Welcome

CS1101S Studio Session Week 8:
Data Structure Design & Stateful Programming

Niu Yunpeng

niuyunpeng@u.nus.edu

October 9, 2018

Overview

1. Data structure design
    - Design principle
    - Examples
2. Stateful programming
    - Mutable data
    - Mutable data structure

Data Structure Design

Three steps to implement a program

In order to solve a problem using a program, you need:

- Think of an appropriate algorithm;
- Design a suitable data structure;
- Do the coding (with good coding style).

Data Structure Design

Thus...

The first three CS modules are:

- CS1101S Programming Methodology
- CS2030 Programming Methodology II
- CS2040 Data Structures and Algorithms

Data Structure Design

Data structure

- In computer science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently.

Algorithm

- In computer science, an algorithm is a self-contained sequence of actions to be performed.

Data Structure Design

Data & information

- Data is the storage of information.
- Two kinds of information: **states** & **procedures**.

Data structure & algorithm

- *To store states efficiently:* use **data structure**;
- *To perform procedures efficiently:* use **algorithm**.

Data Structure Design

Design principle of data structure

- Understand the requirement before doing the actual design;
- Separate the interface from the implementation;
- Compare the advantage and tradeoff;
- Principle of last commitment.

Data Structure Design

1. Understand the requirement

- Your data structure do not need to support everything. You only need to implement what the users really need. Anything else should not be considered.
- There is no “bonus point”.

Data Structure Design

2. Separate the interface from the implementation

- You are free to choose how you are going to implement the data structure, like choice of programming language, abstract data type (ADT), etc.
- However, the interface given to the users should always be the same (and accorded with the convention).
- In other words, users do not need to care about the implementation.

Data Structure Design

3. Compare the advantage and tradeoff

- You may have many choices available to implement the same data structure (or the interface).
- Usually, each of them has its own advantages and tradeoffs.
- You should compare which operation is used more frequently so as to make the final decision.

Data Structure Design

4. Principle of last commitment

- Whenever possible, delay decisions until you have enough information, or until the choice becomes inevitable.
- Make sure your implementation is as generic as possible.
- Try to enhance program re-usability.

Data Structure Design

Examples of data structure so far...

- Coin change
- Symbolic differentiation
- Rational number
- Complex number
- Pair/list/tree
- Set
- ...

Data Structure Design

Common pattern of these examples

- Constructor
- Accessor
- Predicate
- Printer
- ...

Data Structure Design

Advanced data structure & algorithm modules

- CS3230 Design and Analysis of Algorithms
- CS4234 Optimisation Algorithms
- CS5234 Combinatorial and Graph Algorithms
- CS5330 Randomized Algorithms
- CS6234 Advanced Algorithms

Caution

- Could be interesting *(at least to some of you)*
- Need in-depth understanding

Overview

1. Data structure design
   - Design principle
   - Examples
2. Stateful programming
   - Mutable data
   - Mutable data structure

Stateful Programming

Immutable

- A constant holds a value inside it.
- `const x = 1;`
- Cannot hold another value.

Mutable

- A new value can be assigned to the same variable.
- `<variable_name> = <new_value>`
- `let y = 2;`
- To change the value inside `y = 3;`

Stateful Programming

Before Week 8

- Pure functional programming.
- Substitution model.
- Return value do not change if values of arguments are the same.

After Week 8

- Stateful programming.
- Environment model.
- Return value may vary even if values of arguments are the same.

Stateful Programming

The concept of memory allocation

- When we define a variable, the interpreter will allocate a position in memory (random access memory, RAM) randomly so that we can use it any time we want.
- The name is actually the reference to this position in memory.
- Whenever we call the name, the interpreter will just look for the value stored at that position in memory.

Understanding

- A variable is like a **changeable container**.

Stateful Programming

Why can we change the value of a variable?

- Before, when we want to have a new value of a variable, we allocate a new position in memory.
- However, it is not necessary for us to do this at all (because this is in fact a waste of space in memory).
- We can just update the value stored at the original position. When we call that name after that, the interpreter will still look up for the same position and a new value will be found.

Mutable Data Structure

Before today - immutable data structure

- A collection of data into one object.
- Data inside cannot be changed.
- Constructor, accessor, predicate, printer, ...

After today - mutable data structure

- A collection of data into one object.
- Data inside can be changed.
- Constructor, accessor (getter), mutator (setter), predicate, printer, ...

Mutable Data Structure

Mutable pair/list

- `set_head(pr, x)`: set the head of a pair to become `x`;
- `set_tail(pr, y)`: set the tail of a pair to become `y`.

Caution

- Remember identity & equality;
- Remember the concept of memory allocation.

Mutable Data Structure

Things you can do for pair/list

- Re-write some parts of the list library;
  - To make the functions more efficient with respect to time and/or space
- Create a cycle in a list.

Your task today

- Can you write a program to detect the number of cycles in a given list (or return 0 if none)?

Mutable Data Structure

Mutable data structure

- Linked list
- Double-way linked list
- Queue
- Stack
- Table
- ...

Mutable Data Structure

Linked list / double-way linked list 1

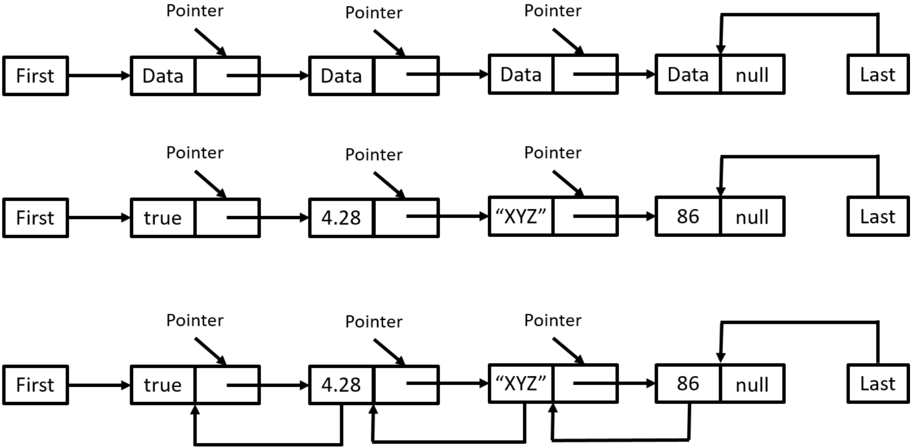
- `make_linked_list()`: create an empty linked list;
- `get_first(lst)`: get the first node of the linked list;
- `get_last(lst)`: get the first node of the linked list;
- `get_next(node)`: get the next node in the linked list;
- `get_prev(node)`: get the last node in the linked list;
- `get_data(node)`: get the data stored in the current node.

Mutable Data Structure

Linked list / double-way linked list 2

- `prepend(lst, x)`: add `x` to the front of the linked list;
- `append(lst, x)`: add `x` to the rear of the linked list;
- `add_before(node, x)`: add `x` before the node;
- `add_after(node, x)`: add `x` after the node;
- `remove_first(lst)`: delete the first node in the linked list;
- `remove_last(lst)`: delete the last node in the linked list;
- `delete(node)`: delete the selected node in the linkd list;
- `empty(lst)`: delete all items in the linked list;
- `is_empty_linked_list(lst)`: check if a linked list is empty.

Mutable Data Structure



Mutable Data Structure

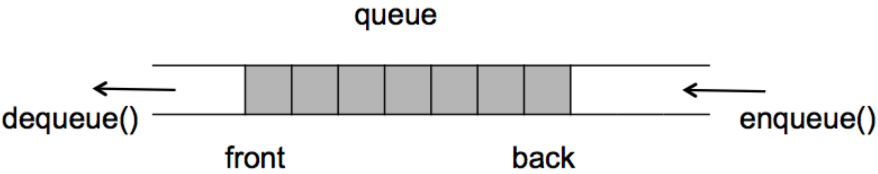
Queue - first in first out (FIFO)

- `make_queue()`: create an empty queue;
- `enqueue(queue, x)`: add `x` to the end of the queue;
- `dequeue(queue)`: delete the first item of the queue;
- `peek(queue)`: retrieve the value the first item of the queue;
- `empty(queue)`: delete all items in the queue;
- `is_empty_queue(queue)`: check if a queue is empty.

Notice

- `dequeue(queue)` and `peek(queue)` will raise an error if the queue is empty.

Mutable Data Structure



Mutable Data Structure

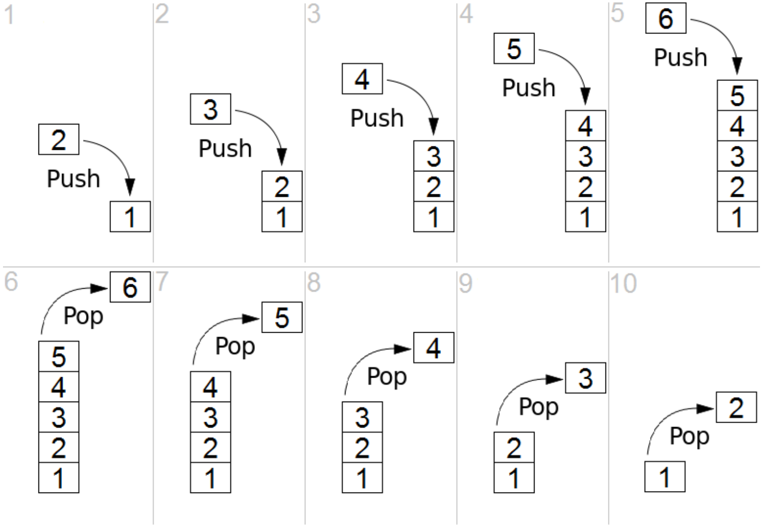
Stack - first in last out (FILO)

- `make_stack()`: create an empty stack;
- `push(stack, x)`: add `x` on the top of the stack;
- `pop(stack)`: delete the first item on the top of the stack;
- `peek(stack)`: retrieve the first value on the top of the stack;
- `empty(stack)`: delete all items in the stack;
- `is_empty_stack(stack)`: check if a stack is empty.

Notice

- `pop(stack)` and `peek(stack)` will raise an error if the stack is empty.

Mutable Data Structure



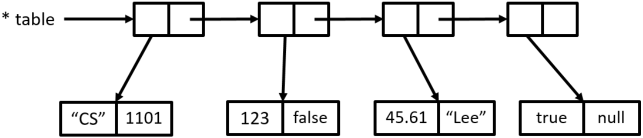
Mutable Data Structure

Table

- `make_table()`: create an empty table;
- `contains(key, table)`: check if the table contains this key;
- `put(key, value, table)`: insert a new entry to the table;
- `lookup(key, table)`: return the value corresponding to the specified key in the table, or `undefined` if the key is not found;
- `empty(table)`: delete all entries in the stack;
- `is_empty_table(table)`: check if a table is empty.

Mutable Data Structure

key	value
“CS”	1101
123	false
45.61	“Lee”
true	null
...	...



Mutable Data Structure

Usage of mutable data structure

- Stack:
    - The interpreter uses stack to implement recursion.
- Table:
    - The binding between names and values in a frame is a table;
    - Later, we will use table to implement memoization.

Robot Mission Demo

Let's do it now.

End

The End

Copyright

This work is mostly an original by *Niu Yunpeng*. It may either directly or indirectly benefit from the previous work of Martin Henz, Cai Deshun. For illustration purposes, some pictures in the public domain are used. Upon request, detailed acknowledgments will be provided.