BÀI TẬP THỰC HÀNH
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Thời gian: 30 tiết

Phân bổ: 03 tiết/tuần

Ngôn ngữ lập trình: C# theo phương pháp lập trình hướng đối tượng

Biên soạn: Trần Minh Thái

Lưu ý:

1. Đánh giá sinh viên theo kết quả thực hiện bài tập của mỗi buổi thực hành
2. Những bài tập có đánh (*) dành cộng điểm khuyến khích cho những sinh viên hoàn thành tại lớp thực hành

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td>1</td><td colspan="2">Mục tiêu: Ôn tập kỹ thuật cơ bản về phương pháp lập trình hướng đối tượng</td></tr>
<tr><td></td><td>Bài 1 (CTDLGT_BT1). Cho thông tin của sinh viên gồm:
- Mã số sinh viên (maSo): Chuỗi ký tự
- Họ tên sinh viên (hoTen): Chuỗi ký tự
- Chuyên ngành (chuyenNganh): Chuỗi ký tự
- Năm sinh (namSinh): Số nguyên
- Điểm trung bình tích lũy (diemTB): Số thực
- Xếp loại (loai): Chuỗi ký tự</td><td>Tạo một Project ngôn ngữ C# mới với tên là CTDLGT_BT1, tên Solution đặt là THCTDLGT_HoVaTenSV (trong đó: HoVaTenSV là họ và tên của sinh viên, không khoảng trắng và không dấu tiếng Việt)
Bước 1. Tạo class mới đặt tên file là SinhVien
Bướ 2. Khai báo các thuộc tính
<pre>namespace CTDLGT_BT1
{
    class SinhVien
    {
        private string maSo;
        private string hoTen;
        private string chuyenNganh;</pre></td></tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td>Hãy thiết kế lớp SinhVien gồm các thông tin trên</td><td>
<pre>
        private int namSinh;
        private float diemTB;
        private string loai;

        //Các phương thức ...
    }
}
</pre>
</td></tr>
<tr><td></td><td>Lớp sinh viên (SinhVien) gồm có các chức năng:
1. Nhập, xuất thông tin sinh viên
2. Xếp loại sinh viên dựa vào điểm trung bình tích lũy theo tiêu chí (dưới 5: Kém; từ 5 đến dưới 7: Trung bình; từ 7 đến dưới 8: Khá; còn lại là Giỏi)

Lưu ý các ràng buộc:
- Tuổi của sinh viên từ 17 đến 70
- Điểm trung bình từ 0 đến 10</td><td>Bước 3. Bổ sung vào class SinhVien

1. Các properties get/set
2. Các constructors
- Default: <code>public SinhVien() { }</code>
- Parameter:
<code>public SinhVien(string ms, string ht, string cn, int ns, float dtb) { }</code>

(Thông tin xếp loại được thực hiện bên trong constructor này bằng cách gọi phương thức XepLoai(), không cần truyền vào tham số loại của sinh viên)
- Copy: <code>public SinhVien(SinhVien sv) { }</code>
3. Các phương thức theo yêu cầu
- Kiểm tra ràng buộc tuổi: <code>public bool KiemTraNamSinh(int ns) { }</code>
- Kiểm tra ràng buộc điểm: <code>public bool KiemTraDiemTB(float dtb) { }</code>
- Nhập: <code>public void Nhap() { }</code>
- Xuất: <code>public void Xuat() { }</code>
- Xếp loại: <code>public void XepLoai() { }</code></td></tr>
<tr><td></td><td>Lớp mảng sinh viên (MangSinhVien) gồm có các chức năng: Nhập và xuất danh sách sinh viên</td><td></td></tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td>Lưu ý các ràng buộc: Số lượng sinh viên phải dương và không quá 1.000.000 sinh viên</td><td></td></tr>
<tr><td></td><td>Chương trình thực hiện theo yêu cầu sau:
1. Tạo đối tượng rỗng svA
2. Nhập thông tin cho svA
3. Xuất thông tin svA
4. Tạo đối tượng svB gồm các thông tin: mã số sinh viên: “18DH001”; họ và tên: “Lam Thanh Ngoc”; chuyên ngành: “CNPM”; năm sinh: 2000; và điểm trung bình tích lũy: 7.6
5. Xuất thông tin svB
6. Tạo đối tượng svC được sao chép thông tin từ svB
7. Nhập họ tên và điểm trung bình tích lũy để cập nhật thông tin cho svC
8. Xuất thông tin svC
9. Xuất thông tin svB
10. Kiểm tra xem thông tin của svB có thay đổi không (nếu svB bị thay đổi như thông tin svC</td><td>Bước 4. Gọi thực hiện yêu cầu của chương trình trong phương thức Main() của class Program
<pre>
namespace CTDLGT_BT1
{
    class Program
    {
        static void TestSinhVien()
        {
            SinhVien svA = new SinhVien();
            svA.Nhap();
            Console.WriteLine("Thong tin svA:");
            svA.Xuat();

            SinhVien svB = new SinhVien("18DH001",
                           "Lam Thanh Ngoc",
                           "CNPM", 2000, 7.6F);
            //Tiep tuc cho nhung yeu cau khac

        }
        static void TestMangSinhVien()
        {
            MangSinhVien dssv = new MangSinhVien();
            dssv.Nhap();
            Console.WriteLine("Danh sach sinh vien:");
            dssv.Xuat();
        }

        static void Main(string[] args)
        {
            TestSinhVien();
            //Hoặc TestMangSinhVien();
        }
    }
</pre></td></tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td>thì Copy Contructor viết sai và phải chỉnh sửa lại)
11. Nhập và xuất danh sách sinh viên dssv</td><td>}</td></tr>
<tr><td>2</td><td colspan="2">Mục tiêu: Cài đặt các thuật toán tìm kiếm trên Cấu trúc dữ liệu mảng một chiều</td></tr>
<tr><td></td><td>Bài 2 (CTDLGT_BT2)
2.1. Cho lớp mảng một chiều số nguyên (IntArray). Lớp gồm các chức năng sau:
1. Properties get/set
2. Constructors
- Default
- Parameter 1: truyền vào kích thước k để phát sinh ngẫu nhiên k giá trị cho mảng có giá trị từ 1 đến 200
- Parameter 2: truyền vào một mảng một chiều a cần sao chép
- Copy: truyền vào đối tượng mảng obj cần sao chép
3. Nhập và xuất mảng</td><td>1. Sử dụng Solution THCTDLGT_HoVaTenSV đã có, sinh viên tạo thêm một Project mới tên là CTDLGT_BT2
2. Trong Project CTDLGT_BT2, thực hiện bổ sung thêm lớp mới (Lưu ý: Sinh viên nhớ Set as Startup Project cho Project này để khi thực thi thì chương trình sẽ chạy Project này).

Bước 1. Tạo class IntArray
Bưới 2. Khai báo các thuộc tính
<pre>namespace CTDLGT_BT2
{
    public class IntArray
    {
        private int []arr;
        //Các phương thức ...
    }
}</pre>Bước 3. Bổ sung vào class IntArray
1. Các properties get/set (bao gồm cài đặt index cho lớp mảng)
2. Các constructors
- Default: <code>public IntArray() { }</code>
- Parameter 1: <code>public IntArray(int k) { }</code>
- Parameter 2: <code>public IntArray(int []a) { }</code></td></tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td>4. Tìm và trả về vị trí của phần tử có giá trị x bằng 2 phương pháp: tuần tự và nhị phân
5. (*) Minh họa quá trình tìm kiếm phần tử có giá trị x bằng 2 phương pháp: tuần tự và nhị phân. Việc minh họa bao gồm: thể hiện quá trình từng bước so sánh giá trị mảng với x, cập nhật đoạn cần tìm, tính số phép so sánh với x

Lưu ý ràng buộc: Kích thước mảng phải dương và không quá 1.000.000 phần tử</td><td>- Copy: <code>public IntArray(IntArray obj) { }</code>
3. Các phương thức theo yêu cầu
- Kiểm tra kích thước mảng: <code>public bool KiemTraKT(int n) { }</code>
- Nhập: <code>public void Nhap() { }</code>
- Xuất: <code>public void Xuat() { }</code>
- Tìm kiếm bằng phương pháp tuần tự:
<pre>public int TimTuanTu(int x)
{
    int n = arr.Length;
    for (int i = 0; i < n; i++)
    {
        if (arr[i] == x)
            return i;
    }
    return -1;
}</pre>- Tìm kiếm bằng phương pháp nhị phân:
<code>public int TimNhiPhan (int x) { }</code></td></tr>
<tr><td></td><td>Chương trình thực hiện theo yêu cầu sau:
1. Tạo đối tượng mảng objA gồm k giá trị ngẫu nhiên (k nhập từ bàn phím)
2. Xem các giá trị được phát sinh trong objA
3. Nhập giá trị cần tìm x, thực hiện tìm x bằng phương pháp tuần tự trong objA và cho biết kết quả</td><td>Bước 4. Gọi thực hiện yêu cầu của chương trình trong Main()
<pre>class Program
{
    static void TestTimTuanTu()
    {
        int k, x, kq;
        Console.Write(">>Nhap so luong mang: ");
        int.TryParse(Console.ReadLine(), out k);
        IntArray obj = new IntArray(k);
        Console.WriteLine(">>Cac phan tu:");
        obj.Xuat();

        Console.Write("\n>>Gia tri can tim x = ");
        int.TryParse(Console.ReadLine(), out x);

        kq = obj.TimTuanTu(x);</pre></td></tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td>4. Tạo đối tượng mảng objB và nhập k phần tử sao cho các giá trị tăng dần
5. Nhập giá trị cần tìm x, thực hiện tìm x bằng phương pháp nhị phân trong objB và cho biết kết quả
6. (*) Dùng phương thức minh họa quá trình tìm kiếm để so sánh hiệu quả tìm kiếm của tuần tự và nhị phân trên dãy có thứ tự tăng dần cho các trường hợp</td><td><pre>
        if (kq == -1)
            Console.WriteLine("->Khong ton tai {0}!", x);
        else
            Console.WriteLine("->Co {0} tai vi tri {1}", x, kq);
    }
    //Test phương pháp tìm nhị phân
    static void TestTimNhiPhan ()
    {
        //Lưu ý: Mảng phải có thứ tự
    }
    static void Main(string[] args)
    {
        TestTimTuanTu();
        //Tương tự cho phương pháp nhị phân
    }
}
</pre></td></tr>
<tr><td></td><td>Bổ sung Bài 1 (CTDLGT_BT1). Bổ sung vào lớp MangSinhVien thêm yêu cầu sao cho nhập vào danh sách sinh viên không bị trùng mã số (nếu trùng báo lỗi và yêu cầu nhập lại mã số mới)</td><td>1. Kiểm tra xem mã số sinh viên vừa nhập tại vị trí vt có tồn tại trong danh sách sinh viên đã nhập
<pre>
public bool TonTai(string msx, int vt)
{
        for(int i=0; i<vt; i++)
        {
            if (a[i].MaSo.CompareTo(msx) == 0)
                return true;
        }
        return false;
}
</pre>2. Bổ sung phương thức nhập danh sách sinh viên: Mỗi lần nhập mã số sinh viên mới, gọi phương thức TonTai() để xác định có trùng mã số không? Nếu trùng yêu cầu nhập lại mã số mới (dùng lặp do…while).</td></tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td>3</td><td colspan="2">Mục tiêu: Cài đặt các thuật toán sắp xếp trên Cấu trúc dữ liệu mảng một chiều</td></tr>
<tr><td></td><td>Bài 2.2 (Tiếp theo - Sử dụng CTDLGT_BT2). Bổ sung vào lớp IntArray thêm các chức năng sau:
1. Sắp xếp giá trị mảng tăng dần bằng các phương pháp:
- Interchange Sort
- Bubble Sort
- Selection Sort
- Insertion Sort
- Quick Sort
- (*) Shell Sort
- (*) Shaker Sort
- (*) Merge Sort
(Tối thiểu thực hiện tại lớp 5 phương pháp đầu, những phương pháp còn lại làm bổ sung ở nhà)
2. (*) Minh họa quá trình sắp xếp của các phương pháp. Việc minh họa bao gồm: thể hiện quá trình</td><td>Định nghĩa các phương thức trong class IntArray
<pre>
public void HoanVi(ref int a, ref int b)
{
	int tam = a;
	a = b;
	b = tam;
}
public void InterchangeSort()
{
	int n = arr.Length;
	for (int i = 0; i < n; i++)
	{
		for (int j = i + 1; j < n; j++)
		{
			if(arr[i]>arr[j])
			{
				HoanVi(ref arr[i], ref arr[j]);
			}
		}
	}
}
- public void InsertionSort() { }
- public void BubbleSort() { }
- public void SelectionSort() { }
- public void QuickSort(int left, int right) { }
- public void ShellSort() { }
- public void ShakerSort() { }
- public void MergeSort() { }
</pre></td></tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td>sắp xếp từng bước và tính số phép gán hoặc hoán vị giá trị các phần tử</td><td></td></tr>
<tr><td></td><td>Chương trình thực hiện theo yêu cầu sau:
1. Tạo đối tượng mảng objA gồm k giá trị ngẫu nhiên (k nhập từ bàn phím)
2. Lần lượt kiểm tra các phương pháp sắp xếp trên objA
3. (*) Dùng phương thức minh họa quá trình sắp xếp để so sánh hiệu quả sắp xếp của các phương pháp</td><td>Kiểm tra từng phương thức sắp xếp trong Main()
<pre>
class Program
{
    static void TestInterchangeSort(IntArray obj)
    {
        //Copy ojb sang objTam để sắp xếp
        IntArray objTam = new IntArray(obj);
        Console.WriteLine("\n>>Mang ban dau:");
        objTam.Xuat();
        objTam.InterchangeSort();
        Console.WriteLine("\n>>Interchange Sort:");
        objTam.Xuat();
    }
    static void TestInsertionSort(IntArray obj)
    {
        //Copy ojb sang objTam để sắp xếp
        IntArray objTam = new IntArray(obj);
        Console.WriteLine("\n>>Mang ban dau:");
        objTam.Xuat();
        objTam.InsertionSort();
        Console.WriteLine("\n>>Insertion Sort:");
        objTam.Xuat();
    }
    //Test những phương pháp sắp xếp khác
    static void Main(string[] args)
    {
        IntArray obj = new IntArray(10);
        TestInterchangeSort(obj);
        TestInsertionSort(obj);
        //Tương tự cho những phương pháp khác
    }
}
</pre></td></tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td>Bổ sung Bài 1 (CTDLGT_BT1). Bổ sung thêm yêu cầu để sắp xếp danh sách sinh viên theo thứ tự tăng dần của họ tên sinh viên theo phương pháp chọn trực tiếp</td><td></td></tr>
<tr><td>4</td><td colspan="2">Mục tiêu: Cài đặt chuyển đổi biển thức (Infix → Postfix) dùng Cấu trúc Stack</td></tr>
<tr><td></td><td>Bài 3 (CTDLGT_BT3).
Bài 3.1. Thiết kế lớp ngăn xếp MyStack (dùng mảng một chiều để biểu diễn stack) ứng dụng cho việc chuyển đổi biểu thức Infix sang Postfix. Biểu thức Infix được nhập từ bàn phím bao gồm các thành phần/ token: số nguyên, dấu ngoặc tròn, các phép tính: cộng (+), trừ (-), nhân (*), hoặc chia (/ hoặc %)
VD biểu thức Infix: ((10+4)*2+(20/5-3))%3
Lớp MyStack bao gồm:
1. Các properties get/ set cho các thuộc tính của lớp
2. Property get Count: Cho biết số lượng phần tử trong Stack
3. Các constructors
4. Các phương thức:</td><td>Tạo class MyStack gồm có các thông tin: số lượng phần tử tối đa (stkMax), vị trí đỉnh stack (stkTop) và mảng một chiều lưu các phần tử của stack (stkArray)
<pre>namespace CTDLGT_BT3
{
    public class MyStack
    {
        private int stkMax;
        private int stkTop;
        private string[] stkArray;
        //Các properties get/ set
        //Constructors
        public MyStack(int maxItem=0)
        {
            StkMax = maxItem;
            stkArray = new string[StkMax];
            stkTop = -1;
        }
        public MyStack(MyStack s)
        {
            //Định nghĩa copy constructor
        }
        //Các phương thức
    }
}</pre></td></tr>
</table>

Tuần	Nội dung	Hướng dẫn
	- **IsEmpty()**: Kiểm tra Stack có rỗng không? - **IsFull()**: Kiểm tra Stack có đầy không? - **Pop()**: Lấy phần tử ra khỏi Stack - **Push()**: Đưa phần tử vào Stack - **Peek()**: Xem giá trị phần tử tại đỉnh Stack mà không xoá khỏi stack - **Clear()**: Xoá tất cả các phần tử trong stack - **Contains(string x)**: Kiểm tra xem stack có chứa x không? - **Input()**: Nhập lần lượt từng thành phần biểu thức sau vào thêm vào Stack cho đến khi nhập dấu chấm (.) thì kết thúc. *Ví dụ cần nhập Infix: ((10+4)*2+(20/5-3))%3 thì lần lượt nhập các chuỗi riêng biệt "(", "(", "10", v.v...* - **GetStack()**: Lấy lần lượt giá trị ra khỏi Stack và xuất ra màn hình. Chạy thử nghiệm nhập, xuất và tất cả các chức năng của lớp **MyStack.**	

Tuần	Nội dung	Hướng dẫn

Bài 3.2 (Tiếp theo - Sử dụng CTDLGT_BT3). Thiết kết lớp **MyExpressionTmp** phục vụ cho việc lưu trữ và chuyển đổi biểu thức **Infix** sang **Postfix** và tính giá trị biểu thức **Postfix** thông qua các thao tác của **Stack ở dạng đơn giản** (*chưa xử lý tách các token trong biểu thức Infix*)

Chương trình thực hiện theo yêu cầu sau:

1. Tạo sẵn một dãy các token từ một biểu thức **Infix** bao gồm các hằng số nguyên, các phép toán: cộng (+), trừ (-), nhân (*), hoặc chia (/ hoặc %) và các ưu tiên của biểu thức được đặt trong cặp dấu ngoặc tròn.

   Ví dụ: **((10+4)*2+(20/5-3))%3**

(	(	10	+	4	)	*	2	+	(	20	/	5	-	3	)	)	%	3

2. Chuyển đổi biểu thức **Infix** sang **Postfix**
3. Tính giá trị biểu thức **Postfix** và cho biết kết quả

Tạo class **MyExpressionTmp** gồm có các thông tin: danh sách ***token*** và biểu thức ***postfix***

```
namespace CTDLGT_BT3
{
    class MyExpressionTmp
    {
        private string[] token;
        private List<string> postfix;
        private MyStack opStack;
        //Các properties
        //Các constructors
        //Các phương thức xử lý
    }
}
```

Các phương thức xử lý bao gồm:

1. Tính độ ưu tiên của phép toán (+, -, *, /, %)
2. Phương thức chuyển Infix → Postfix: **ToInfix()**
3. Phương thức tính giá trị Postfix: **Value()**

```
namespace CTDLGT_BT3
{
    class Program
    {
        static void TestMyExpressionTmp()
        {
            //Infix = ((10+4)*2+(20/5-3))%3

            string[] token = { "(", "(", "10", "+", "4", ")", "*", "2",
                               "+", "(", "20", "/", "5", "-",
                               "3", ")", ")", "%", "3" };
            MyExpressionTmp myExp = new MyExpressionTmp(token);
```

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td></td><td>

```
            myExp.ToPostfix();
            double kq = myExp.Value(myExp.Postfix);
            //...
        }
        static void Main(string[] args)
        {
            TestMyExpressionTmp();
        }
    }
}
```

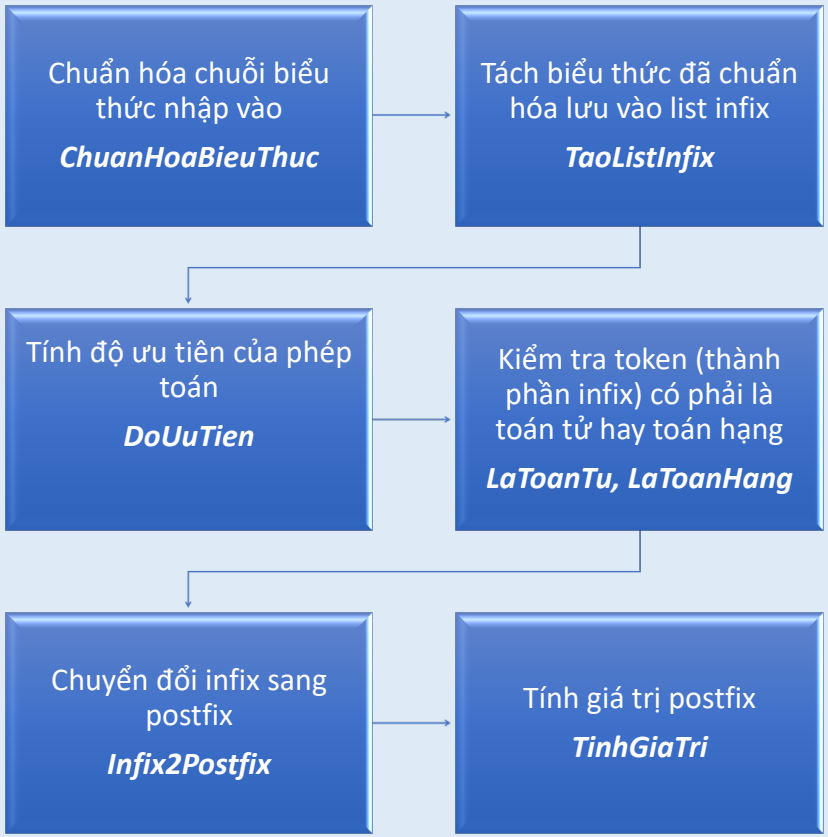
</td></tr>
<tr><td></td><td>Bài 3.3. (Tiếp theo - Sử dụng CTDLGT_BT3). Thiết kết lớp MyExpression phục vụ cho việc lưu trữ và chuyển đổi biểu thức Infix sang Postfix và tính giá trị biểu thức Postfix thông qua các thao tác của Stack

Chương trình thực hiện theo yêu cầu sau:
1. Nhập vào biểu thức Infix từ bàn phím (bao gồm các hằng số nguyên, các phép toán: cộng (+), trừ (-), nhân (*), hoặc chia (/ hoặc %) và các ưu tiên của biểu thức được đặt trong cặp dấu ngoặc tròn)
2. Chuyển đổi biểu thức Infix sang Postfix
3. Tính giá trị biểu thức Postfix và cho biết kết quả</td><td>Tạo class MyExpression gồm có các thông tin: biểu thức được nhập bieuThuc và biểu thức postfix

```
namespace CTDLGT_BT3
{
    class MyExpression
    {
        private string bieuThuc;
        private List<string> postfix;
        private MyStack opStack;
        //Các properties
        //Các constructors
        //Các phương thức xử lý
    }
}
```

Các phương thức xử lý bao gồm</td></tr>
</table>

Tuần	Nội dung	Hướng dẫn



1. Chuẩn hóa chuỗi biểu thức (*bieuThuc*) nhập từ người dùng và trả về chuỗi sau khi chuẩn hóa

```
public string ChuanHoaBieuThuc(string bieuThuc) { }
```

- Loại bỏ các khoảng trắng trong chuỗi: phương thức ***Replace()***
- Tạo ra chuỗi sao cho các thành phần chỉ cách nhau bởi 1 khoảng trắng để xử lý cho việc sử dụng phương thức tách chuỗi (***Split***) dễ dàng: Sử dụng phương thức ***Regex.Replace*** kết hợp với ***String Format***.

Tuần	Nội dung	Hướng dẫn

```
public string ChuanHoaBieuThuc(string bieuThuc)
{
    string chuanHoa = String.Copy(bieuThuc);
    chuanHoa = chuanHoa.Replace(" ", "");
    chuanHoa = Regex.Replace(chuanHoa, @"\+|\-|\*
                |\/|\%|\(|\)", delegate (Match match)
    {
        return String.Format(" {0} ", match.Value);
    });
    chuanHoa = chuanHoa.Replace("  ", " ");
    chuanHoa = chuanHoa.Trim();
    return chuanHoa;
 }
```

2. Tạo và trả về list ***infix*** từ chuỗi biểu thức (***bieuThuc*** - *danh sách các token để đưa vào xử lý chuyển đổi sang postfix*)

```
public List<string> TaoListInfix(string bieuThuc) { }
```

   - Sử dụng phương thức **Split(‘ ‘)** để tách các thành phần (*token*) trong biểu thức đã được chuẩn hóa (*các thành phần được cách nhau bởi 1 khoảng trắng*).
   - Lần lượt thêm các thành phần này vào list infix (*infix.Add(token)*)

3. Tính độ ưu tiên của phép toán

```
public int DoUuTien(string op) { }
```

   Chỉ xét phép toán +, -, *, /, hoặc % (độ ưu tiên cao hơn sẽ trả về giá trị lớn hơn)

4. Kiểm tra token có phải là toán tử

   Sử dụng phương thức ***Regex.Match()***

```
public bool LaToanTu(string token)
{
    return Regex.Match(token, @"\+|\-|\*|\/|\%").Success;
}
```

5. Kiểm tra token có phải là toán hạng

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td></td><td>Sử dụng phương thức Regex.Match(): https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference?view=netframework-4.8
<pre>public bool LaToanHang(string token)
{

    return Regex.Match(token, @"^\d+$|^([a-z]
                                        |[A-Z])$").Success;
}</pre>
6. Chuyển đổi chuỗi biểu thức bieuThuc sang postfix
<pre>public void Infix2Postfix(string bieuThuc) { }</pre>
Sử dụng cấu trúc stack (MyStack) và áp dụng thuật toán chuyển đổi
7. Tính và trả về giá trị biểu thức postfix
<pre>public double TinhPostfix() { }</pre>
Sử dụng cấu trúc stack (MyStack) và áp dụng thuật toán tính giá trị của biểu thức</td></tr>
<tr><td>5</td><td colspan="2">Mục tiêu: Cài đặt Cấu trúc danh sách liên kết đơn</td></tr>
<tr><td></td><td>Bài 4 (CTDLGT_BT4). Thiết kế lớp danh sách liên kết đơn số nguyên: MyList. Lớp MyList gồm các thành phần sau:
1. Các properties get/ set cho các thuộc tính của lớp
2. Property get Count: Cho biết số lượng phần tử trong danh sách
3. Các constructors</td><td>1. Tạo mới class tên IntNode: Cấu trúc của node số nguyên
<pre>namespace CTDLGT_BT4
{
    class IntNode
    {
        private int data;
        private IntNode next;

        public int Data
        {
            get { return data; }
            set { data = value; }
        }
        public IntNode Next
        {
            get { return next; }
            set { next = value; }</pre></td></tr>
</table>

Tuần	Nội dung	Hướng dẫn

4. Các phương thức chính sau (*sinh viên cần xác định các thành phần của phương thức cho phù hợp*):

- **AddFirst()**: Thêm một node có giá trị x vào đầu danh sách
- **AddLast()**: Thêm một node có giá trị x vào cuối danh sách
- **Input()**: Nhập các giá trị vào danh sách (*giá trị thêm vào có thể đầu hoặc cuối danh sách*)
- **ShowList()**: Hiển thị giá trị của danh sách ra màn hình
- **SearchX()**: Tìm và trả về node có giá trị x (nếu có)
- **GetMax()**: Trả về node có giá trị lớn nhất
- **GetMin()**: Trả về node có giá trị nhỏ nhất
- **GetEvenList()**: Trả về danh sách chỉ chứa số chẵn
- **GetOddList()**: Trả về danh sách chỉ chứa số lẻ

```
        }
        public IntNode(int x=0)
        {
            data = x;
            next = null;
        }
    }
}
```

2. Tạo mới ***class*** tên **MyList**: Cấu trúc của danh sách liên kết số nguyên

```
namespace CTDLGT_BT4
{
    class MyList
    {
        private IntNode first;
        private IntNode last;

        public IntNode First
        {
              get { return first; }
              set { first = value; }
        }
        public IntNode Last
        {
              get { return last; }
              set { last = value; }
        }
        public MyList()
        {
            first = null;
            last = null;
        }
        public bool IsEmpty()
        {
            return first == null;
        }
        public void AddFirst(IntNode newNode)
        {
            if (IsEmpty())
```

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td></td><td>

```
                first = last = newNode;
            else
            {
                newNode.Next = first;
                first = newNode;
            }
        }
        public void Input()
        {
            int x;
            do
            {
                Console.Write("Gia tri (0 ket thuc): ");
                int.TryParse(Console.ReadLine(), out x);
                if (x == 0)
                    return;
                IntNode newNode = new IntNode(x);
                AddFirst(newNode);
            } while (true);
        }
        public void ShowList()
        {
            IntNode p = first;
            while(p!=null)
            {
                Console.Write("{0} -> ", p.Data);
                p = p.Next;
            }
            Console.WriteLine("null");
        }
    }
}
```

</td></tr>
<tr><td></td><td>Chạy thử nghiệm các chức năng của lớp MyList</td><td>Trong class Program: Bổ sung các phương thức để kiểm tra việc nhập xuất của danh sách liên kết

```
namespace CTDLGT_BT4
{
    class Program
    {
```

</td></tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td></td><td>

```
        static void TestInput()
        {
            MyList list = new MyList();
            list.Input();
            Console.WriteLine("DSLK so nguyen:");
            list.ShowList();
        }
        static void Main(string[] args)
        {
            TestInput();
        }
    }
}
```

</td></tr>
<tr><td>6</td><td colspan="2">Mục tiêu: Cài đặt Cấu trúc danh sách liên kết đơn (tiếp theo)</td></tr>
<tr><td></td><td>Bài 4 (Tiếp theo - CTDLGT_BT4). Bổ sung vào lớp MyList các phương thức sau (sinh viên cần xác định các thành phần của phương thức cho phù hợp):
1. RemoveAt(int i): Xóa node tại vị trí thứ i (thứ tự node được tính bắt đầu từ 0)
2. RemoveX(): Xóa node có giá trị x
3. InsertAt(int x, int i): Chèn x vào danh sách tại vị trí thứ i (thứ tự node được tính bắt đầu từ 0)
4. InsertXAfterMin(): Chèn x vào sau node có giá trị nhỏ nhất
5. InsertXAfterY(): Chèn x vào sau node có giá trị y</td><td>1. Để thực hiện thao tác xóa, sinh viên cần viết các phương thức trong class MyList như sau:
- Xóa node đầu tiên của DSLK

```
public bool RemoveFirst()
{
    if (IsEmpty())
        return false;
    IntNode del = first;
    first = first.Next;
    del = null;
    return true;
}
```

- Xóa node cuối cùng của DSLK, trong đó phương thức ***PrevNode (IntNode p)*** dùng để tìm node trước node p (*nếu có*)

```
public IntNode PrevNode(IntNode p)
{
    if (p == first)
        return null;
    IntNode pTruoc = first;
    while(pTruoc.Next!=p)
```

</td></tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr>
<td></td>
<td>
6. InsertXBeforeMax(): Chèn x vào trước node có giá trị lớn nhất
7. InsertXBeforeY(): Chèn x vào trước node có giá trị y
8. Sort(): Sắp xếp danh sách theo thứ tự các giá trị tăng dần

Chạy thử nghiệm các chức năng của lớp MyList
</td>
<td>
<pre>
        {
            pTruoc = pTruoc.Next;
        }
        return pTruoc;
    }
    public bool RemoveLast()
    {
        if (IsEmpty())
            return false;
        IntNode pTruoc = PrevNode(last);
        pTruoc.Next = null;
        last = pTruoc;
        return true;
    }
</pre>
- Xóa node bất kỳ trong DSLK
<pre>
    public bool RemoveNode(IntNode p)
    {
        if (IsEmpty())
            return false;
        if (p == first)
            return RemoveFirst();
        if (p == last)
            return RemoveLast();
        IntNode pTruoc = PrevNode(p);
        pTruoc.Next = p.Next;
        p = null;
        return true;
    }
</pre>
2. Để thực hiện thao tác chèn, sinh viên cần viết các phương thức trong class MyList như sau:
- Chèn newNode vào sau node p trong DSLK
<pre>
    public void InsertAfterP(IntNode p, IntNode newNode)
    {
        if (p == last)
            AddLast(newNode);
        else
</pre>
</td>
</tr>
</table>

<table>
<tr><th>Tuần</th><th>Nội dung</th><th>Hướng dẫn</th></tr>
<tr><td></td><td></td><td>

```
            {
                IntNode pSau = p.Next;
                newNode.Next = pSau;
                p.Next = newNode;
            }
        }
```

- Chèn newNode vào trước node p trong DSLK

```
        public void HoanVi(IntNode a, IntNode b)
        {
            int tam = a.Data;
            a.Data = b.Data;
            b.Data = tam;
        }
        public void InsertBeforeP(IntNode p, IntNode newNode)
        {
            InsertAfterP(p, newNode);
            HoanVi(p, newNode);
        }
```

</td></tr>
<tr><td>7</td><td colspan="2">Mục tiêu: Cài đặt Cấu trúc cây nhị phân tìm kiếm</td></tr>
<tr><td></td><td>Bài 5 (CTDLGT_BT5). Thiết kế lớp cây nhị phân tìm kiếm số nguyên: MyBinaryTree. Lớp MyBinaryTree gồm các thành phần sau:
1. Các properties get/ set cho các thuộc tính của lớp
2. Property get:
- Count: Cho biết số lượng phần tử trong cây
- Height: Cho biết độ cao của cây
3. Các constructors</td><td>1. Tạo mới class IntTNode: Nút của cây nhị phân tìm kiếm số nguyên

```
namespace CTDLGT_BT5
{
    class IntTNode
    {
        private int data;
        private IntTNode left;
        private IntTNode right;
        public int Data
        {
            get { return data; }
            set { data = value; }
        }
        public IntTNode Left
        {
            get { return left; }
            set { left = value; }
```

</td></tr>
</table>

Tuần	Nội dung	Hướng dẫn
	4. Các phương thức chính sau (*sinh viên cần xác định các thành phần của phương thức cho phù hợp*): - **Insert()**: Thêm một node có giá trị x vào cây - **Input()**: Nhập các giá trị vào cây từ bàn phím - **NLR()**: Duyệt cây theo thứ tự trước - **LNR()**: Duyệt cây theo thứ tự giữa - **LRN()**: Duyệt cây theo thứ tự sau - **CountLeaf()**: Trả về số lượng node lá trong cây - **(*) ListByLevel()**: In ra các giá trị của node theo từng mức - **(*) ListLastLevel()**: In ra các giá trị của node ở mức cuối cùng của cây Chạy thử nghiệm các chức năng của lớp **MyBinaryTree**	``` } public IntTNode Right { get { return right; } set { right = value; } } public IntTNode(int x = 0) { data = x; left = null; right = null; } public bool InsertNode(int x) { if (data == x) //Trùng giá trị return false; if (data > x) //Giá trị cần thêm nhỏ hơn -> chèn về trái { if (left == null) left = new IntTNode(x); else return left.InsertNode(x); } else //Giá trị cần thêm lớn hơn -> chèn về phải { if (right == null) right = new IntTNode(x); else return right.InsertNode(x); } return true; } public void NLR() { Console.Write(Data + "; "); if (left != null) left.NLR(); if (right != null) right.NLR(); ```

```
        }
    }
}
```

2. Tạo mới ***class MyBinaryTree***: Cây nhị phân tìm kiếm số nguyên

```
namespace CTDLGT_BT5
{
    class MyBinaryTree
    {
        private IntTNode root;

        public IntTNode Root
        {
            get { return root; }
            set { root = value; }
        }
        public MyBinaryTree()
        {
            Root = null;
        }
        public bool Insert(int x)
        {
            if (Root == null)
            {
                Root = new IntTNode(x);
                return true;
            }
            return Root.InsertNode(x);
        }
        public void Input()
        {
            do
            {
                int x;
                Console.Write("Nhap vao gia tri (trung ket thuc): ");
                int.TryParse(Console.ReadLine(), out x);
                if (Insert(x)==true)
                    Console.WriteLine("Da them gia tri vao cay");
                else
```

Tuần	Nội dung	Hướng dẫn

```
                {
                    Console.WriteLine("Gia tri bi trung, ket thuc");
                    return;
                }
            } while (true);
        }
        public void PreOrder()
        {
            if (Root == null)
                return;
            Root.NLR();
            Console.WriteLine();
        }
    }
}
```

3. Test phương thức nhập và xuất cây nhị phân tìm kiếm số nguyên trong ***class Program***

```
namespace CTDLGT_BT5
{
    class Program
    {
        static void TestInput()
        {
            MyBinaryTree root = new MyBinaryTree();
            root.Input();
            Console.WriteLine("Duyet cay theo thu tu NLR:");
            root.PreOrder();
        }
        static void Main(string[] args)
        {
            TestInput();
        }
    }
}
```

Tuần	Nội dung	Hướng dẫn
8	**Mục tiêu: Cài đặt Cấu trúc cây nhị phân tìm kiếm (tiếp theo)**	
	Bài 5 (Tiếp theo - CTDLGT_BT5). Bổ sung vào lớp **MyBinaryTree** các phương thức chính sau (*sinh viên cần xác định các thành phần của phương thức cho phù hợp*): 1. **FindX()**: Tìm và trả về node có giá trị x trong cây 2. **FindMin()**: Tìm và trả về node có giá trị nhỏ nhất trong cây 3. **FindMax()**: Tìm và trả về node có giá trị lớn nhất trong cây 4. **RemoveX()**: Xóa node có giá trị x (nếu có) trong cây Chạy thử nghiệm các chức năng của lớp **MyBinaryTree**	Các phương thức tìm kiếm trả về Node
9	**Mục tiêu: Cài đặt Cấu trúc cây nhị phân tìm kiếm cân bằng (AVL)**	
	Bài 6 (CTDLGT_BT6). Thiết kế lớp cây nhị phân tìm kiếm cân bằng số nguyên: **MyAVLTree**. Lớp **MyAVLTree** gồm các thành phần sau: 1. Các properties get/ set cho các thuộc tính của lớp	

Tuần	Nội dung	Hướng dẫn
	2. Property get: - **Count**: Cho biết số lượng phần tử trong cây - **Height**: Cho biết độ cao của cây 3. Các constructors 4. Các phương thức chính sau (*sinh viên cần xác định các thành phần của phương thức cho phù hợp*): - **Insert()**: Thêm một node có giá trị x vào cây - **Input()**: Nhập các giá trị vào cây từ bàn phím - **NLR()**: Duyệt cây theo thứ tự trước - **LNR()**: Duyệt cây theo thứ tự giữa - **LRN()**: Duyệt cây theo thứ tự sau - **RemoveX ()**: Xóa node có giá trị x trong cây Chạy thử nghiệm các chức năng của lớp **MyAVLTree**	
10	**Tổng kết và công bố điểm học phần, giao bài kiểm tra cho những sinh viên có kết quả kém trong quá trình thực hành và chấm điểm**	