Provide a member function `concat` that concatenates the invoking `ourStr` and a given `ourStr` (`followStr`) and returns a copy of the resulting `ourStr`. (The invoking `ourStr` and `followStr` are to remain unchanged.)

- Extra added to `ourStr` Version 1 Build D:
  - Interface – `ourStr.h`:
    - Documentation:

```
//   ourStr concat(const ourStr& followStr) const
//   pre:  getLen() + followStr.getLen() <= MAX_LEN
//   post: An ourStr representing the concatenation of the invoking ourStr and
//         followStr is returned.
```

      inserted as comments.

    - Prototype:

```
ourStr concat(const ourStr& followStr) const;
```

      inserted as a public member function.

      - Note that the 1st `const` protects `followStr` and the 2nd `const` protects the invoking `ourStr`.

  - Implementation – `ourStr.cpp`:

```
ourStr ourStr::concat(const ourStr& followStr) const
{
   assert(getLen() + followStr.getLen() <= MAX_LEN);
   ourStr answer = *this; // local ourStr initialized to a copy of the invoking ourStr
   for (int i = 0; i < followStr.len; ++i)
      answer.setChar(answer.len + 1, followStr.data[i]);
   return answer; // return a copy of local ourStr
}
```

    inserted as the definition for the member function.

    - Note that a local instance of `ourStr` (`answer`) is created to hold the desired result.
    - Note how `answer` is initialized to a copy of the invoking `ourStr` (through use of "`= *this`").
    - Note that `setChar` is put to use (repeatedly) to append each character of `followStr` to `answer`).
    - Note that a copy of the local `ourStr` (`answer`) is returned by the function (via the `return` statement).

  - Application – `ourStrApp.cpp`:

```
ourStr ss1, ss2;
ss1.setStr("ab");
ss2.setStr("xyz");
( ss1.concat(ss2) ).showStr(cout);
cout << endl;
```

    inserted as exercising code to test the member function.

    - Note that, in the 4th statement, `( ss1.concat(ss2) )` is the copy of `ourStr` returned by `concat` that is directly put to use without capturing.
      - Doing it this way (without capturing it) is good when no further use (of the copy of `ourStr` returned by `concat`) is intended.
      - If we actually want to use the copy of `ourStr` returned by `concat` further, we'd capture it first and then use it, as shown below:

```
ourStr ss1, ss2;
ss1.setStr("ab");
ss2.setStr("xyz");
ourStr ss3 = ss1.concat(ss2);
ss3.showStr(cout);
cout << endl;
... // putting ss3 (that captures the ourStr returned by concat) to further use
```