Design

```
#include <iostream>

#include <algorithm>

using namespace std;


template <typename key, typename info>

class dlr{ //double linked ring

private:

    struct element{  //struct is declared in private scope to protect it from functions outside the class

        key id;

        info data;

        element* previous;//pointing back

        element* next;//pointing front

    };

    element *head;//head

    element *tail;//head->previous


    int length;//length of ring

    void removeAllElements(); //remove all elements

    void copyAllElements(const dlr& obj); //copy all elements


public:

    int getLength() const; //return length;

//continued in next page
```

```
    class custIterator
    {
        friend class dlr;
    private:
        custIterator(element* item); //constructor with one argument for iterator
        element* item;
    public:
        custIterator(const custIterator& obj);//copy constructor
        bool operator!=(custIterator const& obj) const;//operator!= overloading for
iterator
        bool operator==(custIterator const& obj) const;//operator== overloading for
iterator
//two get methods
        info getD() const;
        key getI() const;
        custIterator& operator++();//prefix operator++ overloading for iterator, goes to
next
        custIterator& operator--();//prefix operator-- overloading for iterator, goes to
previous
    };//end of custIterator class declaration
```

//continued in next page

```
    typedef custIterator iterator;
    dlr(); //empty constructor
    ~dlr(); //destructor
    void pushFront(key id, info data); //insert at head
    void insertAt(key id, info data, int pos);//insert at a particular position
    void pushBack(key id, info data);//insert element at tail
    //two basic iterator functions
    iterator begin() const;//return custIterator(head);
    iterator end() const;//return custIterator(tail);
    void removeElement (int pos);//function to remove an element at a particular position
//two get functions needed to access the data from element without exposing pointers
    info getData(int n) const;
    key getId(int n) const;
    void print() const;//print function
    dlr (const dlr & obj);//copyconstructor
    dlr & operator=(const dlr & obj);//assignment operator overloading
    dlr operator+(const dlr & obj);//operator+ overloading
    dlr operator-(const dlr & obj);//operator- overloading
    bool isEmpty() const; //function to check if dlr is empty

};//end of double linked ring class declaration
```

//continued in next page

```
template <typename key, typename info>//produce function allows two create a third dlr out
of 2 supplied as arguments along with other correct input
dlr<key, info> produce (const dlr<key,info>& ring1, int start1, int step1, bool dir1,
                        const dlr<key,info>& ring2, int start2, int step2, bool dir2,
                        int num, bool dir){
    dlr<key,info> ring3;
    try{
        if (ring1.getLength()<2&&ring1.getLength()<=start1){
            throw 1;
        }
        if (ring2.getLength()<2&&ring2.getLength()<=start2){
            throw 2;
        }
    } catch (int j){
        cerr <<j<<" ring position is unacceptable"<<endl;
        return ring3;
    }
    auto iter1 = ring1.begin();//start from beginning ring 1
    auto iter2 = ring2.begin();//start from beginning ring 2
    if(dir1){//go forward in the ring 1
        for (int i=0;i<start1;i++){
            ++iter1;
        }//stop when you reach starting position ring 1
    } else{//go backward in ring 1
        for (int i=0;i<start1;i++){
            --iter1;
        }//stop when you reach starting position ring 1
    }
    if(dir2){//go forward in ring 2
        for (int i=0;i<start2;i++){
            ++iter2;
        }}//stop when you reach starting position ring 2
    } else{//go backward in ring 2
        for (int i=0;i<start2;i++){
            --iter2;
        }}//stop when you reach starting position ring 2
```

```
    }
    for (int j=0;j<num;j++){//third loop for ring3
        for (int k=0;k<step1;k++){//first loop for ring1
            if (dir){
                ring3.pushBack(iter1.getI(), iter1.getD());//adding at the “end”
            } else {
                ring3.pushFront(iter1.getI(), iter1.getD());//adding at the “beginning”
            }
            if (dir1){
                ++iter1;//going through forward in ring1
            } else {
                --iter1;//going through backward in ring1
            }
        }//first elements from first sequence inserted
        for (int l=0;l<step2;l++){//second loop for ring2
            if (dir){
                ring3.pushBack(iter2.getI(), iter2.getD());
            } else {
                ring3.pushFront(iter2.getI(), iter2.getD());
            }
            if (dir2){
                ++iter2;//going through forward in ring2
            } else {
                --iter2;//going through backward in ring2
            }
        }
    }
    return ring3;
};
```

Implementation

The Produce external function first checks if starting position is available in the ring or not by use of try-throw-catch block. Then by using iterator, it finds the starting point at which the ring will be copied. Ring1 and Ring2 are later added to Ring3 with two for loops nested inside a bigger for loop, with if statements checking the direction of which way it should copy (bool dir for Ring3 and bool dir1 and dir2 for Ring1 and Ring2 respectively).

The method does not work with wrong parameters where length of the ring is less or equal to starting position given in Produce function arguments. Length cannot be less than starting position for obvious reasons, it cannot start copying a value beyond the scope of the ring.

(Although it might be possible for it to loop until start%length=0 or something similar but I have not implemented it as it was not necessary in the task)