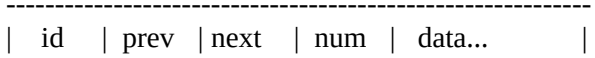
Yan Tu

CSSAP443: MP1
Design Document

1. Class and Public methods definition

The MP1 has four classes: Record, LinkListD, MemoryManager, Tmp1

Record class: represents a fixed size of consecutive memory location. Each record has five kinds of fields (all integers).



idField --> an integer that identifies the each record
prevField --> reference to the previous record, -1 indicates no previous record
nextField --> reference to the next record, -1 indicates no next record
numField --> the number of data in the record
dataField --> the data (integers) in the record

Record class has no data members, just has some static methods that write and read the fields from memory directly.

Methods:

- Sets: following are the methods that write the data into the Record with input value. All data are set to the memory array based on the address of the Record, *ptr*, also is the pointer to the idField.

```
public static void setID (int ptr, int id)          ▷ set idField
public static void setPrev (int ptr, int prev)      ▷ set prevField
public static void setNext (int ptr, int next)      ▷ set nextField
public static void setNum (int ptr, int num)        ▷ set numField
public static void setData (int ptr,int data,int posi) ▷set dataField. posi is 0 for the first data, 1 for the second data …
```

- Gets: following are the methods that read the data from the Record. All data are read from the memory array based on the address of the Record, *ptr*.

```
public static int getID (int ptr)          ▷ get idField
public static int getPrev (int ptr)        ▷ get prevField
public static int getNext (int ptr)        ▷ get nextField
public static int getNum (int ptr)         ▷ get numField
```

- Output: print the record. If the record is marked as free memory, the data in the Record is not for output: ptr [id, pre, next]: data

```
public static void print ( int ptr )
```

LinkListD class: is a doublely linked-list with the property FIFO: insert an element to the end of the list and delete an element from the front of the list. The linked-list is represented in a one dimensional integer array and each node in the list is a record.

Data Member: int head; ▷ *reference to the first Record in the list, -1 indicates empty list*
int tail; ▷ *reference to the last Record in the list, -1 indicates empty list*

Methods:

```
public LinkListD()                 ▷ Constructor
public boolean isEmpty()           ▷ Determine if a list is empty or not
public void insert( int ptr )      ▷ Insert a record to the end of the list with input ( the address of the new record)

public int delete()                ▷ Delete a record from the front of the list. Return the address of the record.

public int remove(int id)          ▷ Remove the record from list which has the same id as input value. The list may include records
                                   with the same idField, remove the one that close to the front. Return the address of the record.if
                                   no matched the record found, return a nullptr (-1).

public void printList()            ▷ Print every element in the list
```

MemoryManager class: manager the free memory in the system. Deleted instances need to be return to the MemoryManager and check the MemoryManager before allocate a memory location.

Data Member:

```
public static int systemMem[];          ▷ Represents the all memory cells in the system
private LinkListD freeRecordList;       ▷ Stores free memory locations, all records with id == -1
```

Methods:

```
public MemoryManager(int memorySize,int recordSize) ▷ Constructor, create the memory array and initialize all the memory
                                                       as free memory

public int getFreeMemory()              ▷ Get a free memor→ call delete method from LinkListD: freeRecordList
public void returnToMemory(int ptr)     ▷ Return memory the MemManager→ call insert method from freeRecordList
public void printMemory()               ▷ Print the infomation of the free memory→call print method from freeRecordList
```

Tmp1 class: implement the program. Most IO code I used here refers to the sample IO code privided by Mr. Kelvin Sung. The Tmp1 only has a static main that does some IO operation and implementation.

```
public static void main(String args[]) ▷ Get the inputs and command from user, then execute the command and provide
                                         response with appropriate error checking.
```

2. Pseudo code & Worse-case run time analysis for the LinkListD class (*Both recordQueue and freeRecordList are instances of the class LinkListD, so I will do thepseudo code and worse-case run time analysis together* **)**

Note: [] means the content of

Constructor:

	Time		**Cost**	**Worse-Case**
`head ← nullPtr`	1	*	Θ(1)	Θ(1)
`tail ← nullPtr`	1	*	Θ(1)	Θ(1)

Worse-case run time analysis: The run-time for this function is T (n) = Θ(1) + Θ(1) = 2 Θ(1) for all the cases. T(n) = O(1)

isEmpty():

	Time		Cost	Worse-Case
`return (head==nullPtr)`	1	*	Θ(1)	Θ(1)

Worse-case run time analysis: The run-time for this function is T (n) = Θ(1) for all the cases. T(n) = O(1)

insert(ptr):

	Time		Cost	Worse-Case
`// the element will be the last element, so no next`				
`next[ptr] ← nullPtr`	1	*	Θ(1)	Θ(1)
`// insert to an empty list, set head and tail`				
`if (isEmpty())`	1	*	Θ(1)	Θ(1)
`prev[ptr] ← nullPtr`	1	*	Θ(1)	3Θ(1)
`head ← ptr`	1	*	Θ(1)	
`tail ← ptr`	1	*	Θ(1)	
`// link the record to the tail and update the tail`				
`else`				3Θ(1)
`prev[ptr] ← tail`	1	*	Θ(1)	3Θ(1)
`next[tail] ← ptr`	1	*	Θ(1)	
`tail ← ptr`	1	*	Θ(1)	

Worse-case run time analysis: The run-time for this function is
T (n) = Θ(1) + Θ(1)+ 3Θ(1) = 5Θ(1) for all the cases. T(n) = O(1)

delete():

	Time		Cost	Worse-Case
`// no more record to delete`				
`if (isEmpty())`	1	*	Θ(1)	Θ(1)
`return nullPtr`	1	*	Θ(1)	Not worse-case
`temp ← head`	1	*	Θ(1)	Θ(1)
`head ← next[head]`	1	*	Θ(1)	Θ(1)

	Time		Cost	Worse-Case
`// delete the only record in the list`				
`if (isEmpty())`	1	*	Θ(1)	Θ(1)
`  tail ← nullPtr`	1	*	Θ(1)	
`else`				Θ(1)
`  prev[head] ← nullPtr`	1	*	Θ(1)	
`// unlink the deleted record`				
`unLinkRecord(temp)`	1	*	Θ(1)	Θ(1)
`return temp`	1	*	Θ(1)	Θ(1)

Worse-case run time analysis: The worse-case run-time for this function is:

T (n) = Θ(1) + Θ(1) + Θ(1) + Θ(1) + Θ(1) + Θ(1) +Θ(1) = 7 Θ(1). T(n) = O(1)

remove(int id):	Time		Cost		Worse-Case
`  // if delete the first record`					
`  if (isEmpty() || id [head] ==id)`	1	*	Θ(1)		Θ(1)
`  return delete()`	1	*	Θ(1)		Not worse case
`  // traversal the list`					
`  current ← head`	1	*	Θ(1)		Θ(1)
`  while ( current != nullPtr )`	t	*	Θ(1)		n Θ(1)→ Θ(n)
`  // if matched record found exit the loop`					
`  if ( id [current]== id )`	t	*	Θ(1)		n Θ(1)→ Θ(n)
`  // if it is last record, update the tail.`					
`  // otherwise link the records before and`					
`  // after the matched record`					
`  if (current==tail)`	1	*	Θ(1)		Θ(1)
`  tail ← prev[current];`	1	*	Θ(1)		
`  else`				Θ(1)	Θ(1)
`  prev[next[current]] ← prev[current];`	1	*	Θ(1)		
`  next[prev[current]] ← next[current];`	1	*	Θ(1)		Θ(1)

	Time		Cost	Worse-Case
`unLinkRecord(current);`	1	*	Θ(1)	Θ(1)
`break;`	1	*	Θ(1)	Θ(1)
`current = next[current];`	t	*	Θ(1)	n Θ(1)→ Θ(n)
`// after the loop if the id matched, current will be the`				
`// address of the record. otherwise, it will be nullPtr`				
`return current;`	1	*	Θ(1)	Θ(1)

Worse-case run time analysis: The run-time for this function is

T (n) = Θ(1) + Θ(1) + t Θ(1) + t Θ(1) + Θ(1) + Θ(1) + Θ(1) + Θ(1) + Θ(1) +Θ(1) + t Θ(1) + Θ(1) = (3t + 8) Θ(1)

The worse case for *t* is to go through all the list which is n, so the worse-case run time is T(n) = 3Θ(n) + 8Θ(1). T(n) = O(n)

printList():

	Time		Cost	Worse-Case
`printListTitle`	1	*	Θ(1)	Θ(1)
`current ← head;`	1	*	Θ(1)	Θ(1)
`while (current != nullPtr)`	n	*	Θ(n)	Θ(n)
`current.print()`	n	*	Θ(n)	Θ(n)
`current ← next[current];`	n	*	Θ(n)	Θ(n)

Worse-case run time analysis: The run-time for this function is

T (n)= Θ(1)+Θ(1)+Θ(n)+Θ(n)+Θ(n)=3Θ(n) + 2Θ(1) for all the cases. T(n) = O(n)

unLinkRecord(ptr)

	Time		Cost	Worse-Case
`next[ptr] ← nullPtr`	1	*	Θ(1)	Θ(1)
`prev[ptr] ← nullPtr`	1	*	Θ(1)	Θ(1)

Worse-case run time analysis: The run-time for this function is T (n) = Θ(1)+Θ(1) = 2Θ(1) for all the cases. T(n) = O(1)