car and cdr.

```
let car lst =
  match lst with
  | hd :: _ -> hd
  | _ -> failwith "Empty list is given."

let cdr lst =
  match lst with
  | _ :: tl -> tl
  | _ -> failwith "Empty list is given."
```

List Range Expressions

A convenient way to construct lists.

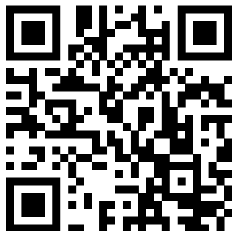
Example list range expressions.

```
[ 1 .. 5 ] // Returns [1; 2; 3; 4; 5]
[ -1 .. 1 ] // Returns [-1; 0; 1]
[ 1 .. 2 .. 5 ] // Returns [1; 3; 5]
[ 1.0 .. 3.2 ] // Returns [1.0; 2.0; 3.0]
[ 1 .. -2 .. 5 ] // ?
```

Attendance Check

Note:

1. This slide appears at random time during the class.
2. This link is only valid for a few minutes.
3. We don't accept late responses.



Prepending/Appending an Element to a List

Can we prepend/append an element to a list? What does it mean to prepend/append an element to a list?

Values are immutable in F#!

In-Class Activity #06

Preparation

We are going to use the same git repository as before. Just in case you don't have it, clone the repository using the following command.

1. Clone the repository to your machine.
   - `git clone https://github.com/KAIST-CS220/CS220-Main.git`
2. Move in to the directory `CS220-Main/Activities`
   - `cd CS220-Main`
   - `cd Activities`

Problem 1

Modify the function `last`, which takes in an arbitrary list as input and returns the last element of the list as output. When the given list is empty, the function will raise an exception with the "`failwith`" function[2].

[2] https://docs.microsoft.com/dotnet/fsharp/language-reference/exception-handling/the-failwith-function

Problem 2

Modify the function `lastButOne` in such a way that it takes in an arbitrary list as input and returns the last but one element of the given list. For example, given `[1; 2; 3; 4]`, the function should return 3. When the input list is empty or is a singleton list, then the function should simply raise an exception with `failwith`.

Write Your Own List Type

Let's Define a Cons Cell Type for Integers

`IntList` can only be either a `nil` or a `cons` of two elements.

> How would you combine two seemingly different types?

Let's Define a Cons Cell Type for Integers (cont'd)

Integer list type.

```
type IntList =
  | Nil
  | Cons of int * IntList
```

Can you now construct a list `[1; 2; 3]` with the newly defined type?

Write basic operators.

```
let empty =
  // ?

let cons elt lst =
  // ?

let car lst =
  // ?

let cdr lst =
  // ?
```

Write infix operator.

Using `cons` for constructing large lists is inconvenient.

Infix operator for consing.

```
let (::) elt lst = ... // Error!
let (++) elt lst = Cons (elt, lst) // First trial.
let (^+^) elt lst = Cons (elt, lst) // Second trial.
```

Read:
https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/symbol-and-operator-reference/index#operator-precedence

Difference between `IntList` vs. `'a list`?

There is a space in the `'a list` type! ☺ Ours can only take integers, whereas `'a list` can take any types.

> Can we make our list implementation ***generic***?

Generic Types

A ***type constructor*** is a function that takes in a type as input and returns a type as output. For example, `list` is a type constructor that takes in `'a` as input. For the `int` type, it returns the `int list` type. We sometimes call a type constructor as ***generic***.

```
/// One way to define a generic type.
type 'a MyList =
  | Nil
  | Cons of 'a * 'a MyList

/// Another way to define a generic type.
type MyList<'T> =
  | Nil
  | Cons of 'T * MyList<'T>
```

In-Class Activity #07

Preparation

We are going to use the same git repository as before. Just in case you don't have it, clone the repository using the following command.

1. Clone the repository to your machine.
   - `git clone https://github.com/KAIST-CS220/CS220-Main.git`
2. Move in to the directory `CS220-Main/Activities`
   - `cd CS220-Main`
   - `cd Activities`

Problem 1: Modify the list type

Convert the `MyList` type into a generic type so that it can represent an arbitrary list of values. Uncomment the code in the file and show that the newly defined type can represent both integer lists and string lists.

Problem 2: Length

Write a function `length` that returns the length of the given list.

Problem 3: Membership Test Function

Write a function `isMember` that takes in an element (of type 'T) and a list (`MyList<'T>`) and returns a boolean indicating whether the element is a member of the given list or not.

1. It is obvious that the function will return false for an empty list.
2. For a given cons cell, we can recursively compare the equality between the given value and the `car` of the list, and then recurse into the `cdr` of the list.

Problem 4: List Append

The infix operator (`@`) joins two lists. For example, `[1; 2] @ [3; 4]` returns `[1; 2; 3; 4]`. Write your own append operator over `MyList<'T>`!

1. `append [1; 2] [3; 4]`

Problem 4: List Append

The infix operator (`@`) joins two lists. For example, `[1; 2] @ [3; 4]` returns `[1; 2; 3; 4]`. Write your own append operator over `MyList<'T>`!

1. `append [1; 2] [3; 4]`
2. `1 :: (append [2] [3; 4])`

Problem 4: List Append

The infix operator (`@`) joins two lists. For example, `[1; 2] @ [3; 4]` returns `[1; 2; 3; 4]`. Write your own append operator over `MyList<'T>`!

1. `append [1; 2] [3; 4]`
2. `1 :: (append [2] [3; 4])`
3. `1 :: 2 :: (append [] [3; 4])`

Problem 4: List Append

The infix operator (`@`) joins two lists. For example, `[1; 2] @ [3; 4]` returns `[1; 2; 3; 4]`. Write your own append operator over `MyList<'T>`!

1. `append [1; 2] [3; 4]`
2. `1 :: (append [2] [3; 4])`
3. `1 :: 2 :: (append [] [3; 4])`
4. `1 :: 2 :: [3; 4]`

Problem 4: List Append

The infix operator (`@`) joins two lists. For example, `[1; 2] @ [3; 4]` returns `[1; 2; 3; 4]`. Write your own append operator over `MyList<'T>`!

1. `append [1; 2] [3; 4]`
2. `1 :: (append [2] [3; 4])`
3. `1 :: 2 :: (append [] [3; 4])`
4. `1 :: 2 :: [3; 4]`
5. `1 :: [2; 3; 4]`

Problem 4: List Append

The infix operator (`@`) joins two lists. For example, `[1; 2] @ [3; 4]` returns `[1; 2; 3; 4]`. Write your own append operator over `MyList<'T>`!

1. `append [1; 2] [3; 4]`
2. `1 :: (append [2] [3; 4])`
3. `1 :: 2 :: (append [] [3; 4])`
4. `1 :: 2 :: [3; 4]`
5. `1 :: [2; 3; 4]`
6. `[1; 2; 3; 4]`

Make it Tail-Recursive!

Previous example was ***not*** tail-recursive. Why?

Can you make them tail-recursive?

Another Problem

The append method we defined is not efficient. Why? Why is it slower than prepending?

Problem 5: List Reverse

Write a function `rev` that takes in a list (`MyList<'T>`) and returns a reversed list.

Conclusion

1. We can create our own list datatype with data abstraction techniques we learned so far.
2. List type is commonly used, and F# has a built-in support for lists.
3. List can be generic.
4. Type constructors is a function that takes in a type and returns a type. It is used to construct generic types such as `List<'T>`.

Question?