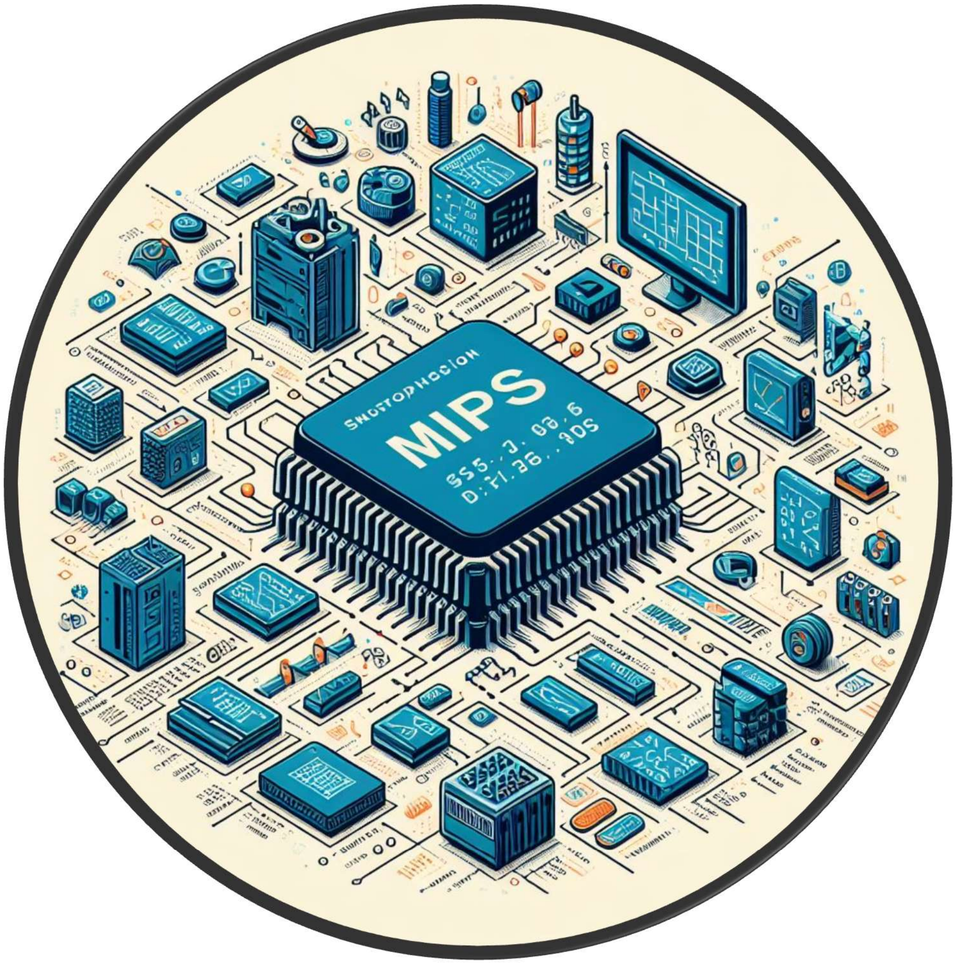
Deques data structure implementation using mips assembly.



Team members

- *Mahmoud Atef Mahmoud Abdelaziz*
- *Mohamed Toukhy Mohamed Ahmed*
- *Youssef Taha Saad Taha*
- *Ali Hossam Ali Nour*
- *Mahmoud Hossen saaid*

Introduction:

The term Deque is an abbreviation for Double Ended Queue. It is a specialized data structure that allows insertion and deletion operations to be performed from both ends, thereby generalizing the concept of a queue.

The deque holds several advantages that make it a preferred choice over other data structures. These include:

- ***Double-Ended Operations**: The most significant advantage of a deque is its ability to add and remove elements from both ends (front and rear). This flexibility allows it to serve as both a queue (FIFO) and a stack (LIFO), making it a versatile data structure.*
- ***Efficient Operations**: Adding or removing elements from either end of a deque is an*
- *O (1) operation, which means it can perform these operations quickly, regardless of the size of the deque. This efficiency is a significant advantage in scenarios where such operations are frequent.*
- ***Sliding Window Problems**: Deques are particularly useful in certain types of problems known as 'sliding window' problems. In these problems, you need to keep track of elements in a 'window' that moves through an array or list. Because you can add or remove elements from both ends in a deque, they are an ideal data structure for these types of problems.*
- ***Real-Time Data**: Deques are useful in real-time computing where the time to process a data element is critical. The element can be added or removed from either side of the deque in constant time and allows for faster processing.*
- ***Data Buffer**: Deques can also be used as a buffer for data streams. The streaming data can be continuously populated for processing from either side.*

Combing both the implementation of the deque with a level low language like assembly will be challenging and a bit interesting.

***MIPS Assembly Language** is a low-level language that offers a granular view of how computer hardware operates. It provides direct control over the system's resources, which can lead to highly efficient and optimized code. However, this comes with its own set of challenges. The syntax is less intuitive than high-level languages, and the programmer must manage memory and registers manually. Despite these challenges, mastering MIPS Assembly Language can provide invaluable insights into the inner workings of computer systems.*

Combining these two - implementing a deque in MIPS Assembly Language - would indeed be an interesting endeavor. It would not only test your understanding of the deque data structure and MIPS Assembly Language but also your ability to translate high-level concepts into low-level operations. This task, while challenging, could significantly enhance your problem-solving skills and deepen your understanding of computer systems.

Deque Implementation:

In this project, we utilized a high-level programming language, specifically C, to construct the code that implements a deque data structure. Following the development in C, we then translate this code into MIPS assembly language.

The C code comprises a multitude of functions, each serving a unique purpose in the implementation of the deque. Here's a brief overview of these functions:

- ✓ *createDeque(): Initializes a new deque.*
- ✓ *insertFront(): Adds an element at the front of the deque.*
- ✓ *insertRear(): Appends an element at the rear of the deque.*
- ✓ *isEmpty(): Checks if the deque is empty.*
- ✓ *deleteFront(): Removes an element from the front of the deque.*
- ✓ *deleteRear(): Deletes an element from the rear of the deque.*
- ✓ *getFront(): Retrieves the front element of the deque.*
- ✓ *getRear(): Fetches the rear element of the deque.*
- ✓ *getSize(): Returns the current size of the deque.*
- ✓ *clear(): Empties the deque.*
- ✓ *display(): Displays the current state of the deque.*

createDeque() *is a function that is used to create a deque and it allows us to define more than one deque at the same program. This idea of creating such a function will help us with the implementation of the project based on the deque. This function must be called before creating any deque. This function is responsible for storing 12bytes from the heap these byte will contain three parts 4bytes for the size of the at which the size will stored and updated, the other 8bytes will be also divided into 4bytes for the front pointer and the other 4bytes will be for the back pointer.*

This is the mips assembly used to implement this function:

```
createDeque:
                la $a0, dequeSize
                lw $a0, 0($a0)
                li $v0, 9
                syscall
                jr $ra
```

*The variable **dequeSize** holds the value of 12, signifying the allocation of 12 bytes of memory by the program. This memory is reserved for the deque operations.*

insertFront () *and* ***insertRear ()*** *functions are used to add elements to the deque. While they serve the same purpose, the key difference lies in where they insert the elements. insertFront () adds an element at the beginning of the deque, while insertRear() appends an element at the end. Both functions utilize the createNode() function to create a new node, which includes a value and pointers to the next and previous nodes, forming a doubly-linked list.*

One of the advantages of a deque is its ability of insertion with a time complexity of O(1).

```
insertRear:
         addi $sp, $sp, -20
         sw $s0, 0($sp)
         sw $s1 ,4($sp)
         sw $ra ,8($sp)
         sw $t2,12($sp)
         sw $t0,16($sp)
         beq $a0,$zero,exitInsertRear
         move $s0,$a0   #$s0 ----> address of the deque
         jal creatNode
         move $s1,$v0   #$s1 ----->node address
         lw $t0 8($s0)
         beq  $t0 ,$zero, firstInsertionAll
         lw $t2 8($s0)  #$t2 ------>address of the last node before insertion
         sw $t2 0($s1)
         sw $zero 4($s1)
         sw $s1 4($t2)
         sw $s1 8($s0)
        j endInsertRear
        firstInsertionAll:
                sw $s1 4($s0)
                sw $s1 8($s0)
                sw $zero, 0($s1)
                sw $zero, 4($s1)
        endInsertRear:
                lw $t5 ,0($s0)
                addi $t5,$t5,1
                sw $t5,0($s0)
        exitInsertRear:
                lw $s0, 0($sp)
                lw $s1, 4($sp)
                lw $ra, 8($sp)
                lw $t2,12($sp)
        lw $t0,16($sp)
                addi $sp, $sp, 20
                jr $ra
```

Both functions will update the value of the next pointer of the old node to point to the new node and the previous pointer of the new node will point to the old node.

```
insertFront:
        addi $sp, $sp, -20
        sw $s0, 0($sp)
        sw $s1, 4($sp)
        sw $ra, 8($sp)
        sw $t1,12($sp)
        sw $t0,16($sp)
        move $s0, $a0   # address returned from createDeque function
        beq $s0, $zero, endInsertion
        jal creatNode
        move $s1, $v0 # address of the node
        lw $t0, 4($s0)
        beq $t0, $zero, emptyDeque
        sw $t0, 4($s1) #t0--------> address of the old front
        sw $zero ,0($s1)
        sw $s1, 0($t0)
        sw $s1, 4($s0)
        lw $t1, 0($s0)
        addi $t1, $t1, 1
        sw $t1, 0($s0)
        j endInsertion
        emptyDeque:
                sw $s1, 4($s0)
                sw $s1, 8($s0)
                sw $zero, 0($s1)
                sw $zero, 4($s1)
                lw $t1, 0($s0)
                addi $t1, $t1, 1
                sw $t1, 0($s0)
        endInsertion:
                lw $s0, 0($sp)
                lw $s1, 4($sp)
                lw $ra ,8($sp)
                lw $t1,12($sp)
                lw $t0,16($sp)
                addi $sp, $sp, 20
                jr $ra
```

createNode() *is the fucntion that is used to implement the douple linked list. We can consider that the create node function is the foundation stone of the deque. The node will serve 17 byte from the heap. The first 4bytes will contain the previous pointer, the other 14byte will be divide into 4bytes and 10bytes the 4bytes will store the next pointer while the other 10bytes will store the values that will be add at the project.*

Createnode *is the base for implementing the double linked list.*

```
creatNode:
        addi $sp, $sp, -4
        sw $s0, 0($sp)
        li $a0, 17
        li $v0, 9
        syscall
        move $s0, $v0
        sw $a1, 8($s0)
        sw $a2, 12($s0)
        sb $a3, 16($s0)
        lw $s0, 0($sp)
        addi $sp, $sp, 4
        jr $ra
```

The ***isEmpty()*** *function checks whether the deque is empty. It returns a boolean value - true if the deque is empty, and false otherwise. This function is crucial as it helps prevent errors and can be invoked at any point in the program.*

```
isEmpty:
    addi $sp, $sp, -4
    sw $ra, 0($sp)
    jal checkCreation
    lw $t0, 8($a0)



    beq $t0, $zero, empty
    li $v0, 0              # Return zero if not null
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    jr $ra
    empty:
    li $v0, 1
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    jr $ra
```

deleteFront() *and* ***deleteRear()*** *functions are integral to the deque operations. deleteFront() removes an element from the front of the deque, while deleteRear() eliminates an element from the rear. These functions ensure the deque remains dynamic and adaptable to various data manipulation needs. Simply put, they keep the deque clean and efficient by removing elements when necessary.*

```
deleteFront:
    addi $sp, $sp, -16
    sw $ra, 0($sp)
    sw $t0,4($sp)
    sw $t2,8($sp)
    sw $t3,12($sp)
    jal isEmpty
    move $t0, $v0
    bne $t0, $zero, error
    lw $t0, 4($a0)
    sw $zero, 8($t0)
    sw $zero, 12($t0)
    sw $zero, 16($t0)
    lw $t2, 4($t0)          # Get the next ptr to make it first
    sw $zero, 4($t0)
    sw $zero, 0($t0)
    beq $t2, $zero, out
        sw $zero, 0($t2)        # Make first ptr of 2nd element 0
    out:
    sw $zero, 4($t0)        # Put 0 in popped next ptr
    lw $t3, 0($a0)          # Get size
    addi $t3, $t3, -1
    sw $t3, 0($a0)
    sw $t2, 4($a0)          # Put the next ptr of the popped in first
    bne $t2, $zero, go
    sw $zero, 8($a0)
    go:
        lw $ra, 0($sp)
         lw $t0,4($sp)
    lw $t2,8($sp)
    lw $t3,12($sp)
        addi $sp, $sp, 16
        jr $ra
```

```
deleteRear:
        addi $sp , $sp , -28
        sw   $ra ,0($sp)
        sw   $s0 ,4($sp)
        sw   $t0,8($sp)
        sw   $t1,12($sp)
        sw   $t2,16($sp)
        sw   $t3,20($sp)
        sw   $t4,24($sp)
        move $s0,$a0    #s0------------>address of deque
        jal isEmpty
        move $t0 , $v0
        li $v0,-1
        bne $t0,$zero, EmptyDeque
        lw $t2 , 8($s0)
        move $t1,$t2
        lw $t3, 0($t2)
        sw $t3 , 8($s0)
        lw $t4 ,8($s0)
        beq $t4 , $zero,LastDeletionRear
        sw $zero 4($t4)
        j  endDeleteRear
        LastDeletionRear:
                sw $zero 4($s0)
        j endDeleteRear
        EmptyDeque:
                la $a0 ,emptyErrorMsg
                li $v0 ,4
                syscall
                li $v0 10
                syscall
        j exitDeleteRear
        endDeleteRear:
                lw $t6  0($s0)
                addi $t6,$t6,-1
                sw $t6,0($s0)
                sw $zero 0($t1)
                sw $zero 4($t1)
                sw $zero 8($t1)
                sw $zero 12($t1)
                sw $zero 16($t1)
        exitDeleteRear:
                lw $ra ,0($sp)
                lw $s0 ,4($sp)
                lw   $t0,8($sp)
                lw   $t1,12($sp)
                lw   $t2,16($sp)
                lw   $t3,20($sp)
                lw   $t4,24($sp)
                addi $sp ,$sp,28
                jr $ra
```

getFront() *and* ***getRear()*** *functions are the gatekeepers of the deque. getFront() retrieves the front element, giving you immediate access to the first item in the deque. On the other hand, getRear() fetches the rear element, providing a quick peek at the last item. These functions offer a simple and efficient way to access your data without disrupting the structure of the deque. They're like the front and back doors to your deque house!.*

getFront() and getRear() functions are often the most frequently used when working with a deque. The unique structure of a deque, with its front and rear pointers, allows us to perform a variety of useful operations.

For instance, we can display the value at the front or the rear of the deque, or even delete these values. This flexibility is one of the key advantages of using a deque. In essence, these functions provide us with direct access to both ends of the deque, enabling efficient manipulation and retrieval of data, which is particularly useful in certain algorithms and data processing tasks. It's like having a two-way street for data, enhancing both versatility and efficiency.

```
getRear:
    addi $sp, $sp, -4
    sw $ra, 0($sp)
    jal isEmpty
    move $t0, $v0
    bne $t0, $zero, error
    lw $v0, 8($a0)
    jr $ra
```

```
getFront:
    addi $sp, $sp, -4
    sw $ra, 0($sp)
    jal isEmpty
    move $t0, $v0
    bne $t0, $zero, error
    lw $v0, 4($a0)
    jr $ra
```

getSize() *function plays a seemingly simple yet crucial role in managing a deque. It returns the current size of the deque, providing an immediate count of the elements it contains. While it may appear straightforward, its utility is widespread across the program.*

This function is particularly useful in scenarios where the number of elements in the deque needs to be monitored or limited. For instance, it can prevent the addition of elements beyond a certain limit, or trigger certain actions when the deque is empty or reaches a specific size. In essence, getSize() serves as a quick check on the deque's capacity, contributing significantly to its efficient management. It's like a handy gauge that keeps track of the deque's occupancy at all times.

```
getSize:
        lw $v0 ,0($a0)
        jr $ra
```

clear() *function is used to empty the deque. It traverses from the first node to the final node using the front pointer, deleting all the values along the way. Once all elements are removed, it resets both the front and rear pointers to NULL. In essence, clear() wipes the deque clean, leaving it empty and ready for new operations.*

```
clear:
        addi $sp, $sp, 8
        sw $s0, 0($sp)
        sw $s3, 4($sp)
        lw $s0, 0($a0)
        lw $s3, 4($a0)
        beq $s0, $zero, endLoop
        Loop:
                beq $s0, $zero, endLoop
                add $t1, $zero, $s3
                lw $s3, 4($s3)
                sw $zero, 0($t1)
                addi $t2, $t1, 4
                sw $zero, 0($t2)
                addi $t2, $t2, 4
                sw $zero, 0($t2)
                addi $t2, $t2, 4
                sw $zero, 0($t2)
                addi $t2, $t2, 4
                sb $zero, 0($t2)
                bne $s3 ,$zero,Loop
        endLoop:
                sw $zero, 0($a0)
                sw $zero, 4($a0)
                sw $zero, 8($a0)
                lw $s0, 0($sp)
                lw $s3, 4($sp)
                addi $sp, $sp, 8
                jr $ra
```

display() *function is like the spotlight on a stage, revealing the current state of the deque. It traverses through the deque from the front to the rear, showcasing each element along the way. This function provides a clear and comprehensive view of the deque's contents, making it an invaluable tool for debugging and understanding the flow of data. In essence, display() is the deque's personal narrator, telling the story of its elements in a simple and accessible manner. It's like having a guided tour of your deque!*

```
display:
    addi $sp, $sp, -36
    sw $ra, 0($sp)
    sw $v0, 4($sp)
    sw $s0, 8($sp)
    sw $t0,12($sp)
    sw $t1,16($sp)
    sw $t2,20($sp)
    sw $t3,24($sp)
    sw $a0,32($sp)
    jal isEmpty
    move $t0, $v0
    bne $t0, $zero, error
    lw $t2, 4($a0)          # Front ptr
    lw $s0, 0($a0)       # size in t1
    li $t1, 1
    li $v0, 4
    la $a0, line2
    syscall
    loop_print:
                lw $t3, 8($t2)
                li $v0, 4
                la $a0, product
                syscall
                li $v0, 1
                move $a0, $t1
                syscall
                addi $t1, $t1, 1   #product num.
                li $v0, 4
                la $a0, coul
                syscall
                li $v0, 4
                la $a0, price
                syscall
                li $v0, 1
                move $a0, $t3
                syscall
                li $v0, 11
                li $a0, 10
                syscall
                li $v0, 4
                la $a0, ser
                syscall
                lw $t3, 12($t2)
                li $v0, 1
                move $a0, $t3
                syscall
                li $v0, 11
                li $a0, 10
              syscall
              li $v0, 4
              la $a0, cat
              syscall
              lb $t3, 16($t2)
              #addi $t3, $t3, -48  # Subtract ASCII offset to convert from ASCII to character
              move $a0, $t3
              li $v0, 11
              syscall
              li $v0, 11
              li $a0, 10
              syscall
              li $v0 ,4
              la $a0 ,line
              syscall
              lw $t2, 4($t2)
   bne $t2, $zero, loop_print
      lw $ra, 0($sp)
   lw $v0, 4($sp)
   lw $s0, 8($sp)
   lw $t0,12($sp)
   lw $t1,16($sp)
   lw $t2,20($sp)
   lw $t3,24($sp)
   lw $a0,32($sp)
    addi $sp,$sp,36
     jr $ra
```

Application Implementation:

After that, we need to use a deque in real-life applications as follows :

“ You've just joined JUMIA as a backend engineer, where you're tasked with designing and developing the best-selling bar and similar bars, akin to this: https://www.jumia.com.eg/ar/.

Your focus lies on incorporating the price and category of products for quicker sorting and display, along with the serial number to retrieve additional information from the database, such as photos and names. ’’

To implement this we will concentrate on three related information for the product the price category and serial number we concentrate on price and category to sort and manage items and serial numbers to reach more data in the dataset of site like pictures, names and etc...

The code :

```
typedef struct{
    Deque* data;
    Deque* view;
    int sz;
}Bar;

void init(Bar* bar);

void add_product(Bar* bar,int price,int serial_number,char category);

void right_arrow(Bar* bar);

void left_arrow(Bar* bar);

Bar* createBar(){
    Bar* bar = (Bar*)malloc(sizeof(Bar));
    bar->data=bar->view=NULL;
}

void init(Bar* bar) {
    bar->data = createDeque();
    bar->view = createDeque();
    bar->sz = 0;
    int n;
    printf("Enter number of initial products :\n");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter price and serial number and category :\n");
        int price ;
        int serial_number ;
        char category ;
        scanf("%d %d %c", &price, &serial_number, &category);
        add_product(bar, price, serial_number, category);
    }
    display(bar->view);
}

void add_product( Bar* bar,int price,int serial_number,char category){
    bar->sz++;
    if (bar->sz <= 6)
        insertRear(bar->view, price,serial_number ,category);
    else
        insertRear(bar->data, price,serial_number ,category);
}

void left_arrow(Bar* bar) {
    if (bar->view->front == NULL || bar->data->rear == NULL) {
        printf("No more data to move left.\n");
        return;
    }

    // Remove data from the front of the view deque
    Product *tmp =  getFront(bar->view);

    // Insert removed data at the front of the data deques
```

```c
    insertRear(bar->data,tmp->price,tmp->serialNumber,tmp->category);

    deleteFront(bar->view);
    // Remove data from the rear of the data deques
    tmp = getFront(bar->data);

    // Insert removed data at the rear of the view deque
    insertRear(bar->view,tmp->price,tmp->serialNumber,tmp->category);
    deleteFront(bar->data);

    display(bar->view);
}

void right_arrow(Bar* bar) {
    if (bar->view->rear == NULL || bar->data->front == NULL) {
        printf("No more data to move right.\n");
        return;
    }

    // Remove data from the rear of the view deque
    Product *tmp =  getRear(bar->view);

    // Insert removed data at the front of the data deques
    insertFront(bar->data,tmp->price,tmp->serialNumber,tmp->category);

    deleteRear(bar->view);
    // Remove data from the rear of the data deques
    tmp = getRear(bar->data);

    // Insert removed data at the rear of the view deque
    insertFront(bar->view,tmp->price,tmp->serialNumber,tmp->category);
    deleteRear(bar->data);

    display(bar->view);
}

int main() {
    Bar* bestSellingBar=createBar();
    init(&bestSellingBar);
    char c;
    while (1) {
        printf("Enter 'l' or 'L' to click on the left button,\n'r' or 'R' to
click on the right button,\nand any other character to finish: ");
        scanf(" %c", &c);
        if (c == 'l' || c == 'L')
            left_arrow(&bestSellingBar);
        else if (c == 'r' || c == 'R')
            right_arrow(&bestSellingBar);
        else
            break;
    }
    printf("Mission completed successfully!!!\n");
    return 0;
}
```

1. Bar Structure

The Bar structure consists of the following components:

- *Deque* data: A pointer to a deque that stores the actual product data.*
- *Deque* view: A pointer to a deque that represents the current view of the products.*
- *int sz: An integer representing the size of the bar, indicating the number of products.*

2. Functions

***Bar* createBar()**: This function allocates memory for a new Bar structure and initializes its components, setting both data and view pointers to NULL. However, it lacks a return statement which should return the allocated Bar structure.*

```
createBar:
                la $a0, barSi
                lw $a0, 0($a0
                li $v0, 9
                syscall
```

***void add_product(Bar* bar, int price, int serial_number, char category)**: This function adds a new product to the bar. If the size of the bar is less than or equal to 6, the product is added to the view deque; otherwise, it is added to the data deque. The sz variable is incremented to reflect the addition of the product.*

```
addProduct:
 addi $sp, $sp, -
 sw $ra, 0($sp)
 sw $t0 ,4($sp)
 lw $t0 ,8($a0)
 addi $t0,$t0,1
 sw $t0,8($a0)
 beq $t0,6,c
 slti $t1 ,$t0 ,
 beq $t1 , 1 , c
 lw  $a0 ,0($a0)
 jal insertRear

 j endAdd
 c:
 lw  $a0 ,4($a0)
 jal insertRear
 endAdd:
```

void init(Bar* bar)*: This function initializes the Bar structure by prompting the user to input the number of initial products and their details (price, serial number, and category). It then adds these products to the view deque. The function also sets the sz variable accordingly.*

```
init:
        addi $sp, $sp,-28
        sw $ra ,0($sp)
        sw $s0 ,4($sp)
        sw $s1 ,8($sp)
        sw $s2 ,12($sp)
        sw $s3 ,16($sp)
        sw $t0 ,20($sp)
        sw $t9 ,24($sp)
        move $s2,$a0        #$s2-------------
        jal createDeque
        move $s0,$v0        #$s0-------------


        jal createDeque
        move $s1,$v0        #$s1-------------



        sw $s0, 0($s2)
        sw $s1 ,4($s2)
```

```
            li $v0, 4
                    la $a0, ser
                    syscall
            li $v0, 5
            syscall
            move  $a2,$v0

            li $v0, 4
                    la $a0, cat
            syscall
            li $v0, 12
            syscall
            move  $a3,$v0

            move $a0,$s2

            jal addProduct

            addi $t0,$t0,1
            bne $t0,$s3,initLoop
```

```
lw $ra ,0($sp)
lw $s0 ,4($sp)
lw $s1 ,8($sp)
lw $s2 ,12($sp)
lw $s3 ,16($sp)
lw $t0 ,20($sp)
lw $t9 ,24($sp)
addi $sp,$sp,28
```

void left_arrow(Bar* bar): *This function simulates the action of clicking the left arrow button, which moves data from the view deque to the data deque. It removes the front product from the view deque, adds it to the rear of the data deque, removes the front product from the data deque, and adds it to the rear of the view deque. Finally, it displays the updated view deque.*

```
leftArrow:



        addi $sp, $sp,-28
        sw $ra ,0($sp)
        sw $t0 ,4($sp)
        sw $t1 ,8($sp)
        sw $t2 ,12($sp)
        sw $t3 ,16($sp)
        sw $t4 ,20($sp)
        sw $t5 ,24($sp)

        move $t0,$a0
        lw $t1,0($t0) # data deque
        lw $t2,4($t0) # view deque
        lw $t3,8($t1) # rear of data
        lw $t4,4($t1) # front of view
        beqz $t3,printNoData
        beqz $t4,printNoData

        # Remove data from the front of the view
```

void right_arrow(Bar* bar)*: This function simulates the action of clicking the right arrow button, which moves data from the data deque to the view deque. It performs the opposite operations of the left_arrow function, moving data from the rear of the data deque to the front of the view deque.*

```
########### RIGHT ARROW  ################
rightArrow:
        addi $sp, $sp,-28
        sw $ra ,0($sp)
        sw $t0 ,4($sp)
        sw $t1 ,8($sp)
        sw $t2 ,12($sp)
        sw $t3 ,16($sp)
        sw $t4 ,20($sp)
        sw $t5 ,24($sp)



        move $t0,$a0
        lw $t1,0($t0) # data deque
        lw $t2,4($t0) # view deque
        lw $t3,4($t1) # front of data
        lw $t4,8($t1) # rear of view
        beqz $t3,printNoData
        beqz $t4,printNoData

        # Remove data from the rear of the view de

       # Insert removed data at the front of the v
       move $t5,$v0  # data of deleted item .
       move $a0,$t2
       lw $a1,8($t5)
       lw $a2,12($t5)
       lw $a3,16($t5)
       jal insertFront

       # display items
       move $a0,$t2
       jal display
       lw $ra ,0($sp)
       lw $t0 ,4($sp)
       lw $t1 ,8($sp)
       lw $t2 ,12($sp)
       lw $t3 ,16($sp)
       lw $t4 ,20($sp)
       lw $t5 ,24($sp)
```

3. Main

- ***Bar Initialization***: *The main function begins by creating a Bar instance named bestSellingBar using the createBar function. It then initializes this instance by calling the init function, which prompts the user to input the number and details of initial products and adds them to the bar's view.*
- ***User Interaction Loop***: *After initialization, the main function enters a while loop that continues indefinitely until the user decides to exit. Within this loop, the user is prompted to input a character representing an action: 'l' or 'L' for clicking the left button, 'r' or 'R' for clicking the right button, or any other character to finish.*
- ***Button Click Handling***: *Depending on the user input, the main function calls either the left_arrow or right_arrow function to simulate clicking the corresponding button. These functions move product data within the Bar structure from one deque to another, updating the view accordingly.*
- ***Loop Termination***: *The loop terminates when the user inputs any character other than 'l', 'L', 'r', or 'R'. At this point, the program prints a message indicating successful completion of the mission and exits.*

```
main:

jal createBar
move $s0,$v0

move $a0,$s0
jal init

whileTrue:
        la $a0, prompt
        li $v0, 4
        syscall

        li $v0, 12
        syscall
        move $t0 ,$v0
        li $t1, 'l'
        li $t2, 'L'
        beq $t0, $t1, doLeft
        beq $t0, $t2, doLeft
```

Conclusion

The implementation of the Bar structure and its associated functions provides basic functionality for managing product data and views within a bar management system. However, there are some areas for improvement, such as error handling, memory management, and potential optimizations for better performance.

Overall, the Bar structure and its functions serve as a good starting point for building a bar management system, but further enhancements are necessary to make it more robust and user-friendly.

Output Sample

Mars Messages | Run I/O

Clear

```
Enter number of initial products :
8

Enter price and serial number and category of Product(1):
Price: 54
Serial Number: 54
Category: a
Enter price and serial number and category of Product(2):
Price: 45
Serial Number: 54
Category: b
Enter price and serial number and category of Product(3):
Price: 654
Serial Number: 465
Category: c
Enter price and serial number and category of Product(4):
Price: 65
Serial Number: 65
Category: d
Enter price and serial number and category of Product(5):
Price: 65
Serial Number: 65
Category: e
Enter price and serial number and category of Product(6):
Price: 65
Serial Number: 65
Category: f
Enter price and serial number and category of Product(7):
Price: 65
Serial Number: 65
Category: g
Enter price and serial number and category of Product(8):
Price: 65
Serial Number: 56
Category: h
```

```
Product 1:
Price: 45
Serial Number: 65
Category: e
==============================
Product 2:
Price: 54
Serial Number: 465
Category: f
==============================
Product 3:
Price: 45
Serial Number: 54
Category: g
==============================
Product 4:
Price: 45
Serial Number: 56
Category: h
==============================
Product 5:
Price: 54
Serial Number: 564
Category: a
==============================
Product 6:
Price: 45
Serial Number: 65
Category: b
==============================
Enter 'l' or 'L' to click on the left button,
'r' or 'R' to click on the right:
```

Task assignment

63.Mahmoud Atef Mahmoud AbdelAziz

1)C code for Deque

2)createDeque function

3)insertRear function

4)deleteRear function

5)getSize function

79.Youssef Taha Saad Taha

1)deleteFront function

2)getFront function

3)isEmpty function

4)display function

5)checkCreation function

6)error function function

56.Mohamed Toukhy Mohamed Ahmed

1)createNode function

2)insertFront function

3)getRear function

4)clear function

41.Ali Hossam Ali Nour

1)C code for application of deque

2)leftArrow function

3)rightArrow function

4)main function

62.Mahmoud Hussein Sayed

1)createBar function

2)addProduct function

3)init function

4)Testing code