Go Documentation Server

Package datatype

```
import "GoInAction2/datatype"
```


Overview

Package datatype includes the data structure to be used for the implementing the appointment list, dentist list as well as the available time slot tagged to the dentist.

Index

Package files

appointment.go datetime.go dentist.go

Constants

```
const ApptArrSize = 10
```

Size is set to 10 so each array will has 10 indexes.

```
const ArrSize = 10
```

Variables

```
var (
    ApptHash   *ApptHashTable      // hashtable for storing appointment using appointment id
    DrApptHash *ApptHashTable      // hashtable for storing appointment using dr Id
    ApptYear   = time.Now().Year() // current year
    TimeArr    = [15]string{
        "09:00AM",
        "09:30AM",
        "10:00AM",
        "10:30AM",
        "11:00AM",
        "11:30AM",
        "01:00PM",
        "01:30PM",
        "02:00PM",
        "02:30PM",
        "03:00PM",
        "03:30PM",
        "04:00PM",
        "04:30PM",
        "05:00PM",
    } // appointment timing
```

```
)
```

func **InitAppt**

```
func InitAppt()
```

InitAppt initialized some appointments to the appoinment linkedlist. These appointments are stored in both the Appointment Hash Table as well as Dentist Appointment Hash Table.

func **InitApptHashTable**

```
func InitApptHashTable()
```

InitApptHashTable initialized 2 hash table i) appointment hash table and ii) dentist appointment table. Appoinment hash table is hashed using the appointment id. Dentist Appoinment hash table is hashed using the dr id.

func **InitDateTable**

```
func InitDateTable() *dateHashTable
```

InitDateTable initialized all the available time slots (session 1 to 15) to all the dates from January to December.

func **InitDentist**

```
func InitDentist()
```

InitDentist initialized the available dentist to the data structure.

type **AppointmentList**

AppointmentList is a linkedlink structure where its node will contain appointment information.

```
type AppointmentList struct {
    Head *AppointmentNode // points to the head of appointment linkedlist
}
```

type **AppointmentNode**

ApoointmentNode is a linkedlink node where appointment information is stored.

```
type AppointmentNode struct {
    ApptId  int              // name of dentist
    Details *ApptDetails     // details will contains appointment details,
refer to ApptDetails
    Next    *AppointmentNode // points to the next node
}
```

type **ApptDetails**

ApptDetails is a structure that contains the appointment information.

```
type ApptDetails struct {
    Name        string // name of dentist
    DrId        int    // dr Id
    ApptMonth   int    // month of appointment
    ApptDate    int    // day of appointment
    ApptTime    int    // time of appointment
    PatientUser string // patient username
}
```

func (*ApptDetails) CheckChanges

```
func (d *ApptDetails) CheckChanges(timeSession int, month int, date int, name
string) (bool, error)
```

CheckChanges will check if appointment information has been change. It will takes in time session, month, day as integer input, dentist name as string input and return true if there is any changes to the appointment, else return false.

func (*ApptDetails) Day

```
func (d *ApptDetails) Day() int
```

Day will return the appointment day as integer.

func (*ApptDetails) DentistName

```
func (d *ApptDetails) DentistName() string
```

DentistName will return the name of dentist as string.

func (*ApptDetails) Month

```
func (d *ApptDetails) Month() int
```

Month will return the appointment month as integer.

func (*ApptDetails) Patient

```
func (d *ApptDetails) Patient() string
```

Patient will return the username of patient as string.

func (*ApptDetails) Time

```
func (d *ApptDetails) Time() int
```

Time will return the appointment time session as integer.

type ApptHashTable

ApptHashTable is a structure that will contains all the appointment list. It has an array of size 10 and each index will be a pointer to a linkedlist.

```
type ApptHashTable struct {
    Arr  [ApptArrSize]*AppointmentList // contain a list of pointer pointing
to linked-link of size 10
    Size int                           // number of available dentists
}
```

func (*ApptHashTable) Add

```
func (a *ApptHashTable) Add(name string, drId int, timeSession int, month int,
date int, apptIdChn chan int, isAppt bool, wg *sync.WaitGroup, userName
string)
```

Add adds an appoitnment to the appointment linkedlist. It will take in dr Id, time session, month, dat and appointment Id as integer input, dentist name and patient's username as string input. If isAppt is true means to update appointment using appointment id else means update appointment using drId. Pointer to waitgroup will need to be passed in if a goroutines is executed on this function.

func (*ApptHashTable) Browse

```
func (a *ApptHashTable) Browse(drId int) ([]string, error)
```

Browse wlll browse all the appointments by a particular dentist. It takes in dr id as integer input and return a list of appointments by the chosen dentist.

func (*ApptHashTable) Search

```
func (a *ApptHashTable) Search(apptId int) (bool, *ApptDetails, string, error)
```

Search check if an appointment is available using the appointment Id It takes in appointment id as integer input and return true if the appointment Id exists, else return false.

func (*ApptHashTable) Update

```
func (a *ApptHashTable) Update(name string, drId int, timeSession int, month
int, date int, apptId int, isAppt bool, wg *sync.WaitGroup, userName string)
```

Update will update the appointment details if same dentist is chosen. It will take in dr Id, time session, month, dat and appointment Id as integer input, dentist name and patient's username as string input. If isAppt is true means to update appointment using appointment id else means update appointment using drId. Pointer to waitgroup will need to be passed in if a goroutines is executed on this function.

func (*ApptHashTable) UpdateDiffDr

```
func (a *ApptHashTable) UpdateDiffDr(name string, newDrId int, oldDrId int,
timeSession int, month int, date int, apptId int, isAppt bool, wg
*sync.WaitGroup, userName string)
```

UpdateDiffDr will update the appointment details if different dentist is chosen. It will take in dr Id, time session, month, dat and appointment Id as integer input, dentist name and patient's username as string input. If isAppt is true means to update appointment using appointment id else means update appointment using drId. Pointer to waitgroup will need to be passed in if a goroutines is executed on this function.

type DentistHashTable

DentistHashTable is a structure that will contains all the available dentists. It has an array of size 10 and each index will be a pointer to a linkedlist.

```
type DentistHashTable struct {
    Dentist [ArrSize]*DentistList // contain a list of pointer pointing to
linked-link of size 10
    Size    int                   // number of available dentists
}
```

```
var (
    DentistHash *DentistHashTable                          // DentishHash is a
pointer that points to dentistHashTable structure that will contains all
available dentists.
    NewArr      = &DentistNameList{make([]string, 10), 0} // NewArr is a
pointer that points to a structure thats contains updated list of available
dentists.

)
```

func (*DentistHashTable) AddTimeSlot

```
func (t *DentistHashTable) AddTimeSlot(name string, timeSession int, month
int, date int, wg *sync.WaitGroup) error
```

AddimeSlot adds a time slot from a dentist available time. It takes in timeSession, month and day as integer input, dentist name as string input. The waitgroup pointer is passed in if a goroutines is used to execute this function.

func (*DentistHashTable) Delete

```
func (t *DentistHashTable) Delete(name string) error
```

Delete deletes a dentist name from the linkedlist in the dentistHashTable. It takes in dentist name as string input and update the dentistHashTable.

func (*DentistHashTable) DrId

```
func (t *DentistHashTable) DrId(name string) int
```

DrdId will return the Dr Id as integer given a dentist name.

func (*DentistHashTable) Insert

```
func (t *DentistHashTable) Insert(name string)
```

Insert inserts the dentist name to the linkedlist in the dentistHashTable. It takes in dentist name as string input and update the dentistHashTable.

func (*DentistHashTable) ListAvailTime

```
func (t *DentistHashTable) ListAvailTime(name string, month int, date int)
([]string, error)
```

ListAvailTime lists the available time slot of a dentist given a month and date. It takes in dentist name as string input, month and day as integer input and return a slice of string containing the available time slots.

func (*DentistHashTable) Search

```
func (t *DentistHashTable) Search(name string, timeSession int, month int,
date int) (bool, string, error)
```

Search check if the date and timeslot of a particular dentist is available. It takes in timeSession, month and day as integer input, dentist name as string input and return true if the timeslot of the chosen for the dentist exists, else return false.

func (*DentistHashTable) UpdateTimeSlot

```
func (t *DentistHashTable) UpdateTimeSlot(name string, timeSession int, month
int, date int, data chan int, wg *sync.WaitGroup)
```

UpdateTimeSlot deletes a time slot from a dentist available time. It takes in timeSession, month and day as integer input, dentist name as string input and return true if the timeslot of the chosen for the dentist exists, else return false. A buffered channel of 1 is also need as the input so that only one update will happen at anytime to avoid racing condition. The waitgroup pointer is passed in if a goroutines is used to execute this function.

type **DentistInfo**

DentistInfo is a structure that contains the dentist information.

```
type DentistInfo struct {
    DrId      int            // dr Id
    TimeAvail *dateHashTable // pointer to a structure that will contains all
the time slots from January to December.
}
```

type **DentistList**

DentistList is a linkedlink structure where its node will contain dentist information.

```
type DentistList struct {
    Head *DentistNode // points to the head of dentist linkedlist
}
```

type **DentistNameList**

DentistNameList is a structure that contains the updated list of dentists.

```
type DentistNameList struct {
    DentistList []string // list of updated list of available dentists
    Size        int      // number of available dentists
}
```

func (*DentistNameList) List

```
func (d *DentistNameList) List() []string
```

List lists the update list of dentists available in the clinics. It will also return a slice of string containing the updated list of available dentists.

type **DentistNode**

DentistNode is a linkedlink node where dentist information is stored.

```
type DentistNode struct {
    Name string       // name of dentist
```

```
    Info *DentistInfo // info will contains drId and time available, refer to
DentistInfo
    Next *DentistNode // points to the next node
}
```

type **TimeBST**

TimeBST is a BST structure that stores the time session for an appointment. Time slot starts from 9AM to 5PM represented by 1 to 15 (each slot has 30minutes duration).

```
type TimeBST struct {
    Root *timeNode // pointer to the root node
    Size int       // number of time slots available (max. 15)
}
```

func (*TimeBST) Add

```
func (t *TimeBST) Add(number int) error
```

Add adds the time node to the BST. It takes in integer (session slot) as input and return error if there is any.

func (*TimeBST) Delete

```
func (t *TimeBST) Delete(number int) error
```

Delete removes the time node from the BST. It takes in integer (session slot) as input and return error if there is any.

func (*TimeBST) InOrder

```
func (t *TimeBST) InOrder() []string
```

InOrder will print the time slot from smallest to largest (earliest to latest timeslot). This function will also return a slice of time slot from smallest to largest (earliest to latest timeslot). for printing of timeslot as it will print from smallest to largest (earliest to latest timeslot) modified for go in action 1 to return []string for timeslot

func (*TimeBST) Search

```
func (t *TimeBST) Search(number int) (bool, error)
```

Search check if the time node exists in the BST. It takes in integer (session slot) as input and return true if it exists, else return false.