You are required to demonstrate the working of a class MyList that manages a dynamically growing array of cstrings, Partial class definition, driver program (main) and required output is given below. Implement the class MyList such that the given main program runs successfully. Make sure that your program doesn't consume extra space and there shouldn't be any memory leakage or exceptions in your code.

```
#include <iostream>
#include <cstring>

using namespace std;

class MyList {
private:
    static const int MAX_SIZE = 100;
    char* arr[MAX_SIZE];
    int size;

public:
    MyList() : size(0) {}

    // Copy constructor
    MyList(const MyList& other) : size(other.size) {
        for (int i = 0; i < size; ++i) {
            arr[i] = new char[strlen(other.arr[i]) + 1];
            strcpy(arr[i], other.arr[i]);  // Use strcpy if strcpy_s is not
available
        }
    }

    // Assignment operator
    MyList& operator=(const MyList& other) {
        if (this != &other) {
            for (int i = 0; i < size; ++i) {
                delete[] arr[i];
            }
            size = other.size;
            for (int i = 0; i < size; ++i) {
                arr[i] = new char[strlen(other.arr[i]) + 1];
                strcpy(arr[i], other.arr[i]);  // Use strcpy if strcpy_s is not
available
            }
        }
        return *this;
    }
```

```
    // Destructor
    ~MyList() {
        for (int i = 0; i < size; ++i) {
            delete[] arr[i];
        }
    }

    // Overloaded + operator to concatenate two lists
    MyList operator+(const MyList& other) const {
        MyList result = *this;
        for (int i = 0; i < other.size; ++i) {
            if (result.size < MAX_SIZE) {
                result.arr[result.size] = new char[strlen(other.arr[i]) + 1];
                strcpy(result.arr[result.size++], other.arr[i]);  // Use strcpy
if strcpy_s is not available
            } else {
                cout << "List is full. Cannot add more strings." << endl;
                break;
            }
        }
        return result;
    }

    // Overloaded << operator to output the list contents
    friend ostream& operator<<(ostream& out, MyList list) {
        out << "[";
        for (int i = 0; i < list.size; ++i) {
            out << list.arr[i];
            if (i < list.size - 1) {
                out << ", ";
            }
        }
        out << "]";
        return out;
    }

    // Overloaded + operator to add a string to the list
    friend void operator+(const char* str, MyList &list) {
        if (list.size < MAX_SIZE) {
            list.arr[list.size] = new char[strlen(str) + 1];
            strcpy(list.arr[list.size++], str);  // Use strcpy if strcpy_s is not
available
        } else {
            cout << "List is full. Cannot add more strings." << endl;
```

```
        }
    }
};

int main() {
    MyList list1; // Initially list will be empty
    cout << "list1= " << list1 << endl;

    "Apple" + list1;
    cout << "list1= " << list1 << endl;

    "Banana" + list1;
    cout << "list1= " << list1 << endl;

    "Peach" + list1;
    cout << "list1= " << list1 << endl;

    MyList list2;
    cout << "list2= " << list2 << endl;

    "I love Pakistan" + list2;
    cout << "list2= " << list2 << endl;

    "Happy Programming" + list2;
    cout << "list2= " << list2 << endl;

    MyList list3;
    list3 = list1 + list2;
    cout << "list3= " << list3 << endl;
}
```

```
list1= []
list1= [Apple]
list1= [Apple, Banana]
list1= [Apple, Banana, Peach]
list2= []
list2= [I love Pakistan]
list2= [I love Pakistan, Happy Programming]
list3= [Apple, Banana, Peach, I love Pakistan, Happy Programming]


=== Code Execution Successful ===
```