ساختمان داده ها

نیم‌سال اول - ۹۸

علوم و تحقیقات تهران

فهرست

جلسه دوم.....۱
جلسه سوم.....۳
جلسه چهارم.....٦
جلسه پنجم.....١٤
جلسه ششم.....١٦
جلسه هفتم.....۱۹
جلسه هشتم.....۲۲
جلسه نهم.....۲۹

جلسه دوم

- نرم‌افزار ← Program + Data + Document
- برنامه ← الگوریتم پیاده‌سازی شده با زبان برنامه نویسی خاص (Java ، C++ و ...)
- الگوریتم ← منطق برنامه (Logic)

★ **نکته:** برای مسائل پیچیده به الگوریتم نیاز داریم.

- **مثال: برنامه بنویسید که ۱۰۰۰ عدد از ورودی بخواند و اولین عدد خوانده شده را در خروجی نمایش دهد.**
  - **توجه:** اینجا از آرایه نیاز نیست استفاده کنیم؛ چون ۱۰۰۰ عدد می‌خوانیم ولی فقط از اولی استفاده می کنیم.
  - بدون استفاده از آرایه:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int first = 0;
        int temp = 0;
        for (int i = 1; i <= 1000; i++) {
            if (i == 1) {
                first = input.nextInt();
                continue;
            }
            temp = input.nextInt();
        }
        System.out.println(first);
    }
}
```

  - با استفاده از آرایه:

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int numbers[] = new int[1000];
        for (int i = 0; i < 1000; i++) {
            numbers[i] = input.nextInt();
        }
        System.out.println(numbers[0]);
    }
}
```

● **نتیجه:** در روش استفاده از آرایه ، ۱۰۰۰ عدد در حافظه ذخیره کردیم ، در حالی که فقط به یک عدد نیاز داریم و ۹۹۹ عدد بدون استفاده خواهند بود و فضای حافظه را اشغال کردیم؛ توجه داشته باشید که هر موقع اعداد بزرگ دیدیم سراغ آرایه نرویم.

★ **نکته:** بسیاری از مسائل پایه یا ابتدایی بدون استفاده از ساختمان‌داده پیچیده قابل حل است.

- **مثال: برنامه بنویسید که با دریافت یک عدد صحیح ، فاکتوریل آن عدد را نمایش دهد.**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int number = scanner.nextInt();
        int factorial = 1;
        for (int i = 1; i <= number; i++) {
            factorial *= i;
        }
        System.out.println(factorial);
    }
}
```

- **مثال: برنامه بنویسید که ۱۰۰۰ عدد از ورودی دریافت کند و حاصل ضرب آن‌ها را در خروجی نمایش دهد.**

• بدون استفاده از آرایه:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int multiply = 1;
        for (int i = 1; i <= 1000;i++){
            multiply *= scanner.nextInt();
        }
        System.out.println(multiply);
    }
}
```

• با استفاده از آرایه:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int numbers[] = new int[1000];
        int multiply = 1;
        for (int i = 0; i < 1000; i++) {
            numbers[i] = scanner.nextInt();
        }
        for (int i = 0; i < numbers.length; i++) {
            multiply *= numbers[i];
        }
        System.out.println(multiply);
    }
}
```

جلسه سوم

Problem (مشکل) ⟶ Conceptual Solution (راه حل مفهومی) ⟶ Algorithm (الگوریتم) ⟶ Program (برنامه)

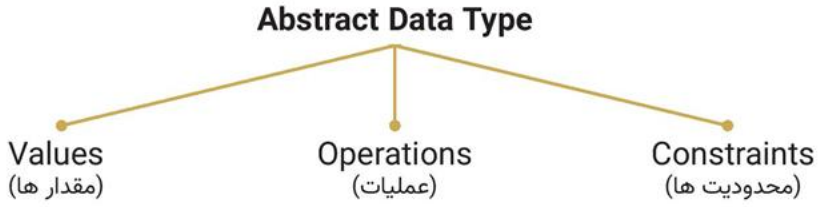
* **الگوریتم:**
راه حل منطقی است که به صورت مجموعه‌ای از گام‌ها ، مراحل کاملا تعریف شده و محدود برای حل یک مسئله است.

* **نکته:** برای حل مسائل نیاز به راه حل مفهومی (Conceptual Solution) یا الگوریتم انتظاعی داریم.
* **نکته:** هر چه مسائل پیچیده تر باشند نیاز به الگوریتم بیشتری داریم.
* **نکته:** در تمامی موارد به الگوریتم نیاز داریم ، ولی بعضی از الگوریتم ها انقدر ساده هستند که بیان نمی‌شود.
* **نکته:** فرق الگوریتم و برنامه در این است که الگوریتم کلی و برنامه وابسته به تکنولوژی است.

- **مثال: برنامه بنویسید که ۱۰۰۰ عدد بگیرد و از آخر به اول آن‌ها را چاپ کند.**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int numbers[] = new int[1000];
        for (int i = 0; i < 1000; i++) {
            numbers[i] = scanner.nextInt();
        }
        for (int i = numbers.length - 1; i >= 0; i--) {
            System.out.println(numbers[i]);
        }
    }
}
```

* **مفاهیم:**
* **Abstract Data Type:**
  - یک مدل ریاضی برای Data Type ها است ، مستقل از اینکه نحوه نمایش آن توسط سخت‌افزار چگونه باشد.
  - **مثال:** اعداد صحیح با مقادیر ∞+ و ∞- با یک سری از عملیات مانند + ، - و ... بدون توجه به نحوه نمایش (نحوه پیاده سازی آن ها) در یک سخت افزار خاص یک ADT می‌باشد.
  - هر ADT از یک سری رفتار (اصول) و همچنین یک سری شرایط تبعیت می‌کند.
    - **مثلا:** اعداد صحیح (int) در یک ساختمان داده می تواند به صورت اعداد دو دویی یا فرمت مکمل دوتایی (Two's compelement) یا هر روش کد گذاری استفاده کرد.
  - **نکته:** ADT ها نه تنها شامل مجموعه مقادیر ، بلکه شامل یکسری عملگرد و محدودیت های تعریف شده می باشد.
  - **جمع بندی:** بنابراین ADT همان مفهوم کلی یا ذهنی ما از مقادیر تعریف شده است. مانند ماتریس ها ، اعداد طبیعی و ... که برای پیاده سازی آن ها در دنیای تکنولوژی نیاز به ساختمان‌داده داریم ، که در زبان‌های برنامه نویسی به آن Data Type گفته می‌شود. در واقع ساختنمان‌داده‌ها فرم فیزیکی و یا پیاده سازی فرم داده هاست.



* نکته: هدف نهایی از طراحی ساختمان داده ها ، فراهم کردن یک سازماندهی و مدیریت داده ها در حجم بالا که امکان عملیاتی مانند جستجو ، حذف ، اضافه و ... با کارایی بالا انجام پذیر باشد.
* نکته: به طور کلی یکی از فاکتور های اصلی طراحی الگوریتم های بهینه ، طراحی ساختمان داده های کارا است و برای طراحی ساختمان داده های کاراتر به تحلیل مسئله نیاز دارد.

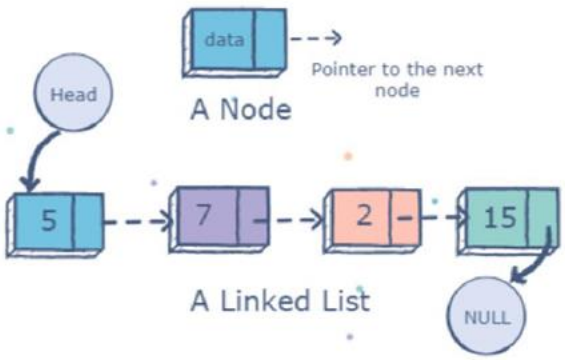
* انواع Data Type:
* Primitive (Built in):
  - مانند: int , float , double , bool , ...
* User-define:
  - این نوع داده ها به صورت نوع داده Primitive استفاده می شود.
  - مانند:

```
struct Employee {
   int id;
   string name;
   string family;
}

Employee employee;
```

- **تمرین: فرض کنید که می‌خواهیم اطلاعات دانشجویان و انتخاب واحد در هر ترم را به صورت یک سری ساختمان‌داده طراحی کنیم ، چه ساختارهایی باید طراحی کنیم؟**

* نکته: روش‌های مختلفی برای پیاده‌سازی یک ADT وجود دارد که بسته به نوع کاربرد ، یکی از آن ها انتخاب می‌شود.
* نکته: در زبان های C++ و Java ، ADT ها از نوع Class پیاده سازی می شوند. ولی در زبانی مانند C که شی گرایی را پشتیبانی نمی‌کند از struct برای پیاده سازی Data Type های جدید بهره می گیرد.
* نکته: زبان C از الگوی رویه‌ای (Procedural) پشتیبانی می‌کند. بنابراین ساختمان‌داده‌ها در زبان C به صورت struct قابل پیاده سازی است. در زبان C++ ، هم از الگوی رویه‌ای و هم از الگوی شی‌گرایی پشتیبانی می کند؛ بنابراین می توان ساختمان‌داده‌ها را به صورت struct ، Class و Data Type طراحی کرد.
* نکته: در زبان Java تنها از زبان مدل شی‌گرایی پشتیبانی می‌کند. لذا ساختمان‌داده‌ها در Java به صورت Data Type یا Class قابل پیاده‌سازی است.

* نکته پیاده‌سازی ساختمان‌داده‌ها در زبان برنامه‌نویسی:
به طور کلی پیاده‌سازی ساختمان‌داده‌ها در هر زبان برنامه‌نویسی مستلزم استفاده از آدرس‌های حافظه است. در ساختمان‌داده‌هایی مانند آرایه‌ها و struct بر اساس نوع اقلام‌داده‌ای عناصر سازنده‌ها ، آدرس‌ها قابل محاسبه خواهد بود. که این نحوه پیاده‌سازی فقط بر اساس محاسبات آدرس است. در مقابل ساختمان‌داده‌های پویا با ذخیره کردن آدرس date item ها درون ساختمان‌داده خود ایجاد می‌شود. (لیست پیوندی)



*** لیست پیوندی:**
برای پیاده کردن این ساختمان داده در C و C++ از نوع داده اشاره‌گر (Pointer) استفاده می‌شود. در حالی که در Java به جای اشاره‌گر از نوع مرجع (Reference Type) استفاده می‌شود.

* **نکته مهم:** با توجه به پیاده‌سازی ساختمان‌داده‌ها معمولا نیاز به نوشتن یک رویه برای ایجاد (Create) و همچنین دستکاری‌داده‌ها (Manipulate) و نمونه (Instance) های یک Data Structure می باشد.
بنابراین طراحی ساختمان‌داده‌ها ، برابر با یک طراحی است که برای اجرای آن یکسری رویه به صورت پیش‌فرض نوشته و اجرا می‌شود.
در زبان های شی‌گرایی به این رویه ها Method گفته می‌شود. و Methodیی که مسئول ساخت ساختمان‌داده در حافظه است ، سازنده (Constructor) نامیده می شود. که حاصل آن در حافظه شئ (Object) نامیده می شود.
بنابراین ساختمان‌داده ها را فرایند مدل‌سازی دنیای بیرون در کامپیوتر می‌توان در نظر گرفت.

جلسه چهارم

* **تمرکز در درس ساختمان داده:**
  * **مصرف حافظه:**
    - امروزه زیاد مورد بحث نیست ، چون همانند گذشته خیلی محدودیت رم نداریم.
  * **پیچیدگی زبان:**
    - معیار ارزیابی (مقایسه کارایی) الگوریتم ها است.

* **مزایای Abstract Data Type:**
  * **Encapsulation:**
    - منظور محصورسازی ، کپسوله کردن ، قراردادن داده‌ها و متدها در قالب یک اسم است.
    - **مزیت:** بدون درگیر شدن به جزییات پیاده‌سازی می‌توان از امکانات موجود از یک کلاس (متدها) استفاده کرد.
      به کمک این در واقع امکاناتی را به زبان برنامه‌نویسی اضافه می‌کنیم که توانایی حل مسئله را بالا می‌برد.
      - **مثال:** کلاس Date یا کلاس Time امکان پردازش داده‌های تاریخ و زمان را به ما می‌دهد.
  * **محدود کردن محل تغییرات برنامه:**
    - زمانی که از یک Abstract Data Type استفاده می‌شود ، در صورتی که نیاز به تغییرات داشته باشد ، کد استفاده کننده همچنان بدون تغییرات ADT خواهد بود.
    - تا زمانی که Interface یا واسط کلاس تغییر نکند ، هرگونه تغییر در پیاده‌سازی ADT ، از نظر کد استفاده کننده مخفی می‌ماند. به این سیاست پنهان‌سازی‌داده‌ها یا اطلاعات می‌گویند.
  * **انعطاف‌پذیری (Flexibility):**
    - به طور کلی هر Abstract Data Type را می‌توان با روش‌های مختلفی پیاده‌سازی کرد.
    - **مثال:** بر اساس آرایه ، لیست پیوندی و ... که از نگاه بیرونی همه دارای قابلیت‌های یکسان هستند.
    - بنابراین کد استفاده کننده (Client Code) همچنان بدون تغییر خواهد بود.

- **تمرین:**
  - کلاس هایی برای:
    1. اعداد مختلط
    2. حساب بانکی
    3. یک نقطه در صفحه دو بعدی
    4. خط برحسب نقطه
    5. دایره روی صفحه (یک نقطه ، یک شعاع)
    6. مثلث (سازنده هایی که بررسی کند امکان تشکیل مثلث وجود دارد و در صورت تشکیل نشدن به سه نقطه پیش‌فرض تشکیل دهد)

    تعریف کنید.

- **تمرین:** کلاسی برای اعداد مختلط تعریف کنید.

```
public class ComplexNumber {

    private double real;
    private double imaginary;

    public ComplexNumber() {
        this.real = 0;
        this.imaginary = 0;
    }

    public ComplexNumber(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }

    public void setRealImaginary(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }

    public void setReal(double real) {
        this.real = real;
    }

    public void setImaginary(double imaginary) {
        this.imaginary = imaginary;
    }

    public double getReal() {
        return real;
    }

    public double getImaginary() {
        return imaginary;
    }

    public ComplexNumber reciprocal() {
        double scale = this.real * this.real + this.imaginary * this.imaginary;
        return new ComplexNumber(this.real / scale, this.imaginary / scale);
    }

    public ComplexNumber plus(ComplexNumber complexNumber) {
        double real = this.real + complexNumber.real;
        double imaginary = this.imaginary + complexNumber.imaginary;
        return new ComplexNumber(real, imaginary);
    }

    public ComplexNumber minus(ComplexNumber complexNumber) {
        double real = this.real - complexNumber.real;
        double imaginary = this.imaginary - complexNumber.imaginary;
        return new ComplexNumber(real, imaginary);
    }
```

```
    public ComplexNumber multiply(ComplexNumber complexNumber) {
        double real = this.real * complexNumber.real - this.imaginary *
complexNumber.imaginary;
        double imaginary = this.real * complexNumber.real + this.imaginary *
complexNumber.imaginary;
        return new ComplexNumber(real, imaginary);
    }

    public ComplexNumber divide(ComplexNumber complexNumber) {
        return this.multiply(complexNumber.reciprocal());
    }

    public ComplexNumber scale(double value) {
        double real = this.real * value;
        double imaginary = this.imaginary * value;
        return new ComplexNumber(real, imaginary);
    }

    public String toString() {
        if (this.imaginary == 0) return String.valueOf(real);
        else if (this.real == 0) return imaginary + "i";
        else if (this.imaginary < 0) return real + " - " + (-this.imaginary) +
"i";
        return this.real + " + " + this.imaginary + "i";
    }

}
```

- **تمرین:** کلاسی برای نقطه در صفحه دوبعدی تعریف کنید.

```
public class Point {

    private double x;
    private double y;

    public Point() {
        this.x = 0;
        this.y = 0;
    }

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public void setXY(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return this.x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return this.y;
    }

    public void setY(double y) {
        this.y = y;
    }

    public double distance(Point point) {
        return Math.sqrt(Math.pow(this.x - point.x, 2) + Math.pow(this.y -
point.y, 2));
    }

    public String toString() {
        return "(" + this.x + ", " + this.y + ")";
    }

}
```

- **تمرین:** کلاسی برای خط برحسب نقطه تعریف کنید.

```
public class Line {

    private Point A;
    private Point B;

    public Line() {
        A.setXY(0 ,0);
        B.setXY(0 ,0);
    }

    public Line(Point A, Point B) {
        if (A == B) {
            throw new IllegalArgumentException("These two points do not form a
line!");
        }
        this.B = B;
        this.A = A;
    }

    public Line(double x1, double y1, double x2, double y2) {
        A.setXY(x1, y1);
        B.setXY(x2, y2);
    }

    public Point getA() {
        return A;
    }

    public void setA(Point A) {
        this.A = A;
    }

    public void setA(double x, double y) {
        this.A = new Point(x, y);
    }

    public Point getB() {
        return B;
    }

    public void setB(Point B) {
        this.B = B;
    }

    public void setB(double x, double y) {
        this.B = new Point(x, y);
    }

    public double lenght() {
        return A.distance(B);
    }

    public String toString() {
        return "Start: " + A.toString() + " , End: " + B.toString();
    }

}
```

- **تمرین:** کلاسی برای دایره روی صفحه (یک نقطه ، یک شعاع) تعریف کنید.

```
public class Circle {

    private Point center;
    private double radius;

    public Circle() {
        this.center = new Point(0 ,0);
        this.radius = 1;
    }

    public Circle(Point center, double radius) {
        this.center = center;
        setRadius(radius);
    }

    public Circle(double x, double y, double radius) {
        this.center = new Point(x, y);
        setRadius(radius);
    }

    public Point getCenter() {
        return this.center;
    }

    public void setCenter(Point center) {
        this.center = center;
    }

    public void setCenter(double x, double y) {
        this.center = new Point(x, y);
    }

    public double getRadius() {
        return this.radius;
    }

    public void setRadius(double radius) {
        if (radius <= 0) {
            System.out.println("The radius must be positive!");
            return;
        }
        this.radius = radius;
    }

    public double getDiameter() {
        return radius * 2;
    }

    public double getCircumference() {
        return Math.PI * getDiameter();
    }

    public double getArea() {
        return Math.PI * radius * radius;
    }
```

```
    public String toString() {
        return "Center: " + center.toString() + " , Radius: " + this.radius;
    }

}
```

- **تمرین: کلاس برای مثلث تعریف کنید.** (سازنده هایی تعریف کنید که امکان تشکیل مثلث را بررسی کند و در صورت تشکیل نشدن با سه نقطه پیش‌فرض تشکیل دهد)

```
public class Triangle {

    private Point A;
    private Point B;
    private Point C;

    public Triangle() {
        this.setDefault();
    }

    public Triangle(Point A, Point B, Point C) {
        if (isValid(A, B, C)) {
            this.A = A;
            this.B = B;
            this.C = C;
        } else {
            this.setDefault();
        }
    }

    public boolean isValid(Point A, Point B, Point C) {
        double side1 = A.distance(B);
        double side2 = A.distance(C);
        double side3 = B.distance(C);

        if (side1 + side2 > side3 || side1 + side3 > side2 || side2 + side3 >
side1)
            return true;
        return false;
    }

    public void setDefault() {
        this.A = new Point(0 ,0);
        this.B = new Point(3 ,0);
        this.C = new Point(0 ,4);
    }

    public double getLenghtAB() {
        return this.A.distance(this.B);
    }

    public double getLenghtAC() {
        return this.A.distance(this.C);
    }

    public double getLenghtBC() {
        return this.B.distance(this.C);
    }

    public String toString() {
        return "A: " + A.toString() + " , B: " + B.toString() + " , C: " +
C.toString();
    }

}
```

جلسه پنجم

*** ADT:**

★ لیست (List):

- **تعریف:** مجموعه مرتبی از اقلام داده‌ای (اشیاء) را نگه می‌دارد.
  - **مثال:** لیست خرید ، لیست دانشجویان ، لیست کارها
- **پیاده‌سازی (User-Define):**
  1. struct:
     - C ، C++ ، ...
  2. class:
     - Java ، C++ ، ...

★ Open Closed Principle(OCP):

بر اساس اصل باز و بسته بودن ، موجودیت های برنامه که شامل کلاس ها، ماژول ها، توابع و... باید برای گسترده‌شدن باز، اما برای تغییرکردن بسته باشند.

● جمع‌بندی:

- پیاده‌سازی یک لیست ADT به کمک آرایه در عملیات(Method) های درج(Insert) یا حذف(Delete) کُند عمل می کند؛ یعنی با هر بار درج یا حذف عنصر باید لیست جابه‌جا شود ، که این تعداد جابه‌جایی(Shift) به موقعیت(Position) عنصر درج یا حذف بستگی دارد.
  - **مثلا:** در بدترین حالت درج یا حذف عنصر به ابتدای لیست بالاترین هزینه را دارد.
- با توجه به اینکه آرایه یک ساختمان‌داده ایستا (سایز آرایه باید در زمان طراحی مشخص شود و در زمان اجرا(Runtime) ثابت بماند!) است ، برای لیست‌هایی که در زمان اجرا نیاز به حافظه یا ظرفیت بیشتر داشته باشند ، روش آرایه مناسب نیست؛ بنابراین برای لیست‌هایی که نیاز به حافظه پویا(Dynamic) دارند ، از روش لیست‌های پیوندی استفاده می‌شود.

★ سوال: لیست اعداد صحیح را به کمک آرایه پیاده‌سازی کنید.

```
public class IntList {

    private int[] elements;
    private int size;

    public IntList(int length) {
        elements = new int[length];
    }

    public void add(int element) {
        if (isFull())
            return;

        elements[size++] = element;
    }

    public void add(int index, int element) {
        if (!isIndexValid(index) || isFull())
            return;

        for (int i = size; i > index; i--)
            elements[i] = elements[i - 1];

        elements[index] = element;
        size++;
    }
```

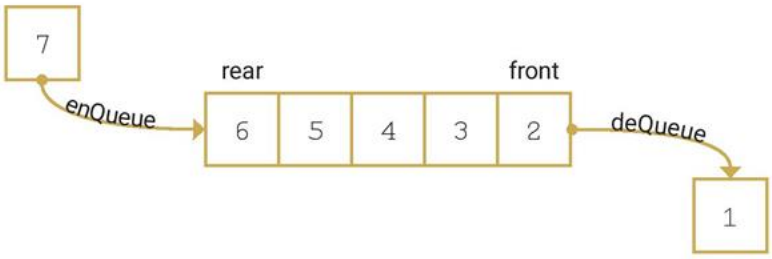
```
    public int remove(int index) {
        if (!isIndexValid(index) || isEmpty())
            return -1;

        int element = elements[index];
        for (int i = index; i < size - 1; i++)
            elements[i] = elements[i + 1];

        size--;
        return element;
    }

    public int get(int index) {
        if (!isIndexValid(index))
            return -1;
        return elements[index];
    }

    public void set(int index, int element) {
        if (!isIndexValid(index))
            return;
        elements[index] = element;
    }

    public void print() {
        for (int i = 0; i < size; i++)
            System.out.println(elements[i]);
    }


    public void reversePrint() {
        if (isEmpty())
            return;

        for (int i = size - 1; i >= 0; i--)
            System.out.println(elements[i]);
    }

    public boolean isIndexValid(int index) {
        return index >= 0 && index <= size;
    }

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return size() == 0;
    }

    public boolean isFull() {
        return size() == elements.length;
    }
}
```

جلسه ششم

*** ADT:**

★ صف (Queue):

- **تعریف:** مجموعه‌ای از اقلام داده‌ای (اشیاء) را بر اساس اصل **First In First Out** (خروج به ترتیب ورود) نگه می‌دارد.



★ سوال: صف را به کمک آرایه پیاده‌سازی کنید.

```
public class Queue {

    private int[] elements;
    private int front;
    private int rear;

    public Queue(int length) {
        elements = new int[length];
        front = 0;
        rear = 0;
    }

    public void enQueue(int element) {
        if (isFull()) {
            System.out.println("Queue is full!");
            return;
        }

        elements[rear++] = element;
    }

    public int deQueue() {
        if (isEmpty()) {
            System.out.println("Queue is empty!");
            return -1;
        }

        int element = elements[front];
        for (int i = 0; i < rear - 1; i++)
            elements[i] = elements[i + 1];
        rear--;
        return element;
    }
```

```
    public void print() {
        if (isEmpty()) {
            System.out.println("Queue is empty!");
            return;
        }

        for (int i = front; i < rear; i++)
            System.out.println(elements[i]);
    }

    public int size() {
        return rear;
    }

    public boolean isEmpty() {
        return front == rear;
    }

    public boolean isFull() {
        return rear == elements.length;
    }

}
```

* **سوال:** صف را به کمک لیست پیاده‌سازی کنید. (راهنمایی: به کمک اعمال محدودیت‌هایی بر روی لیست)

```
public class QueueIntList {

    private IntList intList;

    public QueueIntList(int length) {
        intList = new IntList(length);
    }

    public void enQueue(int data) {
        if (intList.isFull()) {
            System.out.println("Queue is full!");
            return;
        }

        intList.add(data);
    }

    public int deQueue() {
        if (intList.isEmpty()) {
            System.out.println("Queue is empty!");
            return -1;
        }

        return intList.remove(0);
    }

    public void print() {
        intList.print();
    }

    public int size() {
        return intList.size();
    }
```

```
    public boolean isEmpty() {
        return intList.isEmpty();
    }

    public boolean isFull() {
        return intList.isFull();
    }

}
```

★ **تمرین:** خروجی قطعه کد زیر را به کمک صف بدست آورید.

```
Queue queue = new Queue(5);
int i = 1;
while (!queue.isFull()) {
    queue.enQueue(i);
    if (i == 12)
        break;
    if (i >= 5)
        queue.deQueue();
    i++;
}
queue.print();
```

Output:

```
8
9
10
11
12
```

i	queue											
1	1											
2	1	2										
3	1	2	3									
4	1	2	3	4								
5	1	2	3	4	5							
6	1	2	3	4	5	6						
7	1	2	3	4	5	6	7					
8	1	2	3	4	5	6	7	8				
9	1	2	3	4	5	6	7	8	9			
10	1	2	3	4	5	6	7	8	9	10		
11	1	2	3	4	5	6	7	8	9	10	11	
12	1	2	3	4	5	6	7	8	9	10	11	12

● **جمع‌بندی:**

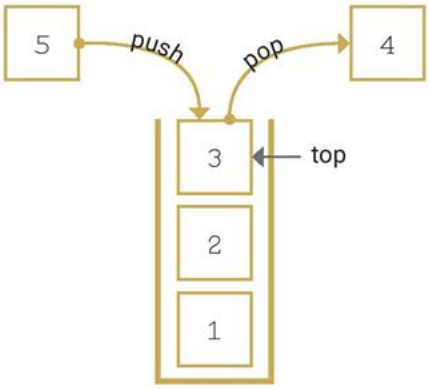
در این پیاده‌سازی ، اضافه کردن به صف تنها با یک عمل مقدارگذاری در انتهای صف قابل انجام است؛ در حالی که حذف عنصر از ابتدای صف در بدترین حالت معادل با n عمل (جابه‌جایی که عناصر صف) می باشد. بنابراین برای پیاده‌سازی ساختمان‌داده صف بهتر است به جای استفاده از آرایه از لیست پیوندی استفاده کنیم.

جلسه هفتم

*** ADT:**

★ پشته (Stack):

- **تعریف:** مجموعه‌ای از اقلام داده‌ای (اشیاء) را بر اساس اصل **Last In First Out** (اولویت خروج با عناصر تازه وارد) نگه می‌دارد.



★ سوال: پشته را به کمک آرایه پیاده‌سازی کنید.

```
public class Stack {

    private int[] elements;
    private int top;

    public Stack() {
        initialize(100);
    }

    public Stack(int length) {
        initialize(length);
    }

    private void initialize(int length) {
        elements = new int[length];
        top = -1;
    }

    public boolean push(int element) {
        if (isFull()) {
            System.out.println("Stack is full!");
            return false;
        }

        elements[++top] = element;
        return true;
    }
```

```
    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return -1;
        }

        return elements[top];
    }

    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return -1;
        }

        return elements[top--];
    }

    public void print() {
        for (int i = top; i >= 0; i--)
            System.out.println(elements[i]);
    }

    public int size() {
        return top + 1;
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public boolean isFull() {
        return size() == elements.length;
    }

}
```

* **سوال:** به کمک لیست ، پشته را پیاده‌سازی کنید. (راهنمایی: به کمک اعمال محدودیت‌هایی بر روی لیست)

```
public class StackIntList {

    private IntList intList;

    public StackIntList() {
        this.intList = new IntList(100);
    }

    public StackIntList(int length) {
        this.intList = new IntList(length);
    }

    public boolean push(int data) {
        if (isFull()) {
            System.out.println("Stack is full!");
            return false;
        }

        intList.add(data);
        return true;
    }
```

```
    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return 1-;
        }
        return intList.get(size() - 1);
    }

    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return -1;
        }

        return intList.remove(size() - 1);
    }

    public void print() {
        intList.reversePrint();
    }

    public int size() {
        return intList.size();
    }

    public boolean isEmpty() {
        return intList.isEmpty();
    }

    public boolean isFull() {
        return intList.isFull();
    }

}
```

★ **تمرین:** خروجی قطعه کد زیر را به کمک پشته بدست آورید.

```
Stack stack = new Stack();
for (int i = 0; i < 10; i++) {
    stack.push(i);
}
for (int i = 1; i <= 5; i++) {
    stack.pop();
}
stack.print();
```

Output:

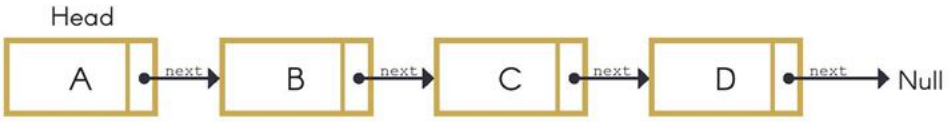
```
4
3
2
1
0
```

جلسه هشتم

*** ADT:**

★ لیست پیوندی(Linked List):

- **تعریف:** مجموعه‌ای از اقلام داده‌ای (اشیاء) است که در آن هر عنصر به عنصر بعدی خود اشاره می‌کند. این عناصر در کنار یکدیگر تشکیل یک دنباله می‌دهند.



- **تمرین:**

۱. با در نظر گرفتن تعداد عناصر برای لیست‌پیوندی متد size را پیاده‌سازی کنید.
۲. متد set و isFull را برای لیست‌پیوندی پیاده‌سازی کنید.
۳. در لیست‌پیوندی یک سوی ، جهت تمام پیوندها را معکوس کنید.
٤. به کمک تابع بازگشتی متد print را در لیست پیوندی پیاده‌سازی کنید.
۵. تابع معکوس‌کردن لینک‌های یک لیست را به صورت بازگشتی بنویسید.

- **سوال:** لیست‌پیوندی را به همراه تمرینات اعلام‌شده پیاده‌سازی کنید.

```
public class SinglyLinkedList {

    class Node {
        private Object data;
        private Node next;

        public Node(Object data) {
            this.data = data;
        }
    }

    private Node head;
    private Node tail;
    private int size;

    public void addFirst(Object data) {
        Node node = new Node(data);
        if (isEmpty()) {
            head = node;
            tail = node;
        } else {
            node.next = head;
            head = node;
        }
        size++;
    }

    public void addLast(Object data) {
        Node node = new Node(data);
        if (isEmpty()) {
            head = node;
            tail = node;
        } else {
            tail.next = node;
            tail = node;
        }
        size++;
    }

    public void add(int index, Object data) {
        if (index < 0 || index > size)
            return;
        Node node = new Node(data);
        if (index == 0) {
            addFirst(data);
            return;
        }
        Node prev = find(index - 1);
        Node current = prev.next;
        node.next = current;
        prev.next = node;
        if (index == size) {
            tail = node;
        }
        size++;
    }
```

```
public Object removeFirst() {
    return remove(0);
}

public Object removeLast() {
    return remove(size - 1);
}

public Object remove(int index) {
    if (index < 0 || index >= size || isEmpty())
        return null;

    if (index == 0) {
        Object data = head.data;
        head = head.next;
        if (size == 1)
            tail = null;
        size--;
        return data;
    }

    Node prev = find(index - 1);
    Node current = prev.next;
    Node next = current.next;
    prev.next = next;

    if (index == size - 1) {
        tail = prev;
    }

    size--;
    return current.data;
}

public void set(int index, Object data) {
    Node node = find(index);
    if (node == null)
        return;
    node.data = data;
}

public Object get(int index) {
    Node node = find(index);
    if (node == null)
        return null;
    return node.data;
}

public void print() {
    Node temp = head;
    while (temp != null) {
        System.out.println(temp.data);
        temp = temp.next;
    }
}
```

```
private void printRecursivly(Node node) {
    System.out.println(node.data);
    if (node.next != null)
        printRecursivly(node.next);
}

public void printRecursivly() {
    printRecursivly(head);
}
```

```
public SinglyLinkedList reverse() {
    SinglyLinkedList reverseList = new SinglyLinkedList();
    Node temp = head;
    while (temp != null) {
        reverseList.addFirst(temp.data);
        temp = temp.next;
    }
    return reverseList;
}
```

```
private Node reverseRecursively(Node node) {
    if ((node.next == null))
        return node;
    Node newHead = reverseRecursively(node.next);
    node.next.next = node;
    node.next = null;
    return newHead;
}

public void reverseRecursively() {
    head = reverseRecursively(head);
}
```

```
public CircularDoublyLinkedList convertToCircularDoublyLinkedList() {
                CircularDoublyLinkedList circularDoublyLinkedList = new
CircularDoublyLinkedList();
    Node temp = head;
    while (temp != null) {
        circularDoublyLinkedList.addLast(temp.data);
        temp = temp.next;
    }
    return circularDoublyLinkedList;
}

private Node find(int index) {
    if (index < 0 || index >= size || isEmpty())
        return null;

    Node temp = head;
    for (int i = 0; i < index; i++)
        temp = temp.next;
    return temp;
}
```

```
    private Node findTail() {
        if (isEmpty())
            return null;

        Node temp = head;
        while (temp.next != null)
            temp = temp.next;
        return temp;
    }

    public void clear() {
        head = null;
        tail = null;
        size = 0;
    }

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return head == null;
    }

    public boolean isFull() {
        return false;
    }

}
```

- **تمرین:** صف را به کمک لیست‌پیوندی پیاده‌سازی کنید.

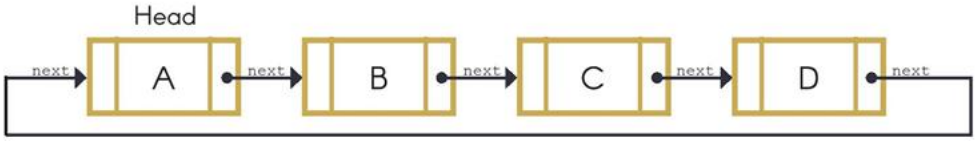
```
public class QueueLinkedList {

    private SinglyLinkedList linkedList;

    public QueueLinkedList() {
        linkedList = new SinglyLinkedList();
    }

    public void enQueue(Object data) {
        linkedList.addLast(data);
    }

    public Object deQueue() {
        return linkedList.remove(0);
    }

    public void print() {
        linkedList.print();
    }

    public int size() {
        return linkedList.size();
    }

    public boolean isEmpty() {
        return linkedList.isEmpty();
    }

    public boolean isFull() {
        return linkedList.isFull();
    }

}
```

* **تمرین:** پشته را به کمک لیست‌پیوندی پیاده‌سازی کنید.

```
public class StackLinkedList {

    private SinglyLinkedList linkedList;

    public StackLinkedList() {
        this.linkedList = new SinglyLinkedList();
    }

    public boolean push(int data) {
        linkedList.addFirst(data);
        return true;
    }

    public Object peek() {
        return linkedList.get(0);
    }

    public Object pop() {
        return linkedList.remove(0);
    }

    public void print() {
        linkedList.print();
    }

    public int size() {
        return linkedList.size();
    }

    public boolean isEmpty() {
        return linkedList.isEmpty();
    }

    public boolean isFull() {
        return linkedList.isFull();
    }

}
```

جلسه نهم

* **لیست پیوندی:**
* **لیست چرخشی (Circular Signly Linked List):**



* **نکته:** برای ساختمان‌داده‌ها پیاده‌سازی های مختلفی وجود دارد.
  - **مثال:** در یک لیست‌پیوندی می‌توان آدرس ابتدای لیست ، تعداد عناصر و آدرس انتهای لیست را نگه داریم؛ یا اینکه یک Node اولیه که حاوی اطلاعاتی از لیست است را ایجاد کنیم. (موارد گفته‌شده در Constructor پیاده‌سازی می‌شود)
  - **مثال:** در لیست چرخشی برای اضافه کردن یک Node به ابتدای لیست ، نیاز به آدرس انتهای لیست داریم ، اگر این آدرس در ساختمان‌داده‌ها یا کلاس‌ها تعریف نشده باشد با پیچیدگی زمانی O(n) مجبور به یافتن انتهای لیست خواهیم بود.

- **تمرین:**
  ۱. بدون داشتن tail ، آدرس انتهای لیست را بدست آورید.
  ۲. متُدی برای کپی کردن عناصر یک لیست در لیست‌پیوندی چرخشی بنویسید.

- **سوال: لیست‌پیوندی چرخشی را به همراه تمرینات اعلام‌شده پیاده‌سازی کنید.**
-

```
public class CircularSinglyLinkedList {

    class Node {
        private Object data;
        private Node next;

        public Node(Object data) {
            this.data = data;
        }
    }

    private Node head;
    private Node tail;
    private int size;

    public void addFirst(Object data) {
        Node node = new Node(data);
        if (isEmpty()) {
            head = node;
            node.next = head;
            tail = node;
        } else {
            node.next = head;
            head = node;
            tail.next = node;
        }
        size++;
    }
```

```
public void addLast(Object data) {
    Node node = new Node(data);
    if (isEmpty()) {
        head = node;
        node.next = head;
        tail = node;
    } else {
        tail.next = node;
        node.next = head;
        tail = node;
    }
    size++;
}

public void add(int index, Object data) {
    if (index < 0 || index > size)
        return;

    Node node = new Node(data);

    if (index == 0) {
        addFirst(data);
        return;
    }

    Node prev = find(index - 1);
    Node current = prev.next;
    node.next = current;
    prev.next = node;

    if (index == size) {
        tail = node;
        node.next = head;
    }

    size++;
}

public Object removeFirst() {
    return remove(0);
}

public Object removeLast() {
    return remove(size - 1);
}

public void set(int index, Object data) {
    Node node = find(index);
    if (node == null)
        return;
    node.data = data;
}

public Object get(int index) {
    Node node = find(index);
    if (node == null)
        return null;
    return node.data;
}
```

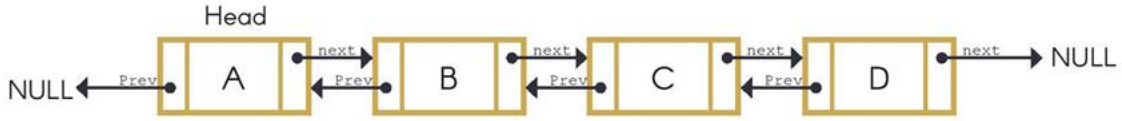
```
public Object remove(int index) {
    if (index < 0 || index >= size || isEmpty())
        return null;

    if (index == 0) {
        Object data = head.data;
        head = head.next;
        tail.next = head;
        if (size == 1) {
            tail = null;
            head = null;
        }
        size--;
        return data;
    }

    Node prev = find(index - 1);
    Node current = prev.next;
    Node next = current.next;
    prev.next = next;

    if (index == size - 1) {
        tail = prev;
        tail.next = head;
    }

    size--;
    return current.data;
}

public void print() {
    Node temp = head;
    for (int i = 0; i < size; i++) {
        System.out.println(temp.data);
        temp = temp.next;
    }
}

public CircularSinglyLinkedList copy() {
    CircularSinglyLinkedList copyList = new CircularSinglyLinkedList();
    Node temp = head;
    for (int i = 0; i < size; i++) {
        copyList.addLast(temp.data);
        temp = temp.next;
    }
    return copyList;
}

private Node find(int index) {
    if (index < 0 || index >= size || isEmpty())
        return null;

    Node temp = head;
    for (int i = 0; i < index; i++)
        temp = temp.next;
    return temp;
}
```

```
    private Node findTail() {
        if (isEmpty())
            return null;

        Node temp = head;
        while (temp.next != head)
            temp = temp.next;
        return temp;
    }

    public int size() {
        return size;
    }

    public void clear() {
        head = null;
        tail = null;
        size = 0;
    }

    public boolean isEmpty() {
        return head == null;
    }

    public boolean isFull() {
        return false;
    }

}
```

- **سوال:** چرا متُد findTail را به صورت private در نظر می‌گیریم؟
  - با توجه به اصل پنهان‌سازی داده‌ها (Encapsulation) متُد findTail آدرس انتهای لیست را باز می‌گرداند (از نوع Node) ، که این Node یک ارجاع (Reference) به کلاس‌داخلی Node است. درحالی که در پیاده‌سازی ساختمان‌داده‌ها ، نمایش داخلی (Representation) ساختمان‌داده‌ها برای Client Code دیده شود.
    به طور کلی تمام متُدهایی که به پیاده‌سازی داخلی یک ساختمان‌داده‌ها اشاره می کند ، باید دسترسی محدود private داده شود.

★ **نکته:** برای ثابت ماندن Client Code از تغییرات پیاده‌سازی ساختمان‌داده‌ها ، باید Interface یا واسط برای تمام ساختمان‌داده‌های ساخته شده در Client Code ثابت بماند؛ بنابراین به جای اینکه کدهای نوشته شده ، وابسطه به کلاس‌های پیاده‌سازی شده باشد ، باید وابسطه به Interface باشد.
- Interface قراردادی بین استفاده کننده از یک ساختمان‌داده و توسعه‌دهنده می باشد

* لیست پیوندی:
* لیست دوطرفه(Doubly Linked List):



- **تمرین:** از یک لیست‌پیوندی دوطرفه داده‌شده ، یک لیست پیوندی یک طرفه بسازید.
- **سوال:** لیست‌پیوندی دوطرفه را به همراه تمرین اعلام‌شده پیاده‌سازی کنید.

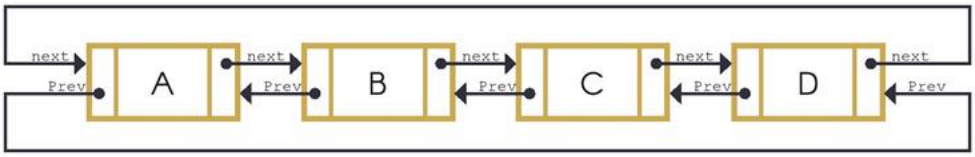
```
public class DoublyLinkedList {

    class Node {
        private Object data;
        private Node prev;
        private Node next;

        public Node(Object data) {
            this.data = data;
        }
    }

    private Node head;
    private Node tail;
    private int size;

    public void addFirst(Object data) {
        Node node = new Node(data);
        if (isEmpty()) {
            head = node;
            tail = node;
        } else {
            node.next = head;
            head.prev = node;
            head = node;
        }
        size++;
    }

    public void addLast(Object data) {
        Node node = new Node(data);
        if (isEmpty()) {
            head = node;
            tail = node;
        } else {
            tail.next = node;
            node.prev = tail;
            tail = node;
        }
        size++;
    }
```

```
public void add(int index, Object data) {
    if (index < 0 || index > size)
        return;

    Node node = new Node(data);

    if (index == 0) {
        addFirst(data);
        return;
    }

    Node prev = find(index - 1);
    Node current = prev.next;
    prev.next = node;
    node.prev = prev;
    node.next = current;
    if (current != null)
        current.prev = node;

    if (index == size)
        tail = node;

    size++;
}

public Object removeFirst() {
    if (isEmpty())
        return null;
    Object data = head.next;
    head = head.next;
    if (head != null)
        head.prev = null;
    else
        tail = null;
    size--;
    return data;
}

public Object removeLast() {
    if (isEmpty())
        return null;
    Object data = tail.data;
    tail = tail.prev;
    if (tail != null)
        tail.next = null;
    else
        head = null;
    size--;
    return data;
}
```

```
public Object remove(int index) {
    if (index < 0 || index >= size || isEmpty())
        return null;

    if (index == 0) {
        Object data = head.data;
        head = head.next;
        if (size == 1)
            tail = null;
        else
            head.prev = null;

        size--;
        return data;
    }

    Node current = find(index);
    Node prev = current.prev;
    Node next = current.next;
    prev.next = next;
    if (next != null)
        next.prev = prev;

    if (index == size - 1) {
        tail = prev;
    }

    size--;
    return current.data;
}

public void set(int index, Object data) {
    Node node = find(index);
    if (node == null)
        return;
    node.data = data;
}

public Object get(int index) {
    Node node = find(index);
    if (node == null)
        return null;
    return node.data;
}

public void print() {
    Node temp = head;
    while (temp != null) {
        System.out.println(temp.data);
        temp = temp.next;
    }
}
```

```
    public SinglyLinkedList convertToSinglyLinkedList() {
        SinglyLinkedList linkedList = new SinglyLinkedList();
        Node temp = head;
        while (temp != null) {
            linkedList.addLast(temp.data);
            temp = temp.next;
        }
        return linkedList;
    }

    private Node find(int index) {
        if (index < 0 || index >= size || isEmpty())
            return null;

        Node temp = head;
        for (int i = 0; i < index; i++)
            temp = temp.next;
        return temp;
    }

    private Node findTail() {
        if (isEmpty())
            return null;

        Node temp = head;
        while (temp.next != null)
            temp = temp.next;
        return temp;
    }

    public int size() {
        return size;
    }

    public void clear() {
        head = null;
        tail = null;
        size = 0;
    }

    public boolean isEmpty() {
        return head == null;
    }

    public boolean isFull() {
        return false;
    }

}
```

* لیست پیوندی:
* لیست دوطرفه چرخشی(Circular Doubly Linked List):



- **سوال:** لیست‌پیوندی دوطرفه چرخشی را پیاده‌سازی کنید.

```
public class CircularDoublyLinkedList {

    class Node {
        private Object data;
        private Node prev;
        private Node next;

        public Node(Object data) {
            this.data = data;
        }
    }

    private Node head;
    private Node tail;
    private int size;

    public void addFirst(Object data) {
        Node node = new Node(data);
        if (isEmpty()) {
            head = node;
            tail = node;
            node.prev = node;
            node.next = node;
        } else {
            node.next = head;
            head.prev = node;
            head = node;
            head.prev = tail;
            tail.next = head;
        }
        size++;
    }

    public void addLast(Object data) {
        Node node = new Node(data);
        if (isEmpty()) {
            head = node;
            tail = node;
            node.prev = node;
            node.next = node;
        } else {
            node.prev = tail;
            tail.next = node;
            node.next = head;
            tail = node;
            head.prev = tail;
        }
        size++;
    }
```

```
public void add(int index, Object data) {
    if (index < 0 || index > size)
        return;

    Node node = new Node(data);

    if (index == 0) {
        addFirst(data);
        return;
    }

    Node prev = find(index - 1);
    Node current = prev.next;
    prev.next = node;
    node.prev = prev;
    node.next = current;
    if (current != null)
        current.prev = node;

    if (index == size) {
        tail = node;
        tail.next = head;
        head.prev = tail;
    }

    size++;
}

public Object removeFirst() {
    if (isEmpty())
        return null;
    Object data = head.data;
    head = head.next;
    head.prev = tail;
    tail.next = head;
    if (size == 1) {
        head = null;
        tail = null;
    }
    size--;
    return data;
}

public Object removeLast() {
    if (isEmpty())
        return null;
    Object data = tail.data;
    tail = tail.prev;
    tail.next = head;
    head.prev = tail;
    if (size == 1) {
        head = null;
        tail = null;
    }
    size--;
    return data;
}
```

```
public Object remove(int index) {
    if (index < 0 || index >= size || isEmpty())
        return null;

    if (index == 0) {
        Object data = head.data;
        head = head.next;
        head.prev = tail;
        tail.next = head;
        if (head == tail) {
            head = null;
            tail = null;
        }
        size--;
        return data;
    }

    Node current = find(index);
    Node prev = current.prev;
    Node next = current.next;
    prev.next = next;
    if (next != null)
        next.prev = prev;

    if (index == size - 1) {
        tail = prev;
        tail.next = head;
        head.prev = tail;
    }

    size--;
    return current.data;
}

public void set(int index, Object data) {
    Node node = find(index);
    if (node == null)
        return;
    node.data = data;
}

public void print() {
    Node temp = head;
    for (int i = 0; i < size; i++) {
        System.out.println(temp.data);
        temp = temp.next;
    }
}

public Object get(int index) {
    Node node = find(index);
    if (node == null)
        return null;
    return node.data;
}
```

```
    private Node find(int index) {
        if (index < 0 || index >= size || isEmpty())
            return null;

        Node temp = head;
        for (int i = 0; i < index; i++)
            temp = temp.next;
        return temp;
    }

    private Node findTail() {
        if (isEmpty())
            return null;

        Node temp = head;
        while (temp.next != head)
            temp = temp.next;
        return temp;
    }

    public int size() {
        return size;
    }

    public void clear() {
        head = null;
        tail = null;
        size = 0;
    }

    public boolean isEmpty() {
        return head == null;
    }

    public boolean isFull() {
        return false;
    }

}
```

نیم‌سال اول - ۹۸

علوم و تحقیقات تهران