National University of Computer and Emerging Sciences, Lahore Campus

Course:	**Data Structures**	**Course Code:**	**CS 218**
Program:	**BS (CS)**	**Semester:**	**Fall 2023**
Due Date:		**Total Marks:**	**250**
Section:	**3E,3F**	**Page(s):**	**13**
Type:	**Assignment 1**		

Important Instructions:

Submit separate .cpp files for each question. The naming format of each file Should be: 22L-RollNo._Q_No.cpp i.e. 22L-XXX_Q_X.cpp. The files violating the Format won't be considered for grading. DO NOT COMMENT THE CODE. Commented codes will be marked 0. Do not submit .zip files. Late Submissions won't be accepted.

Question # 1: [20 Marks]

Given a singly linked list and a positive integer k, write a function to rearrange the nodes in the list such that the nodes are grouped by k nodes into sub-lists. Within each sub-list, the nodes should be in their original relative order.

Example:

Input:

LIST: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9
k = 3

Output:

[1 -> 2 -> 3] -> [4 -> 5 -> 6] -> [7 -> 8 -> 9]

If the list cannot be divided exactly on k, the last sub-list will be of the remaining nodes that are less than k.

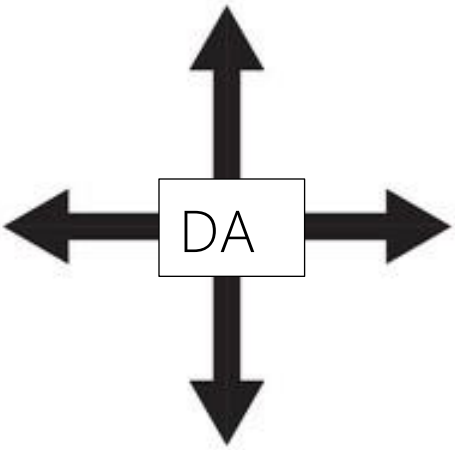
Note: The output should be a linked list of linked lists, where each sub-list is represented as a separate linked list node.

Question # 2: **[50 Marks]**

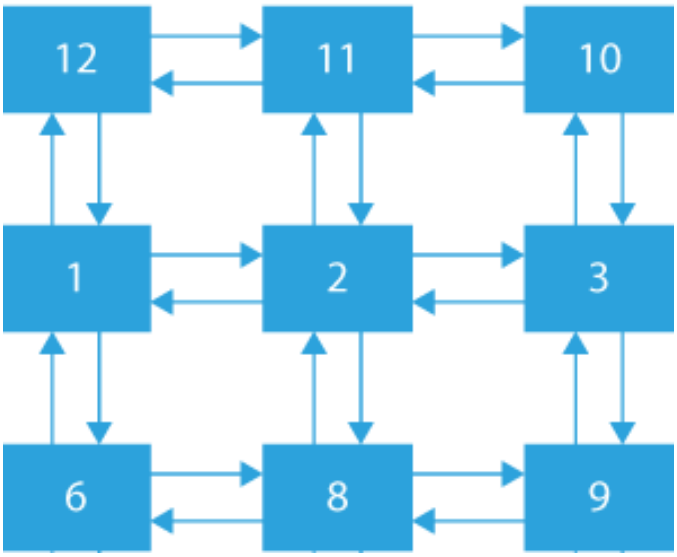
Luck is the second most important thing to reach what you're looking for. Hard work and following the clues is the first. Your task is to reach the elite node in the network of linked lists.

Each node will have following members:

- *int data;*
- *node* up;*
- *node* down;*
- *node* left;*
- *node* right;*



The entire structure will look like:



You have to make the maze using file, where each line represents a row with values seperated by commas. The file of the figure will look like :

12,11,10
1,2,3
6,8,9

You wil start from the top left corner node and reach the next node using the data as the Clue.
Decode the clue as following:

Row of next node = (sum of all digits % No. of rows) + 1
Column of next node = number of digits

Elite Node: The node whose clue will bring you to itself will be the elite node.

Functionalities:

- ✓ **Read(string filename)**

  This method should be able to read data from a text file. Each line in the file contains all the entries in that row, values seperated by commas. There is one row per line.

  12,34,56,78,90
  44,76,34,87,99
  88,65,12,19,50

  This data shows that the maze will consist of 3 rows and 5 columns.

- ✓ **Print(node* head)**

  Prints the maze created.

- ✓ **ClueRow(int data)**

Returns the row number of the next node to be visited (1st row is called row 1).

- ✓ **ClueColumn(int data)**

Returns the column number of the next node to be visited (1st column is called column 1).

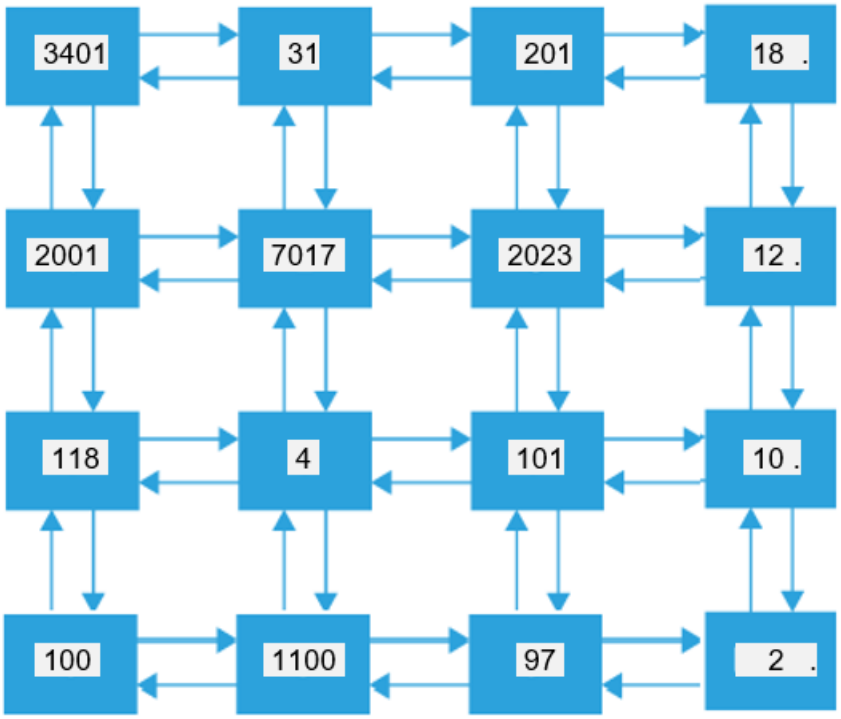
- ✓ **Visited(node* Current)**

  Prints the data of the node you're currently at.
  You need to print all the nodes you visit while reaching the elite node.

- ✓ **EliteNode(node* elite)**

  When elite node is found, send it to this function to print the data and a message that elite node has been found.
  If the there doesn't exist any elite node, your program should be able to display a message for it. Also, if any node get's visited for the second time, the program should end after displaying a meaningful message.

Input:



Output: 3401->18->7017->2->118->101
Elite Node = 101

YOU ARE NOT ALLOWED TO MAKE A 2D MATRIX AT ANY INSTANCE IN YOUR PROGRAM. READING DATA THROUGH FILE AFTER THE CREATION OF THE MAZE IS ALSO NOT ALLOWED.

Traverse the maze (left / right / top / bottom) to reach the next node.

Question # 3: [100 Marks]

In a computer system the main memory (RAM) is a shared resource that all running programs share. The operating systems serves as a resource manager. The main job of the operating system for memory management includes: allocation of free memory to the programs whenever they request it, reclaim the memory of the programs after finishing their execution and keeping record of free and allocated memory.

Programs can claim memory at the start of the program and during its execution. So the memory reserved by one program may not be contiguous. Similarly, due to dynamic nature of the programs, new programs are added to the system and existing programs are removed after finishing their execution. This may cause memory fragmentation. **As processes are loaded and removed from memory, the free memory space is broken into little pieces is called fragmentation.** In the figure below shaded areas are free memory scattered in the main memory.

The job of an operating system is to allocate available byte(s) when a new process(program) is created and deallocate reserved memory when an existing program is finished. A program in memory is stored in multiple bytes. The Memory management module of the operating system maintains a **pool** of available bytes (or contiguous chunks of bytes) where new data of an existing program or a new program can be stored. It also store the list of programs currently in execution and the parts of memory allocated to them. Whenever a new program is started, the memory manager uses available bytes of RAM to store the data of that program on the disk. Similarly when a program is finished, the memory(bytes) allocated to that program are moved back to the pool of available memory.

Whenever a program needs memory, the memory manger asks the total number of bytes required to store the data. The memory manager finds a chunk of contiguous memory in RAM and allocate that memory to the program. If the program asking memory is a new program a new program is added to the list of current programs. Otherwise the information of the block of bytes allocated to the existing program is kept with the program information. A block maintains the Id of starting byte and total number of bytes. So a program in a memory management system has a collection of blocks where each block maintains the address of next block of the program and program stores the address of first block. In order to make the whole process efficient, the memory management system maintains the *pool of available blocks instead of available bytes*.

Your task is to simulate the memory management system. The Memory management system includes: RAM, list of available blocks (pool) and list of Programs currently in execution. We define the RAM as a one-dimensional array of bytes of size numOfBytes. A pool of available blocks is kept in a linked list of block. A block has two fields: start_byte_ID, and total_bytes. Programs are kept in a linked list Progs where each node has a program as data. Each program has an ID, and a linked list of blocks allocated to that program. Asking memory requires claiming a sufficient number of bytes from pool, if available and update the pool accordingly. Removal of a program requires removing the list of blocks allocated to the program and transferring the bytes assigned to this program back to pool.

Suppose we have a RAM of size 100KB. This means we will have byte ID's from 0 to 102400-1.

Figure 1 shows a link list of blocks called pool. The first available block is of 30 bytes starting from byte ID 420 and last block is of 9900 bytes starting at byte ID 4000. Figure 2 shows a sample link list of Programs. There are 3 programs in the memory. The Id of program1 is P1 currently has acquired 260 bytes. It also tells that the data of the program is stored in 3 blocks. First block starts at byte ID 0 and has 40 bytes, second block has 20 bytes that starts from byte ID 70 and last block also has 200 bytes starting from byte ID 120.

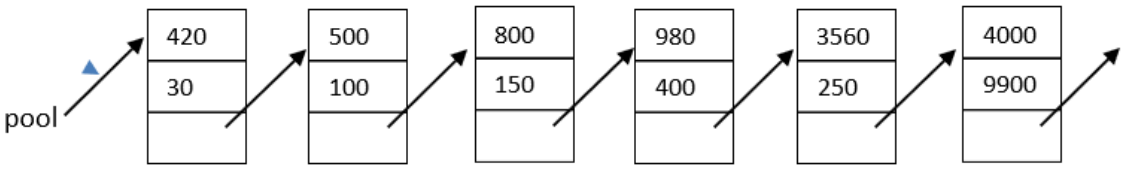


Figure1: pool of blocks

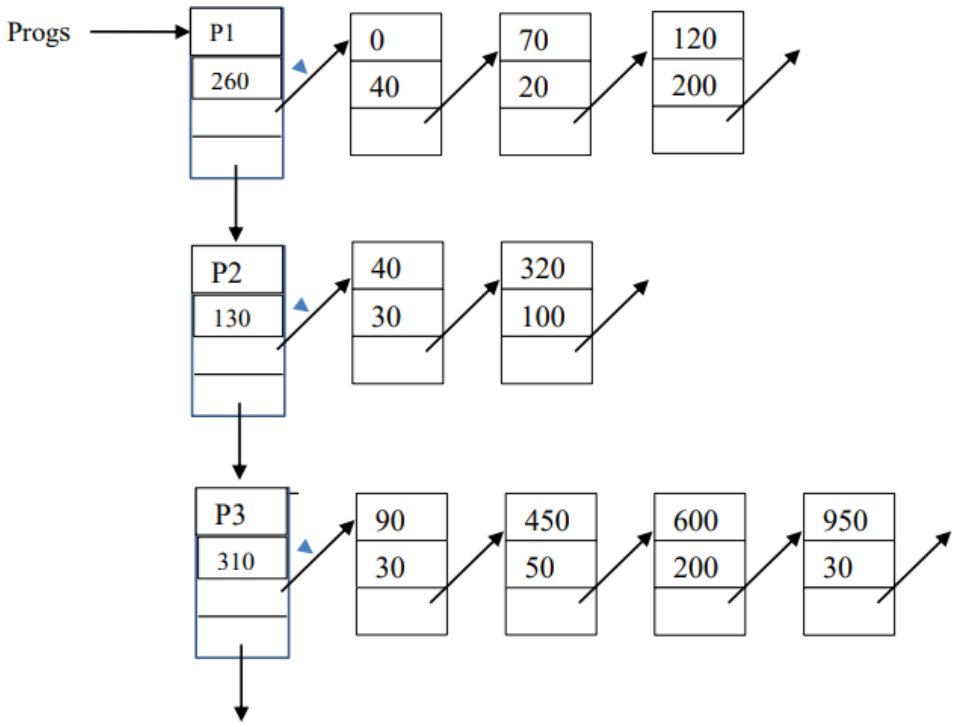


Figure2: Programs list containing three Programs P1, P2 and P3

In order to implement the memory management system you need to implement the following classes:

Class template of node

Data members:

- data
- next

Class template of singly linked list

Data member:

- head
- tail
- size

Class of block

Data member:

- start_byte_ID
- total_bytes

Class of program

Data members:

- Id
- Size (memory)
- Link list of blocks

Class MemoryManagementSystem

Data members:

- pool (Sorted Linked list of block)
- Progs (Linked list of Programs)
- sizeOfMemory
- strategy (a Boolean variable)

Member functions:

GetMem: The GetMem function must take program Id and size of the memory required. as parameters and should work as follows:

Find the memory of required size from the pool of free memory using strategy s (will be explained later). If the requested memory is not available then return false. Otherwise, Id is searched in the list of programs. If the program Id already exists then a new block of memory is removed from the pool of free memory and added to the existing program at the end of the program linked list of blocks. If no such program already exists then first a new program is inserted at the end of Progs and then this block is added to the list of blocks of the newly inserted program.

The memory management system implements one of the two strategies:

First fit strategy:
The memory management system allocates the first available block of memory that fits the program requirement.

Best fit strategy:
The management system allocates the free block of free memory that has smallest size and meets the requirement of the program.

For example a new program P4 requests a block of 200 size. According to the first fit strategy, the memory will be allocated starting at 980 byte. This block has 400 bytes. The first 200 bytes will be allocated to P4 and remaining 200 resides in the pool. The updated pool and Progs list is shown the figure 3 and figure 4.

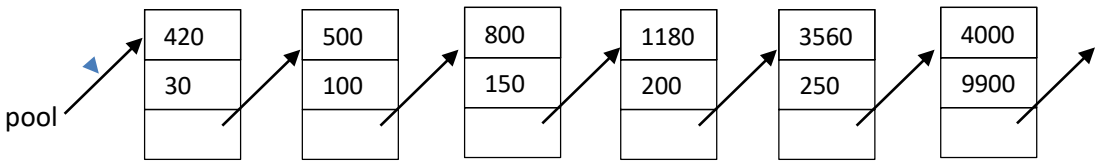


Figure3: Updated pool of blocks using first fit strategy

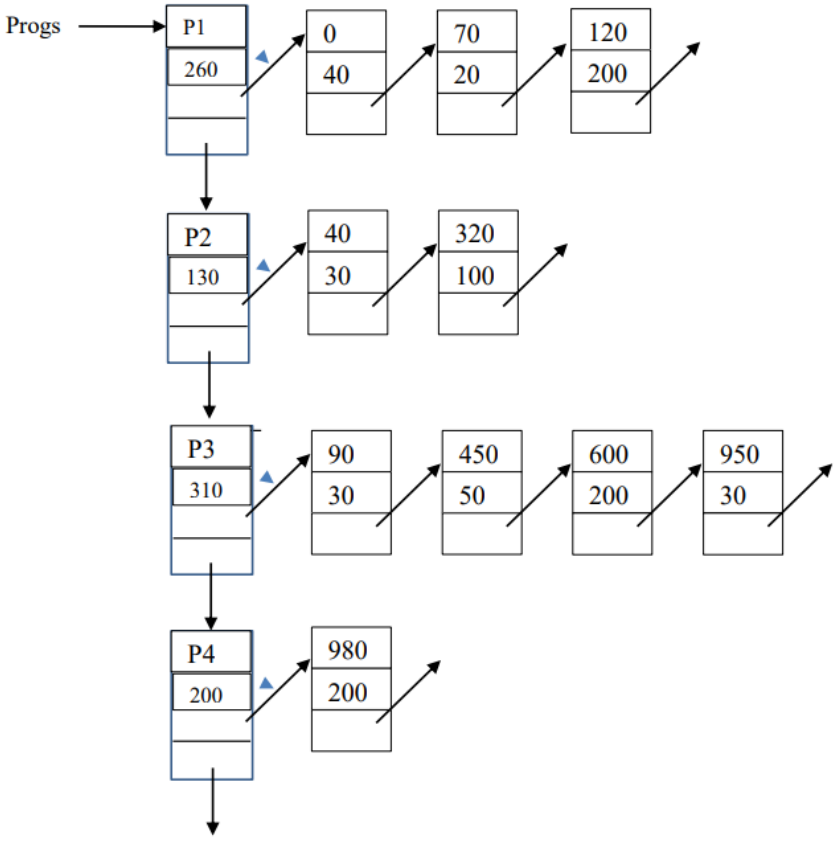


Figure4: Updated Progs list using first fit strategy

However if the strategy is best fit then the block starting at 3560 byte will be allocated. The updated pool and Progs is shown in Figure 5 and 6.

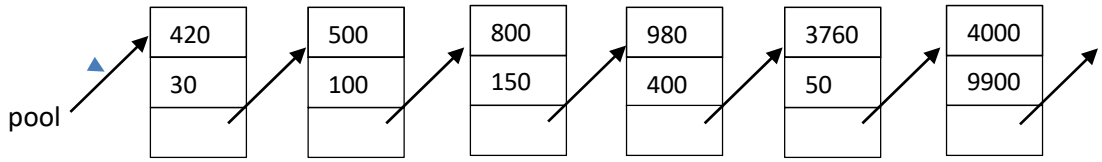


Figure5: Updated pool of blocks using best fit strategy

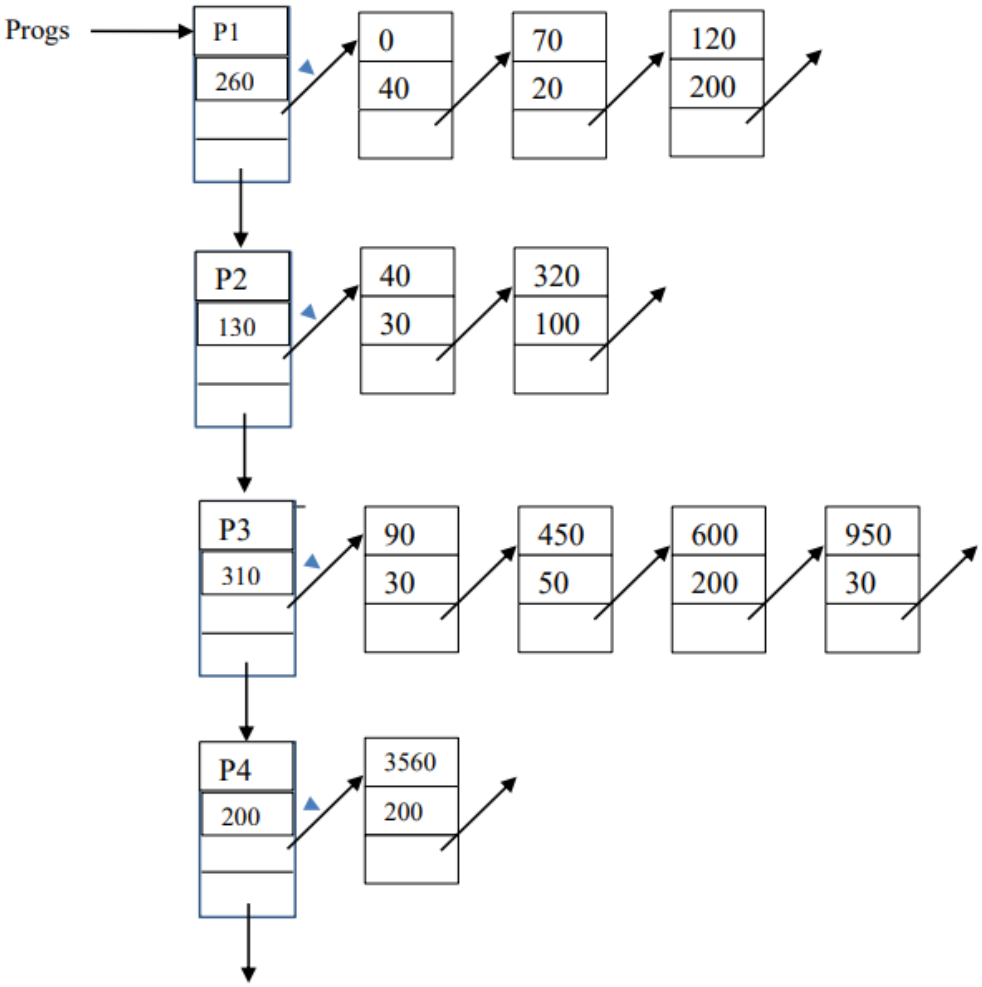


Figure6: Updated Progs list using best fit strategy

DeleteProgram:

This function must take program Id as parameter and delete the program with the given Id from Progs. Also move all the blocks of the memory to pool in sorted order. If two consecutive block become one single contiguous block then you must also merge them into one single block. For example if program P3 is deleted then the block starting at 450 must be merged with the block in pool starting at byte position 420 and so on. The updated pool will be as follows

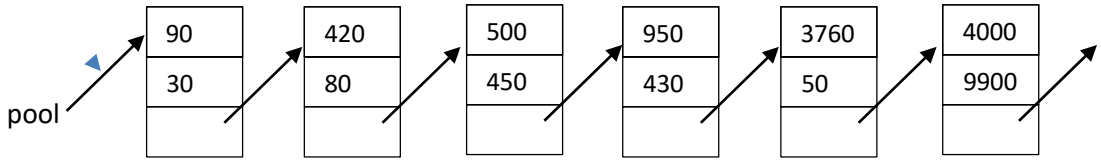


Figure5: Updated pool of blocks after deletion of P3

Constructor:

Initially Progs is empty and pool has only one block. SizeofMemory and Strategy will be initialized to user provided values and will remain the same throughout the program.

Provide a driver function that creates a memory management system object and a menu that prompts the user to perform different operations of the memory management.

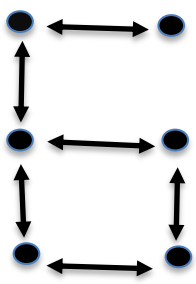
Question # 4: **[80 Marks]**

You all may have noticed the digits displayed on calculators and digital clocks. Every digit is derived from the digit '8'.

Consider that every point where two or more lines meet in a node. The structure of a single digit will be as following where each black dots represents a node.

Each node has 4 pointers (node* left, node* right, node* up, node* down). Any digit can be written using these 6 nodes, by establishing essential links.

For example, 6 can be made as:



The link is two way because it does not matter you link it left to right or right to left.
You will receive a time input from the user in a 24-Hours format.

Your task is to:

Make separate linked list for each digit, and print the time.

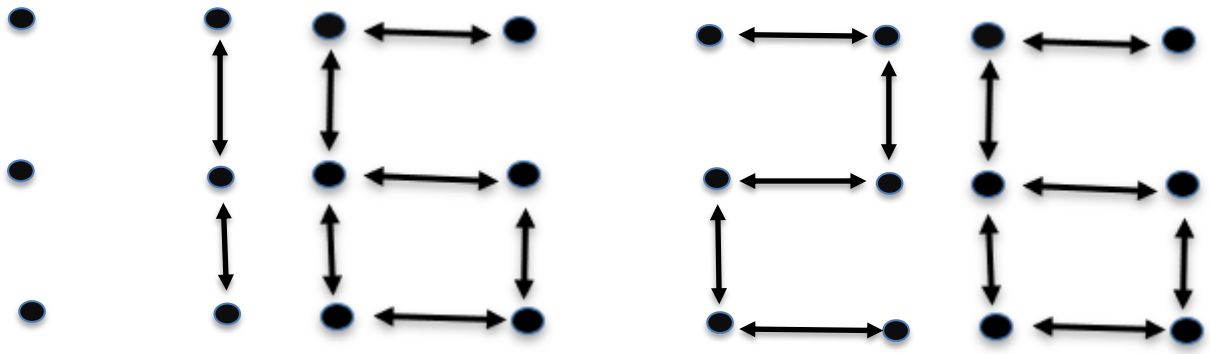
Sample Input:

16:26

Output:

```
*  *******     *******  *******
*  *           *  *     *
*  *******     *******  *******
*  *     *     *        *     *
*  *******     *******  *******
```

Internal Structure:



You need to make sure that all the pointeres are properly handled. Every pointor that is not pointing to any of the node should be pointing to null.

FUNCTIONAL IMPLEMENTATIONS:

1. Addition of Minutes

The function should take the minutes input from the user, add it in the time and print the time after addition of respective minutes.
NOTE: Addition of minutes may affect hours.

2. Subtraction of Minutes

The function should take the minutes input from the user, subtract it from the time and print the time after subtaction of respective minutes.
NOTE: subtraction of minutes may affect hours.

3. Addition of Hours

The function should take the hours input from the user, add it in the time and print the time after addition of respective hours.

4. Subtraction of Hours

The function should take the hours input from the user, subtract it from the time and print the time after subtaction of respective hours.

5. Conversion to 12 hour Format

The function should display the entered time in 12 hour format along with AM/PM.

6. Change of Date

The program should display a date changed message if any of the functionalities above, when performed on time, results in the changing of date.