Задание.

В каждом упражнении требуется реализовать в том или ином виде определение нового класса.

Создать базовый класс Array с полями: массив типа unsigned char и поле для хранения количества элементов у текущего объекта-массива. Максимально возможный размер массива задается статической константой. Реализовать конструктор инициализации, задающий количество элементов и начальное значение (по умолчанию 0). Реализовать метод доступа к элементу, перегрузив операцию индексирования []. При этом должна выполняться проверка индекса на допустимость.

Реализовать в классе Array виртуальную функцию поэлементного сложения массивов. Реализовать два класса, переопределив виртуальную функцию сложения. Вызывающая программа должна продемонстрировать все варианты вызова виртуальных функций.

15. Создать класс BitString (Класс для работы с битовыми строками не более чем из 100 бит. Битовая строка должна быть представлена массивом типа unsigned char, каждый элемент которого принимает значение 0 или 1.) и класс String (Класс для работы со строками, аналогичными строками Turbo Pascal (строка представляется как массив 255 байт, длина - в первом байте). Максимальный размер строки должен задаваться.).

MArray.h

```
#include <iostream>

using namespace std;

class MArray
{
   const static int MAX;
   unsigned char* arr = nullptr;
   int len;

   public:
       MArray(int n, int max_val = MAX);
       virtual ~MArray(void);

       int length(){ return len; }

       unsigned char& operator[](int index);
       virtual MArray operator+(MArray &in_arr);
       friend ostream& operator<<(ostream &out, MArray &a);
};

class IncorrectArraySize
{
   public:
       string message = "An exception(IncorrectArraySize) was thrown!
Your size exceeds the maximum!\n";
```

```
};
```

MArray.cpp

```
#include "MArray.h"

const int MArray::MAX = 512;

MArray::MArray(int n, int max_val)
{
   try
   {
       if (n > max_val || max_val > MAX){ throw IncorrectArraySize(); }
       arr = new unsigned char[n];
       len = n;
       for (int index = 0; index < n; index++) arr[index] = '0';
       return;
   }
   catch (IncorrectArraySize e){ cout << e.message; exit(EXIT_FAILURE);
}
}

MArray::~MArray(void)
{
   delete [] arr;
}

unsigned char& MArray::operator[](int index)
{
   unsigned char ch = '\n';
   if (index < 0 || index > MAX) return ch;
   return arr[index];
}

MArray MArray::operator+(MArray &in_arr)
{
   int max_len = max(length(), in_arr.length());
   int min_len = min(length(), in_arr.length());
   MArray ret(max_len);
   for (int index = 0; index < max_len; index++)
   {
```

```
        ret[index] = (index < min_len) ? arr[index] + in_arr.arr[index]
:
        ((min_len == length()) ? in_arr.arr[index] : arr[index]);
    }
    return ret;
}

ostream &operator<<(ostream &out, MArray &a)
{
    out << "[";
    for (int index = 0; index < a.length() - 1; index++)
    {
        out << a.arr[index] << ",";
    }
    out << a.arr[a.length() - 1] << "]"<< endl;
    return out;
}
```

BitString.h

```
#include "MArray.h"

using namespace std;

class Bitstring : public MArray
{
    int length = 0;
    public:
        Bitstring(int set_length);
        int get_length();
        void write_BitString();
        virtual Bitstring operator+(Bitstring &in_arr);
        virtual ~Bitstring(void);
};
```

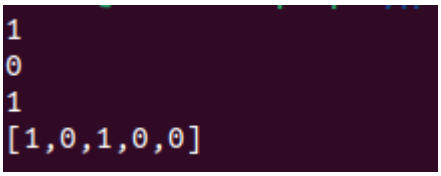
BitString.cpp

```
#include "BitString.h"

using namespace std;
```

```
Bitstring::Bitstring(int set_length) : MArray(set_length, 100){
this->length = set_length; }

int Bitstring::get_length(){return this->length;}

void Bitstring::write_BitString()
{
   Bitstring* ob;
   int n;
   // сначала берём значение текущего объекта а после ссылку на него.
   ob = &(*this);
   for (int index = 0; index < this->get_length(); index++)
   {
       cin >> n;
       (n >= 1) ? (*ob)[index] = '1' : (*ob)[index] = '0';
   }
}

Bitstring Bitstring::operator+(Bitstring &in_arr)
{
   int max_len = max(this->get_length(), in_arr.get_length());
   int min_len = min(this->get_length(), in_arr.get_length());
   Bitstring ret(max_len);
   Bitstring* ob;
   ob = &(*this);
   for (int index = 0; index < max_len; ++index)
   {
       if (index < min_len){ ret[index] = ((*ob)[index] == '1' ||
in_arr[index] == '1') ? '1' : '0'; }
       else
       { ret[index] = (min_len == get_length()) ? in_arr[index] :
(*ob)[index]; }
   }
   return ret;
}

Bitstring::~Bitstring(void){}

int main()
{
   Bitstring test1(3), test2(5);
   test1.write_BitString();
```

```
    Bitstring test3 = test1 + test2;
    cout << test3 << endl;
    return 0;
}
```

Результат.



MString.h

```
#include "MArray.h"

using namespace std;

class MString : public MArray
{
    int length;

    public:
        MString(int max_length);
        int get_length();
        void write_MString();
        virtual MString operator+(MString &in_arr);
        ~MString(void);
};
```

MString.cpp

```
#include "MString.h"

using namespace std;

MString::MString(int max_length) : MArray(max_length, 255){
this->length = max_length; }

int MString::get_length(){ return this->length; }

void MString::write_MString()
```

```
{
    MString* ob;
    unsigned char n;
    ob = &(*this);
    for (int index = 0; index < this->get_length(); index++)
    {
        cin >> n;
        (*ob)[index] = n;
    }
}

MString MString::operator+(MString &in_arr)
{
    MString ret(this->get_length() + in_arr.get_length());
    MString* ob;
    ob = &(*this);
    for (int index = 0; index < ret.get_length(); index++)
    { ret[index] = (index + 1 <= this->get_length()) ? (*ob)[index] :
in_arr[index - this->get_length()]; }
    return ret;
}

MString::~MString(void){}

int main()
{
    MString test1(2), test2(3);
    test1.write_MString();
    test2.write_MString();
    MString test3 = test1 + test2;
    cout << test3;
    return 0;
}
```

Результат.

```
O
k
Y
e
p
[O,k,Y,e,p]
```