

**SVEUČILIŠTE U SPLITU**  
**FAKULTET ELEKTROTEHNIKE,**  
**STROJARSTVA I BRODOGRADNJE**

**Algoritmi i strukture podataka**

**Vježba 4.**

*Lists*

Ivan Banovac, 220

Datum izvođenja vježbe:

08.12.2025



## Zadatak 1. – Single Linked Lists

Node.cs:

```
using System;
namespace SingleList
{
    public class Node
    {
        public object Element { get; private set; }
        public Node Next { get; set; }

        public Node(object element, Node next)
        {
            Element = element;
            Next = next;
        }
    }
}
```

List.cs:

```
using System;
namespace SingleList
{
    public class List
    {
        private Node head;
        private Node tail;
        public List()
        {
            head = null;
            tail = null;
        }
        public bool IsEmpty()
        {
            return head == null;
        }
        public void InsertFront(object data)
        {
            Node newNode = new Node(data, head);
            head = newNode;
            if (tail == null)
            {
                tail = head;
            }
        }
        public void InsertEnd(object data)
        {
            if (IsEmpty())
            {
                InsertFront(data);
            }
            else
            {
                Node newNode = new Node(data, null);
                tail.Next = newNode;
                tail = newNode;
            }
        }
        public object RemoveFront()
        {
            if (IsEmpty())
                throw new Exception("Prazna");
            object data = head.Element;
            head = head.Next;
            if (head == null)
            {
                tail = null;
            }
            return data;
        }
        public object RemoveEnd()
        {
            if (IsEmpty())
                throw new Exception("Prazna");
            object data = tail.Element;

            if (head == tail)
            {
                head = null;
                tail = null;
            }
            else
```

Program.cs:

```
using System;

namespace SingleList
{
    class Program
    {
        static void Main(string[] args)
        {
            List list = new List();
            list.InsertFront("Ivo");
            list.InsertEnd(1);
            list.InsertEnd(3.14);
            list.InsertFront(true);
            list.InsertEnd(100m);

            list.Display();

            try
            {
                list.RemoveEnd();
                list.RemoveFront();

                list.Display();

                list.RemoveEnd();
                list.RemoveFront();
                list.RemoveFront();
            }
            catch (Exception x)
            {
                Console.WriteLine(x.Message);
            }

            list.Display();
            Console.ReadKey();
        }
    }
}
```

Klasa **Node** služi kao osnovna jedinica gdje - *Element* omogućuje pohranu podataka bilo kojeg tipa unutar samog čvora, *Next* djeluje kao poveznica (referenca) na idući čvor u nizu a *konstruktoru se* prilikom stvaranja novog čvora, inicijaliziraju njegova vrijednost i pokazivač na sljedeći element.

Klasa **List** sadrži logiku za kontrolu elemenata putem pokazivača *head* i *tail*. Provjera stanja se radi pomoću metode *IsEmpty()* - utvrđuje je li lista prazna provjeravajući postojanje početnog čvora (*head*). Umetanje podataka se izvodi metodom *InsertFront* koja postavlja novi element na sam početak, dok *InsertEnd* dodaje element na kraj. *RemoveFront* i *RemoveEnd* služe za brisanje elemenata s početka ili kraja, uz povrat obrisane vrijednosti i prilagodbu strukture liste. Sadržaj prikazujemo metodom *Display* koja prolazi kroz cijelu listu i ispisuje sve pohranjene objekte u konzolu.

Unutar **Main-a** vrši se provjera ispravnosti implementiranih metoda nad objektom *list*. Program dodaje različite tipove podataka (string, int, double, decimal, bool), uklanja ih te ispisuje rezultate i eventualne poruke o pogreškama u konzolu

Rezultat:

```
True
Ivo
1
3,14
100
Ivo
1
3,14
```

## Zadatak 2. – Double Linked Lists

Za razliku od jednostrukih liste, čvorovi u dvostruko povezanoj listi (klasa *Node*) posjeduju reference i na sljedeći (*Next*) i na prethodni (*Prev*) element.

Node.cs:

```
using System;

namespace DoubleList
{
    public class Node
    {
        public object Element { get; private set; }
        public Node Next { get; set; }
        public Node Prev { get; set; }
        public Node()
        {
            Element = null;
            Next = null;
            Prev = null;
        }
        public Node(object element, Node prev, Node next)
        {
            Element = element;
            Prev = prev;
            Next = next;
        }
    }
}
```

### Student.cs:

```
using System;
namespace DoubleList
{
    public class Student
    {
        private string name;
        private double grade;
        public Student(string name, double grade)
        {
            this.name = name;
            this.grade = grade;
        }
        public override string ToString()
        {
            return name + grade;
        }
        public override bool Equals(object obj)
        {
            if (obj == null || GetType() != obj.GetType())
            {
                return false;
            }
            Student other = (Student)obj;
            return name == other.name && grade == other.grade;
        }
        public override int GetHashCode()
        {
            return name.GetHashCode() ^ grade.GetHashCode();
        }
        public static bool operator ==(Student a, Student b)
        {
            if (ReferenceEquals(a, null))
            {
                return ReferenceEquals(b, null);
            }
            return a.Equals(b);
        }
        public static bool operator !=(Student a, Student b)
        {
            return !(a == b);
        }
    }
}
```

### Program.cs:

```
using System;
namespace DoubleList
{
    class Program
    {
        static void Main(string[] args)
        {
            BllList list = new BllList();
            Student a = new Student("Tom", 4.7);
            Student b = new Student("Kim", 4.3);
            if (a == b)
                Console.WriteLine("Jednaki");
            else
                Console.WriteLine("Nisu jednaki");
            list.Append(a);
            list.Append(b);
            list.Append(new Student("Mia", 4.2));
            list.Append(new Student("Tea", 4.4));
            list.Append(new Student("Lea", 4.1));
            Student c = new Student("Kim", 4.3);
            list.Append(c);
            list.Append(new Student("Bob", 4.5));
            list.Append(new Student("Ena", 4.9));
            Console.WriteLine("Count: " + list.Count);
            list.Display();
            Console.WriteLine();
            if (list.Search(c))
                Console.WriteLine("Element je u listi");
            else
                Console.WriteLine("Element nije u listi");
            try
            {
                list.Remove(a);
                list.Remove(c);
            }
            catch (Exception x)
            {
                Console.WriteLine(x.Message);
            }
            Console.WriteLine("Count: " + list.Count);
            list.Display();
        }
    }
}
```

BllList:

```
using System;

namespace DoubleList
{
    public class BllList
    {
        private Node head;
        private int count;
        public int Count
        {
            get { return count; }
        }
        public BllList()
        {
            head = new Node();
            head.Next = head;
            head.Prev = head;
            count = 0;
        }
        private bool IsEmpty()
        {
            return count == 0;
        }
        public void Append(object data)
        {
            Node last = head.Prev;
            Node newNode = new Node(data, last, head);
            last.Next = newNode;
            head.Prev = newNode;
            count++;
        }
        private Node Find(object data)
        {
            Node current = head.Next;
            while (current != head)
            {
                if (current.Element.Equals(data))
                {
                    return current;
                }
                current = current.Next;
            }
            return null;
        }
        public void Remove(object data)
        {
            Node nodeToDelete = Find(data);
            if (nodeToDelete != null)
            {
                nodeToDelete.Prev.Next = nodeToDelete.Next;
                nodeToDelete.Next.Prev = nodeToDelete.Prev;
                count--;
            }
            else
            {
                throw new Exception("Element not found");
            }
        }
        public bool Search(object data)
        {
            return Find(data) != null;
        }
        public void Display()
```

*Append* metodom novi čvor se umeće na kraj liste, pri čemu se automatski ažuriraju pokazivači susjednih čvorova i inkrementira brojač *count*. Pretraživanje elemenata izvodimo metodom *Find* koja sekvencijalno prolazi listom tražeći podatak, dok *Search* koristi tu logiku kako bi vratila informaciju o postojanju elementa. Metodom *Remove* smanjujemo ukupni broj elemenata tako da premostimo traženi element unutar liste.

Unutar klase **Student** definiramo metode *Equals* - definira kriterij jednakosti dva studenta prema imenu i ocjeni, *GetHashCode* - generira hash vrijednost temeljenu na istim parametrima, te operatore `==` i `!=` radi lakše usporedbe u kodu.

Rezultat:

```
Nisu jednaki
Count: 8
Tom4,7
Kim4,3
Mia4,2
Tea4,4
Lea4,1
Kim4,3
Bob4,5
Ena4,9

Element je u listi
Count: 6
Mia4,2
Tea4,4
Lea4,1
Kim4,3
Bob4,5
Ena4,9
```