Question 1
Code

pp dayType.h dayType.cpp

dayType

```
1  #include <string>
2  #include <iostream>
3
4  using namespace std;
5
6  class dayType {
7
8  public:
9      dayType();
10     dayType(int input);
11
12     ~dayType();
13
14     std::string getDay() const;
15     std::string getNextDay() const;
16     std::string getPreviousDay() const;
17
18     void setDay(int day);
19
20     void print() const;
21
22     std::string calcDay(int numDays);
23
24 private:
25     int day;
26     const std::string DAYS[7] = {"sun", "mon", "tues", "wed", "thurs", "fri", "sat" };
27 };
```

.cpp | dayType.h | dayType.cpp

x1

```cpp
#include <string>
#include <iostream>
#include "dayType.h"

using namespace std;

dayType::dayType() {
    day = 0;
}

dayType::dayType(int input) {
    day = input;
}

dayType::~dayType() {}

std::string dayType::getDay() const {
    return DAYS[day];
}

std::string dayType::getNextDay() const {
    return DAYS[(day + 1) % 7];
}

std::string dayType::getPreviousDay() const {
    return DAYS[(day - 1) % 7];
}

void dayType::setDay(int input) {
    day = input;
}

void dayType::print() const {
    cout << "Your day is: " << DAYS[day] << endl;
}

string dayType::calcDay(int numDays) {
    return DAYS[(day + numDays) % 7];
}
```

```cpp
#include <iostream>
#include "dayType.h"

using namespace std;

int main() {
    int day;
    cout << "Enter a day of the week as a number. Sunday = 0 and Saturday = 6. " << endl;
    cin >> day;
    dayType myDay(day);
    myDay.print();
    int numDays;
    cout << "Enter the number of days to add: " << endl;
    cin >> numDays;
    cout << "Your new day is: " << myDay.calcDay(numDays) << endl;
    return 0;
}
```

Output

```
Microsoft Visual Studio Debug Console
Enter a day of the week as a number. Sunday = 0 and Saturday = 6.
1
Your day is: mon
Enter the number of days to add:
4
Your new day is: fri
```

```
Enter a day of the week as a number. Sunday = 0 and Saturday = 6.
2
Your day is: tues
Enter the number of days to add:
13
Your new day is: mon
```

Question 2

a. Singly linked list

```
struct Node {
  int data;
  Node next;
}

class LinkedList {

        Node head;
```

```
// method to add a new node to the end of the linkedlist
append(LinkedList list, Node newNode) {
  // if the list is empty i.e. head node is null, the new node becomes the head
  if (list->head == null) {
    list->head = newNode;
  } else {
    // if the list is not empty, iterate over linked list until you find the last node, then set
    // the new node as the next node
    Node curr = head;
    // exits the loop when the next node is null, ie curr is the last node of the list
    while (curr->next != null) {
      curr = curr->next;
    }
    curr->next = newNode;
  }

// method to add a new node to the front of the linkedlist
prepend(LinkedList list, Node newNode) {
  // if the list is empty, the new node becomes the head
  if (list->head == null) {
    list->head = newNode;
  } else {
  // if the list is not empty, set the new node's next to the current head, then make the
  // new node the new head
  newNode->next = list->head;
  list->head = newNode;
}

// method to insert a new node after a given node
insert(LinkedList list, Node preNode, Node newNode) {
  // if list is empty, sets the head to the new node
  if (list->head == null) {
    list->head = newNode;
  } else if (contains(list, preNode)) {
    // if the node exists, insert after that node
    Node curr = list->head;
    // loop end condition is if the current node is the node to insert after
    while (curr != preNode) {
      curr = curr->next;
    }
    newNode->next = curr->next;
    curr->next = newNode;
  } else {
    // if the node does not exist, insert at the end of the list
```

```
      append(list, newNode);
    }
  }

  // method to remove the node after the given node
  remove(LinkedList list, Node pre) {
    // case 1: head node is node to remove
    if (list->head == null) {
    // set head to be the 2nd node in the list
      list->head = list->head->next;
    // case 2: given node is in list
    } else if (contains(list, pre)) {
      // iterate until given node
      Node curr = list->head;
      while (curr != pre) {
         curr = curr->next;
       }
      // case 2a: node to remove is not tail
      if (curr->next->next != null) {
        curr->next = curr->next->next;
      } else {
      // case 2b: node to remove is tail
        curr->next = null;
      }
    }
  }
  // case 3: given node not in list (do nothing)

  // method that returns whether a given node exists in the given list
  contains(LinkedList list, Node node) {
    Node curr = list->head;
    while (curr != null) {
      if (curr == node) {
        return true;
      } else {
        curr = curr->next;
      }
    }
    return false;
  }
}
```

b. Doubly linked list

```
struct Node {
  int data;
  Node prev;
  Node next;
}

class DoublyLinkedList {

Node head;
Node tail;

// add the new node to the end of the doubly linked list
append(DoublyLinkedList list, Node newNode) {
        // if list is empty new node is head and tail
        if (list->head == null) {
                list->head = newNode;
        }
        else {
                // set the next of the current tail to the new node
                list->tail->next = newNode;
                // set the prev of the new node to the current tail
                newNode->prev = list->tail;
                // update the tail to the new node
                list->tail = newNode;
        }
}

// append the new node to the start of the doubly linked list
prepend(DoublyLinkedList list, Node newNode) {
        // if list is empty new node is head and tail
        if (list->head == null) {
                list->head = newNode;
        }
        else {
        // set new node's next to curr head
        // set curr head's prev to new node
        // set head to new node
                newNode->next = list->head;
                list->head->prev = newNode;
                list->head = newNode;
        }
}
```

```
// insert the new node in the doubly linked list after the given node
insert(DoublyLinkedList list, Node key, Node newNode) {
	// if list is empty new node is head and tail
	if (list->head == null) {
		list->head = newNode;
	}
	// if key is tail, set new node to tail and connect to list
	else if (key == list->tail) {
		list->tail->next  = newNode;
		newNode->prev = list->tail;
		list->tail = newNode;
	}
	// if key is not head or tail, set key's next to new Node and
	// new node's next to key's next
	else {
		newNode->next = key->next;
		key->next->prev = newNode;
		newNode->prev = key;
		key->next = newNode;
	}
}

// remove the given node in the doubly linked list
remove(DoublyLinkedList list, Node key) {}
	// case 1: node to remove is not null
	if (key->next != null) {
		key->next->prev = key->prev;
	}
	// case 2: prev node of key is not null
	if (key->prev != null) {
		key->prev->next = key->next;
	}
	// case 3: given key is the head
	if (key == list->head) {
		list->head = key->next;
		key->next->prev = null;
	}
	// case 4:
	if (key == list->tail) {
		list->tail = key->prev;
		key->prev->next = null;
	}
}
```