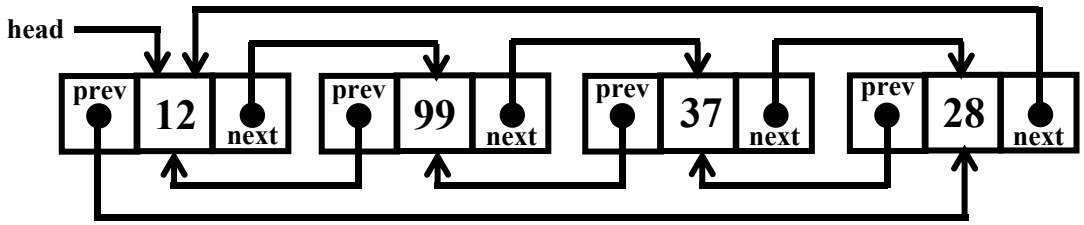
5. (42%) Dice Game with Templated Doubly Circular Linked List

In a *doubly circular linked list(DCLL)*, each node contains, besides the next-node link, a second link field pointing to the "previous" node in the sequence. The two links may be called "forwards" and "backwards", or "**`next`**" and "**`prev`**"("previous"). The last node of the linked list points to the first node of the list (i.e., the "next link" pointer of the last node has the memory address of the first node and the "prev link" pointer of the first node has the memory address of the last node). For example, for the doubly circular linked list "**`12 -> 99 -> 37 -> 28 -> 12(head)`**", it can be represented as with "**`head`**" points to "**`12`**":



A partial **`class DCLL`** is defined in **`DCLL.h`**:

DCLL.h

```
template <class T>
class DCLL {
   // Overview: a templated doubly circular linked list

  public:
   T *getVal() const;
   // EFFECTS: return the val of the node pointed by head

   bool isEmpty() const;
   // EFFECTS: returns true if the list is empty
   //          and false otherwise.

   void insert(T *val);
   // MODIFIES: this
   // EFFECTS: Insert a new node at the head of the list;
   //          the original node pointed by head becomes
   //          the next node of the new node.
   // EXAMPLE: Initial: 1 -> 3 -> 5 -> 1(head)
   //          Insert(4): 4 -> 1 -> 3 -> 5 -> 4(head)

   T *remove();
   // MODIFIES: this
   // EFFECTS: If the list is empty, return NULL.
   //          Otherwise, remove the node at the head of the
   //          list; the original node pointed by head->next
   //          becomes pointed by head.
   // EXAMPLE: Initial: 4 -> 1 -> 3 -> 5 -> 4(head)
   //          Remove: 1 -> 3 -> 5 -> 1(head)
```

```
   DCLL(); // constructor
   DCLL(const DCLL<T> &l); // copy constructor
   DCLL<T> &operator=(const DCLL<T> &l); // assignment operator
   ~DCLL(); // destructor

   // (3) To Do ...

  protected:
   struct Node {
      Node *next;
      Node *prev;
      T *val;
   };
   Node *head;

   void removeAll();
   // MODIFIES: this
   // EFFECTS: called by destructor/operator= to remove and destroy
   //          all list elements

   void copyAll(Node *head);
   // MODIFIES: this
   // EFFECTS: called by copy constructor/operator= to copy elements
   //          from the head of source instance to this->head

   std::ostream &print(std::ostream &os) const;
   // MODIFIES: os
   // EFFECTS: print the structure of this list through os
};

template <class T>
T *DCLL<T>::getVal() const {
   return this->head->val;
}

template <class T>
bool DCLL<T>::isEmpty() const {
   return !this->head;
}

template <class T>
void DCLL<T>::insert(T *val) {
   Node *newNode = new Node;
   newNode->val = val;
   if (this->isEmpty()) {
```

```
      // (2-a) To Do ...
   } else {
      // (2-b) To Do ...
   }
   this->head = newNode;
}

template <class T>
T *DCLL<T>::remove() {
   if (this->isEmpty()) return NULL;
   // (2-c) To Do ...
   Node *victim = this->head;
   T *ret = this->head->val;
   // (2-d) To Do ...
   delete victim;
   return ret;
}

template <class T>
DCLL<T>::DCLL() : head(NULL) {}

template <class T>
DCLL<T>::DCLL(const DCLL<T> &l) : head(NULL) {
   this->copyAll(l.head);
}

template <class T>
DCLL<T> &DCLL<T>::operator=(const DCLL<T> &l) {
   this->removeAll();
   this->copyAll(l.head);
}

template <class T>
DCLL<T>::~DCLL() {
   this->removeAll();
}

template <class T>
void DCLL<T>::removeAll() {
   while (!this->isEmpty()) {
      this->remove();
   }
}

template <class T>
void DCLL<T>::copyAll(Node *ohead) {
```

```
   if (ohead == NULL) return ;
   Node *pos = ohead->prev;
   do {
      insert(pos->val);
      pos = pos->prev;
   } while (pos != ohead->prev);
}

template <class T>
std::ostream &DCLL<T>::print(std::ostream &os) const {
   if (this->isEmpty()) {
      os << "\n[Empty List]\n";
      return os;
   }
   Node *pos = this->head;
   int index = 0;
   do {
      os << "\n[" << index++ << "] : \n";
      os << "prev : " << pos->prev->val << "\n";
      os << "this : " << pos->val << "\n";
      os << "next : " << pos->next->val << "\n";
      pos = pos->next;
   } while (pos != this->head);
   return os;
}
```

You are going to complete templated **`DCLL`** ADT. **Illegible solutions WILL BE discarded upon grading.** It is your job to think first before writing down anything.

(1) (7%) There are five obvious problems in **`DCLL.h`**. Find these problems and write down how to fix them. (Directly write codes with brief descriptions.)

(1-a)

(1-b)

(1-c)

(1-d)

(1-e)

****

Solution:

(1-a) (1%) miss header guard:

```
#ifndef DCLL_H
#define DCLL_H
...
#endif
```

(1-b) (2%) in `copyAll`:

```
insert(pos->val);
```

should be:

```
T *newVal = new T(*(pos->val));
insert(newVal);
```

(1-c) (1%) in `print`:

```
os << "prev : " << pos->prev->val << "\n";
os << "this : " << pos->val << "\n";
os << "next : " << pos->next->val << "\n";
```

should be:

```
os << "prev : " << *(pos->prev->val) << "\n";
os << "this : " << *(pos->val) << "\n";
os << "next : " << *(pos->next->val) << "\n";
```

(1-d) (1%) in `operator=` miss return value, should be:

```
return *this;
```

(1-e) (2%) in `removeAll`:

```
this->remove();
```

should be:

```
T *obj = this->remove();
delete obj;
```

****

(2) (10%) Fill in the blanks in the implementation of function `insert` and `remove`. **Your solution must be provided in the following box. At most one statement each line.**

(2-a)

********** 2 lines limitation for (2-a) **********

(2-b)

********** 4 lines limitation for (2-b) **********

(2-c)

********** 2 lines limitation for (2-c) **********

(2-d)

********** 2 lines limitation for (2-d) **********

(ONLY SOLUTIONS IN THE BOX ARE VALID)

****

Solution:

(2-a) (2%)

```
newNode->next = newNode;
newNode->prev = newNode;
```

(2-b) (4%)

```
newNode->next = this->head;
newNode->prev = this->head->prev;
this->head->prev->next = newNode;
this->head->prev = newNode;
```

(2-c) (2%)

```
this->head->prev->next = this->head->next;
this->head->next->prev = this->head->prev;
```

(2-d) (2%)

```
if (this->head != this->head->next) this->head = this->head->next;
else this->head = NULL;
```

****

(3-a) (3%) Overloading output operator **<<** for the class **`DCLL`** which displays the structure of the list. You can use any member function which already exists and declare **`operator<<`** as a friend of class **`DCLL`**. **Your solution must be provided in the following box. At most one statement each line.**

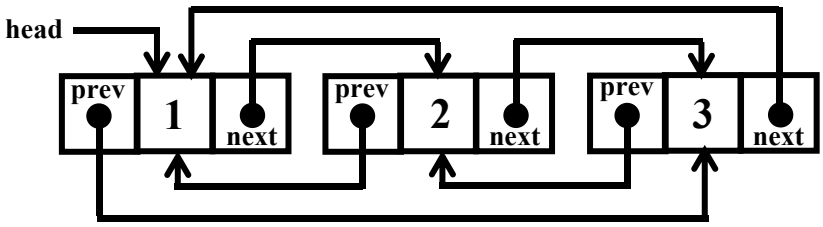
(3-a)

(ONLY SOLUTIONS IN THE BOX ARE VALID)

(3-b) (3%) What is the output of "`std::cout << l;`" where:

(i) `l` is a type of `DCLL<int>`

(ii) `l` is with the following structure:



(3-b)

(ONLY SOLUTIONS IN THE BOX ARE VALID)

****

Solution:

(3-a) (3%)

```
friend std::ostream &operator<<\
   (std::ostream &os, const DCLL<T> &l) {
   l.print(os);
```

```
   return os;
}
```

(3-b) (3%)

```
[0] :
prev : 3
this : 1
next : 2

[1] :
prev : 1
this : 2
next : 3

[2] :
prev : 2
this : 3
next : 1
**************************************************************************
```

Now, you have an abstract data type. Then, a client Vonda comes to you. He wants to design a dice game. In the game, several players sit in a circle, take turns to roll the dice and pass a bomb. **The default direction is clockwise**. Before the game starts, the one of the players hands the bomb. The game has **m** rounds. For each round, the player who hands the bomb will first rolls the dice to get a random number **n** from 1 to 6. Then, he/she decides whether to change the direction of the game (from counterclockwise to clockwise OR from clockwise to counterclockwise). Finally, he/she throws the bomb to the $\mathbf{n^{th}}$ player next to him/her according to the direction. After **m** rounds the bomb will explode and the player who hands it will be eliminated. Vonda discovers that the **`DCLL`** is a nice ADT for this dice game design. Every node can represent a player and the circular structure is similar to players sitting in a circle. However, **`DCLL`** doesn't provide the corresponding public member functions for some behaviors in his game (e.g. passing the bomb among players, changing the direction and moving to next player according to the direction). So, Vonda wants you to design a class in file **`players.h`** called **`Players`** which inherits from class **`DCLL`**. Read the content carefully and understand what each function and attribute is used for.

players.h

```
#ifndef PLAYERS_H
#define PLAYERS_H

#include "DCLL.h"
using std::uint8_t;

enum Direction : uint8_t {
   clockwise, // the direction of the game is clockwise
   counterclockwise, // the direction of the game is counterclockwise
   nothing // exception handling for direction
```

```
};

template <class T>
class Players : public DCLL<T> {
   // OVERVIEW: derived doubly circular linked list for dice game
  public:
   void nextPlayer();
   // MODIFIES: this->hand
   // EFFECTS: move hand through one node according to the dir
   // EXAMPLE: clockwise:
   //             original: P1 -> P2 -> P3 -> P1(head)
   //             nextPlayer: P2 -> P3 -> P1 -> P2(head)
   //          counterclockwise:
   //             original: P1 -> P2 -> P3 -> P1(head)
   //             nextPlayer: P3 -> P1 -> P2 -> P3(head)

   void changeDir();
   // MODIFIES: this->dir
   // EFFECTS: change the direction of the game
   // EXAMPLE: from clockwise to counterclockwise or
   //          from counterclockwise to clockwise

   Direction getDir();
   // EFFECTS: return dir

   Players(); // constructor
   Players(const Players<T> &p); // copy constructor
   Players<T> &operator=(const Players<T> &p); // assignment operator

  private:
   Direction dir; // record the direction of the game
};

// (4-b) To Do ...

template <class T>
Direction Players<T>::getDir() {
   return this->dir;
}

template <class T>
Players<T>::Players() : DCLL<T>(), dir(clockwise) {}

template <class T>
Players<T>::Players(const Players<T> &p) : \
   DCLL<T>(p), dir(clockwise) {}
```

```
template <class T>
Players<T> &Players<T>::operator=(const Players<T> &p) {
   this->removeAll();
   this->copyAll(p.head);
   return *this;
}

#endif
```

(4-a) (3%) Correspond the following behaviour or things with the member attributes or the member functions in the class **`Players`**: (You just need to write down the name of the member attributes or the member functions)

(i) bomb	
(ii) passing the bomb according to the direction	
(iii) changing the direction	

(4-b) (7%) Complete the definition of two member functions of **`Players`**:

(i) **`nextPlayer`**

(ii) **`changeDir`**

Your solution must be provided in the following box. At most one statement each line.

(i)

******************************* 10 lines limitation for (i) *******************************

(ii)

********** 8 lines limitation for (ii) **********

(ONLY SOLUTIONS IN THE BOX ARE VALID)

****

Solution:

(4-a) (3%)

(i) (1%) `hand`

(ii) (1%) `nextPlayer`

(iii) (1%) `changeDir`

(4-b) (7%)

(i) (4%)

```
template <class T>
void Players<T>::nextPlayer() {
   if (this->isEmpty()) return ;
   else if (this->dir == clockwise) \
      this->head = this->head->next;
   else if (this->dir == counterclockwise) \
      this->head = this->head->prev;
   else if (this->dir == nothing) return ;
}
```

(ii) (3%)

```
template <class T>
void Players<T>::changeDir() {
   if (this->dir == nothing) return ;
   else if (this->dir == counterclockwise) this->dir = clockwise;
   else if (this->dir == clockwise) this->dir = counterclockwise;
}
```

****

Vonda is satisfied with your **`Players`** class. However he needs your help to complete has game. Read the codes and commands he wrote in **`game.cpp`**:

`game.cpp`

```
#include <iostream>
#include <string>
#include "players.h"

class Act {
   // OVERVIEW: record the number of dice rolled and the choice
   //           made (whether change direction) by the player
```

```
  public:
    void setChangeDir(bool changeDir) { this->changeDir = changeDir; }
    void setJump(unsigned int jump) { this->jump = jump; }
    bool getChangeDir() { return this->changeDir; }
    unsigned int getJump() { return this->jump; }

    Act(bool changeDir, unsigned int jump) : \
       changeDir(changeDir), jump(jump) {}

  private:
    bool changeDir; // If decide to change, true. Otherwise, false.
    unsigned int jump; // The number rolled.
};

class Player {
    // OVERVIEW: one player in the game

  public:
    std::string getName() { return this->name; }
    unsigned int getAge() { return this->age; }
    Act *getAct() { return &(this->act); }

    void acts();
    // MODIFIES: this->act
    // EFFECT: read player's number rolled and choice from std::cin
    //         and update these information to this->act

    Player(std::string const &name, unsigned int age) : \
       name(name), age(age), act(false, 0) {}

  private:
    void print(std::ostream &os) const {
       os << name << "(" << age << ")";
    }

    std::string name; // name of the player
    unsigned int age; // age of the player
    Act act; // number and choice of the player
};

void Player::acts() {
    std::string dirInfo;
    unsigned int jump = 0;
    std::cin >> dirInfo >> jump;
    if (dirInfo == "Change") // (5-a):(i) To Do ...
    else if (dirInfo == "NotChange") // (5-a):(ii) To Do ...
```

```
    // (5-a):(iii) To Do ...
    this->print(std::cout);
    std::cout << " acts : dir(" << dirInfo << "), jump(" << jump << ")\n";
}

int main() {
    unsigned int round = 0;
    std::cin >> round; // read number of rounds from std::cin
    Players<Player> p;
    p.insert(new Player("Alice", 18));
    p.insert(new Player("Peter", 20));
    p.insert(new Player("Danny", 17));
    p.insert(new Player("Kitty", 21));
    p.insert(new Player("Steve", 16));
    for (unsigned int i = 0; i < round; i++) {
        // (5-a):(iv) To Do ...
        if (p.getVal()->getAct()->getChangeDir()) p.changeDir();
        const unsigned int jump = p.getVal()->getAct()->getJump();
        // (5-a):(v) To Do ...
    }
    // (5-a):(vi) To Do ...
    delete del;
    return 0;
}
```

(5-a) (6%) In Vonda's programme, he tries to simulate his game with five players: Alice, Peter, Danny, Kitty and Steve. Try to fill in the blanks in **`game.cpp`**. **At most one statement each blank.**

(i)	
(ii)	
(iii)	
(iv)	
(v)	
(vi)	

(5-b) (3%) There exists one potential problem with Vonda's code. Briefly explain what the problem is **or** directly write down codes to fix it.

****

Solution:
(5-a) (6%)
(i) (1%) `this->act.setChangeDir(true);`
(ii) (1%) `this->act.setChangeDir(false);`

(iii) (1%) `this->act.setJump(jump);`
(iv) (1%) `p.getVal()->acts();`
(v) (1%) `for (unsigned int j = 0; j < jump; j++) p.nextPlayer();`
(vi) (1%) `Player *del = p.remove();`
(5-b) (3%) Miss overloading output operator **<<**. "**`std::cout << p;`**" will report an error. To fix this problem, the following codes should be added to the "**`public`**" field of class **`Player`**.

```
friend std::ostream &operator<<(std::ostream &os, const Player &p) {
   p.print(os);
   return os;
}
```

****