Module 4:
Rule of Three

Dr. Nguyen Van Vu

Topics

- Templates
- Vector of objects
- Rule of three

Templates

- How do you sort an array of integers, floats, chars or even classes?

- Write each sort() function for each data type?
  - sort(int a[])
  - sort(float a[])
  - sort(MyString s[])

  - Never do this in programming!

Templates

- In C++, we avoid doing that by using template
- Template is a tool used to pass data type as a parameter
- Two types: function template and class template
- Function template

```
template <class T>
T const& max (T const& a, T const& b)
{
        return a > b ? a : b;
}
```

```
void main()
{
        int i = 5, j = 6;
        cout << max(i, j) << endl;
        float f = 0.5, d = 1.1;
        cout << max(f, d) << endl;
}
```

Templates

- Class template – similar to function template but used in class declaration

```
template <class T>
class Stack
{
private:
        vector<T> elements;
public:
        void push(T const& elmt);
}
```

```
template <class T>
void Stack<T>::push (T const& elmt)
{
        //push an element to stack
        elements.push_back(elmt);
}
void main()
{
        Stack<int> inStack;
        Stack<MyString> strStack;

        inStack.push(10);
        strStack.push("hello");
}
```

Vector of objects

- Vector is a very convenient way to represent a set of objects of a variable size
- Vector is defined in STL

```
#include <vector>
using namespace std;

template <class T>
class Stack
{
private:
        vector<T> elements;
public:
        void push(T const& elmt);
}
```

```
void main()
{
//empty vector
        vector<int> vect1;
//vector of size 10
        vector<int> vect2(10);
//vector of size 10 with initial values of 5
        vector<int> vect3(10, 5);
//vector of MyString
        vector<MyString> strVect;
}
```

Rule of tree

- Also known as the "Law of The Big Three" or "The Big Three"
- Is a rule of thumb (general rule) for C++
- Claims that a class should define explicitly
  - Destructor
  - Copy constructor
  - Copy assignment operator

The Big Three

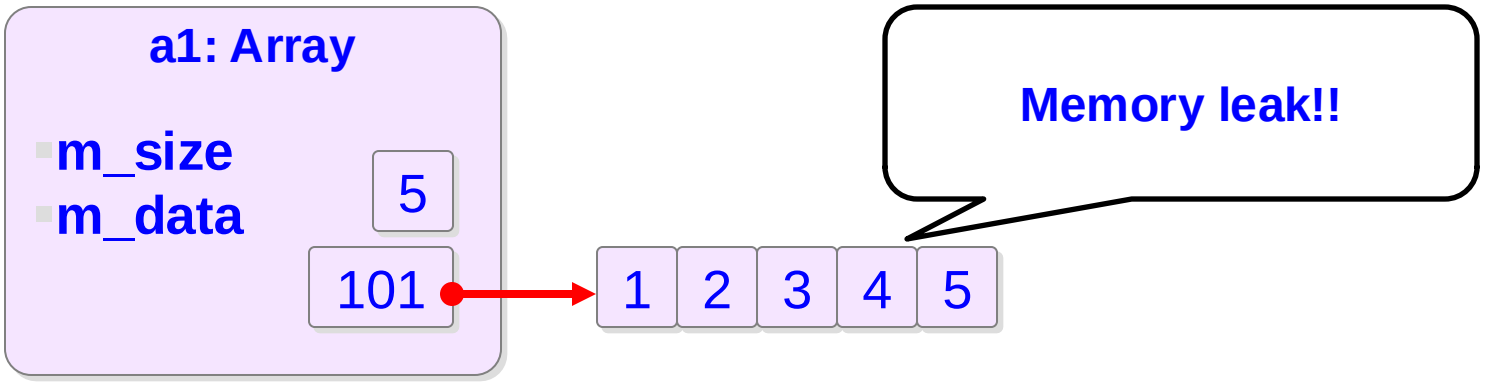
- What is the problem in the following code?

```
class Array
{
private:
        int         m_size;
        int         *m_data;
public:
        Array(int size);
};
Array::Array(int size)
{
        m_size = size;
        m_data = new int[m_size];
}
```

```
void main()
{
    Array   a1(5);
    …
}
```

The Big Three

- Problem with the default destructor
  - Class has pointer attribute and memory allocation
  - Default destructor does not de-allocate memory!!



Implement destructor EXPLICITLY to de-allocate memory!!

The Big Three

- Explicit destructor

```
class Array
{
private:
        int         m_size;
        int         *m_data;
public:
        Array(int size);
        ~Array();
};
Array::~Array()
{
        delete []m_data;
}
```

```
void main()
{
    Array   a1(5);
    …
}
```

The Big Three

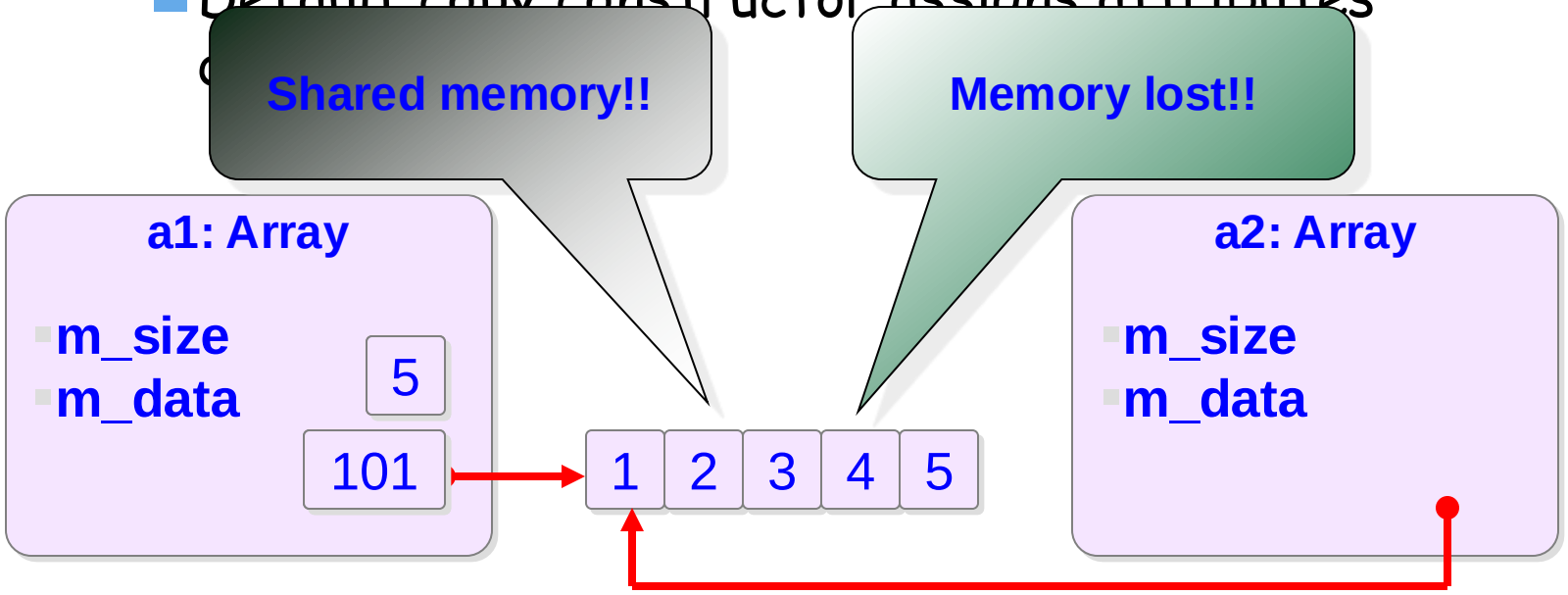
- Any problem with the following code?

```
class Array
{
private:
        int         m_size;
        int         *m_data;
public:
        Array(int size);
        ~Array();
};
```

```
void main()
{
    Array   a1(5);
    Array   a2(a1);
    …
}
```

The Big Three

- Problem with default copy constructor
  - Default copy constructor assigns attributes



Implement copy constructor EXPLICITLY to allocate memory!!

The Big Three

- Solution: make the copy constructor explicit

```
class Array
{
private:
        int         m_size;
        int         *m_data;
public:
        Array(int size);
        Array(const Array &a);
        ~Array();
};
```

```
Array::Array(const Array &a)
{
    m_size = a.m_size;
    m_data = new int[m_size];
    for (int i = 0; i < m_size; i++)
        m_data[ i ] = a.m_data[ i ];
}

void main()
{
    Array   a1(5);
    Array   a2(a1);
    …
}
```

The Big Three

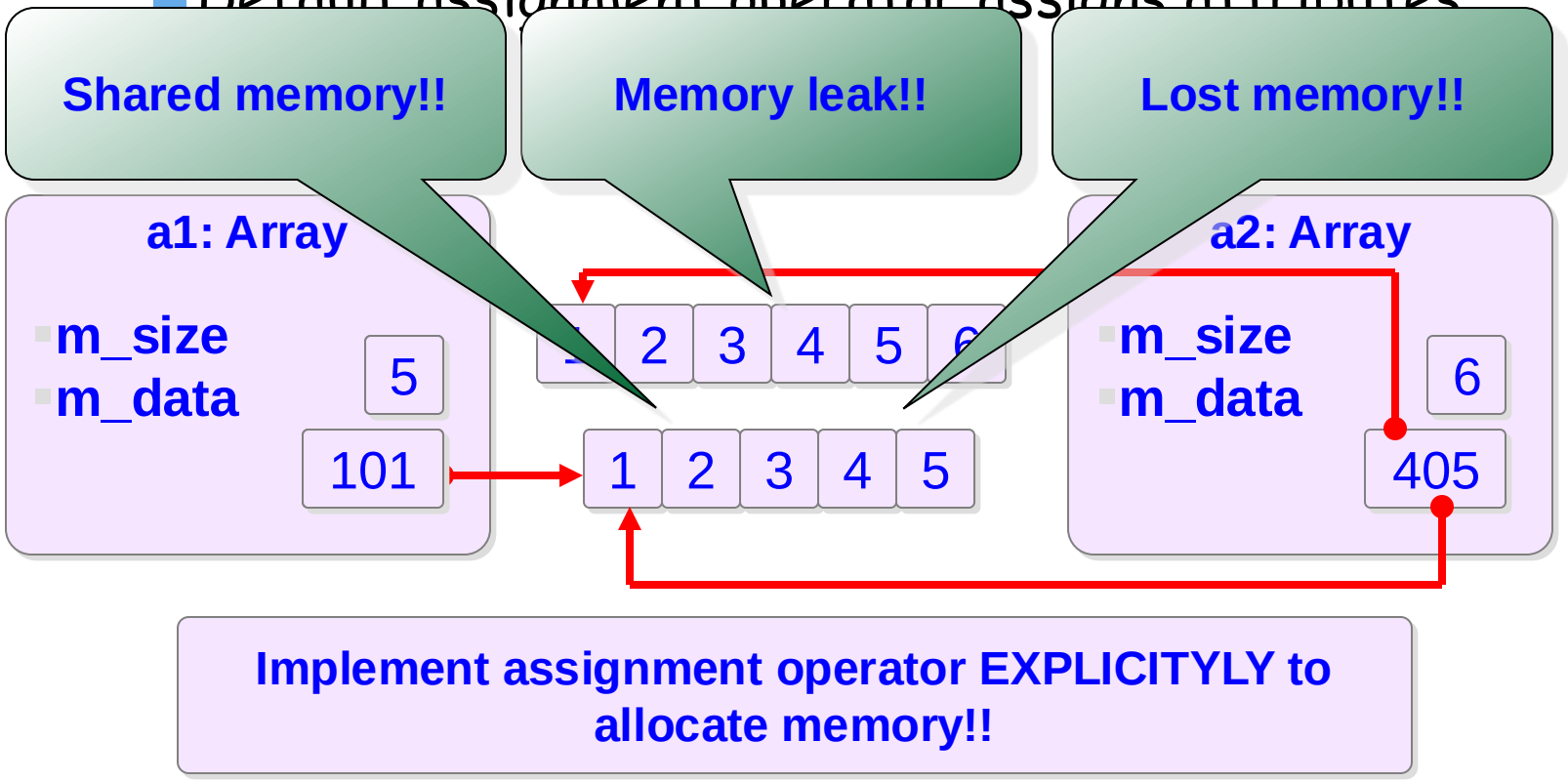
- Problem with the following code?

```
class Array
{
private:
        int         m_size;
        int         *m_data;
public:
        Array(int size);
        Array(const Array &a);
        ~Array();
};
```

```
void main()
{
    Array   a1(5);
    Array   a2(6);
    …
    a2 = a1;
    …
}
```

The Big Three

- Problem with the default assignment operator
  - Default assignment operator assigns attributes



The Big Three

- Solution: implement the assignment operator

```
class Array
{
private:
        int         m_size;
        int         *m_data;
public:
        Array(int size);
        Array(const Array &a);
        ~Array();
        Array & operator =(const Array &a);
};
```

```
Array & Array::operator =(const Array &a)
{
    delete []m_data;
    m_size = a.m_size;
    m_data = new int[m_size];
    for (int i = 0; i < m_size; i++)
        m_data[ i ] = a.m_data[ i ];
     return *this;
}
```

```
void main()
{
    Array   a1(5);
    Array   a2(6);
    …
    a2 = a1;
    …
}
```

The Big Three

- Summary
  - When a class has a pointer member and allocate memory dynamically, implement desctructor, copy constructor, copy assignment operator **explicitly**

Practice

- Write three different methods/functions in different ways to compare two MyString
- Use vector instead of array to implement the relationship between student, course, university classes
- Rewrite MyString class by explicity implementing copy constructor, destructor, and copy assignment operator