Encapsulation

Inst. Nguyễn Minh Huy

Contents

- The Big Three.
- Encapsulation.

Contents

- **The Big Three.**
- Encapsulation.

The Big Three

- Class three default methods:
  - Provided by compiler when not declared.
    - Default destructor.
    - Default copy constructor.
    - Default assignment operator.

```
class Fraction
{
private:
        int  m_num;
        int  m_den;
public:
        Fraction( int  num, int  denom );
};
```

```
int main()
{
        Fraction  p1( 1, 3 );

        // Default copy constructor.
        Fraction  p2( p1 );

        // Default assignment.
        p1 = p2;
}
```

The Big Three

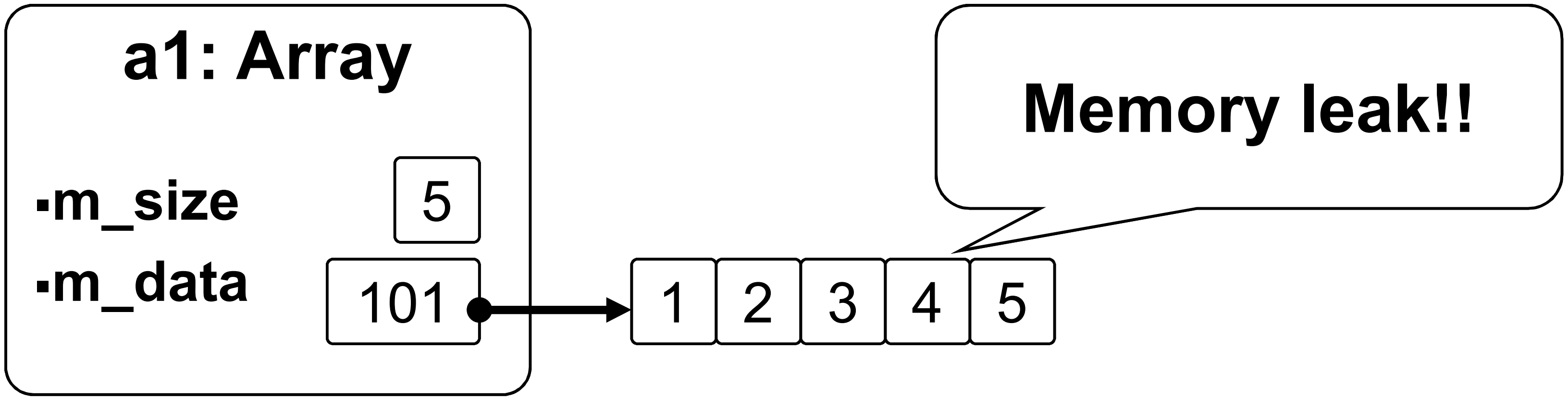
- Example 1:

```
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];
}
```

```
int main()
{
    Array   a1(5);
    …
} // Default destructor called.
```

The Big Three

- Default destructor problem:
  - Class has pointer attribute and memory allocation.
  - Default destructor does not de-allocate memory!!



Implement destructor EXPLICITLY to de-allocate memory!!

The Big Three

- Example 1:

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

```
int main()
{
    Array   a1(5);
    …
} // Explicit destructor called.
```

The Big Three

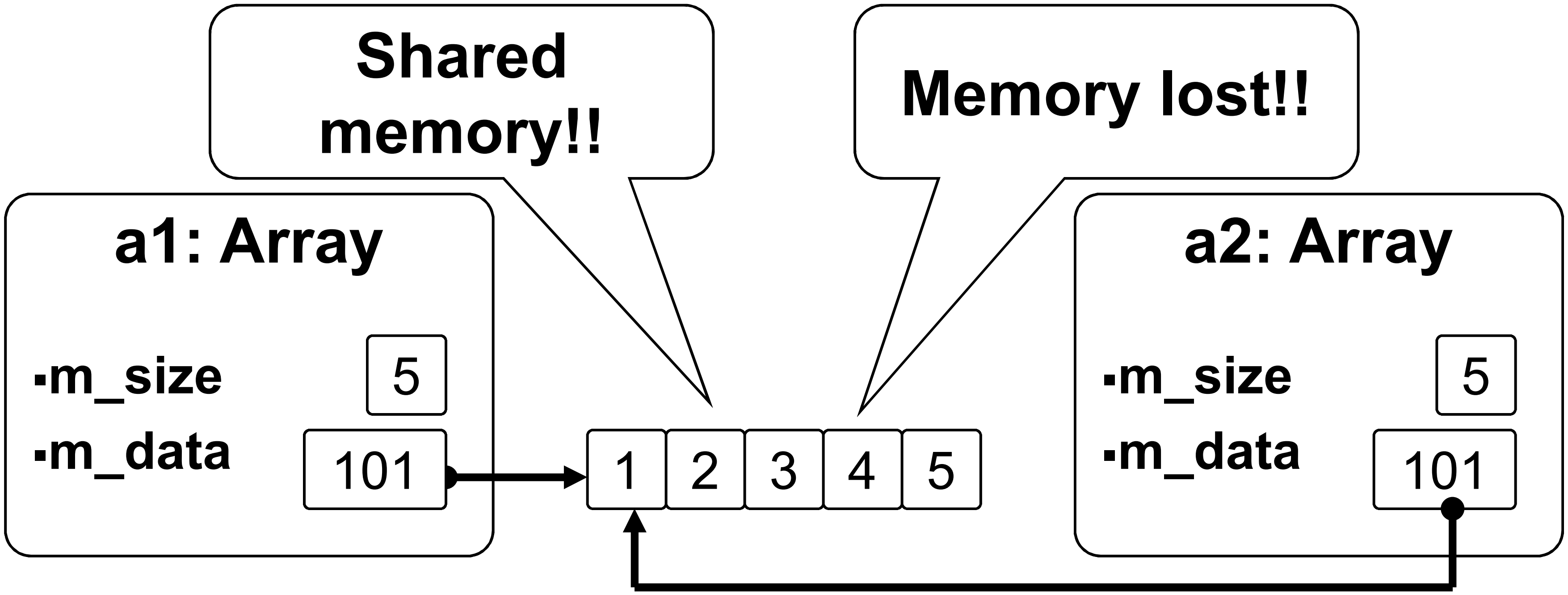
- Example 2:

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

```
int main()
{
    Array   a1(5);
    Array   a2(a1); // Default copy
    …                // constructor called.
}
```

The Big Three

- Default copy constructor problem:
  - Default copy constructor assign attributes directly!!



Implement copy constructor EXPLICITLY to allocate memory!!

The Big Three

- Example 2:

```
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 ];
    std::copy( a.m_data,
            a.m_data + m_size, m_data );
}

int main()
{
    Array   a1(5);
    Array   a2(a1); // Explicit copy
    …               // constructor called.
}
```

The Big Three

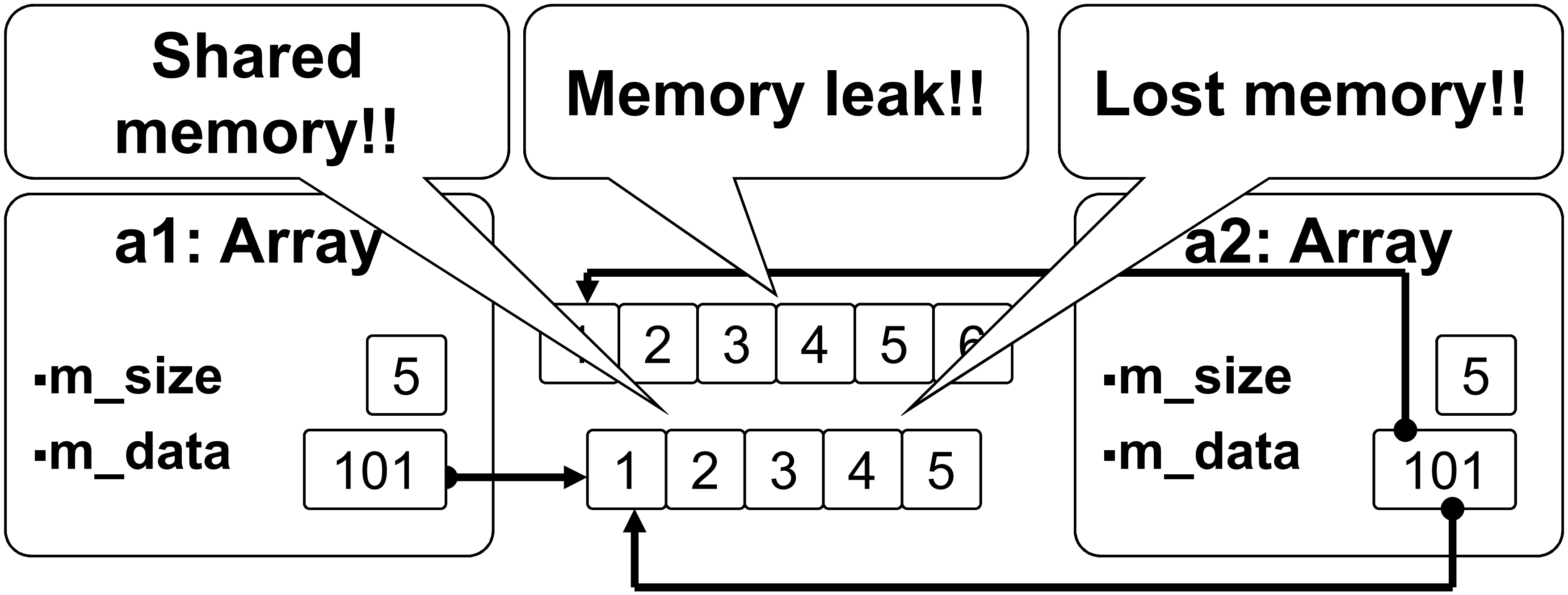
- Example 3:

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

```
int main()
{
     Array   a1(5);
     Array   a2(6);
     …
     a2 = a1; // Default assignment.
     …
}
```

The Big Three

- Default assignment operator problem:
  - Default assignment operator assigns attributes directly!!



Implement assignment operator EXPLICITYLY to allocate memory!!

The Big Three

- Example 3:

```
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)
{
    if ( this != &a ) {
        delete [ ]m_data;

        m_size = a.m_size;
        m_data = new int[ m_size ];
        std::copy( a.m_data,
                a.m_data + m_size, m_data );
    }
    return *this;
}
```

The Big Three

- Example 3:

```
class Array
{
private:
    int  m_size;
    int *m_data;
public:
    Array(int size);
    Array(const Array &a);
    ~Array();
    // Copy and swap idiom
    Array & operator =(const Array a);
    void swap( Array &a );
};
```

```
// Copy and swap idiom
// -Use pass-by-value to make copy
// -Then swap contents.
Array & Array::operator =(const Array a)
{
    swap( a );
    return *this;
}
void Array::swap( Array &a )
{
    std::swap( m_size, a.m_size );
    std::swap( m_data, a.m_data );
}
```

The Big Three

- Dr. Guru advises: **“Rule of Three”**
  - Class having pointer and memory allocation,
    ➔ Implement The Big Three EXPLICITLY:
    - Destructor: de-allocate memory.
    - Copy constructor: allocate new and copy memory.
    - Assignment: de-allocate old, then allocate new and copy.

```
class Student
{
private:
        char        *m_name;
public:
        ~Student();
        Student(const Student &s);
        Student & operator =(const Student &s);
};
```

Contents

- The Big Three.
- **Encapsulation.**

Encapsulation

- Rule of Black Box:
  - Attributes: **private** to limit access.
  - Methods: **public** to provide functions.

```
class Student
{
private:
        char*    m_name;
        float    m_math;
        float    m_literature;
};
```

Attributes

Methods

Encapsulation

- Data hiding vs. access demand:
  - Require to access attributes to do tasks?
    - Solution 1: private → public.
    - Solution 2: use getters.

      ➔ Violate Rule of Black Box!!

```
class Student
{
public:
        char*       m_name;
        float       m_math;
        float       m_literature;
};
```

```
class Student
{
private:
        char*       m_name;
        float       m_math;
        float       m_literature;
public:
        float       getMath();
        float       getLiterature();
};
```

Encapsulation

- How to follow Rule of Black Box?
  - Give tasks to object instead of asking them attributes.

```
class Student
{
private:
        char*      m_name;
        float      m_math;
        float      m_literature;
public:
        float calculateGPA();
        int rank();
};
```

```
int main()
{
        Student  s;

        // Need to calculate GPA??
        // Let student do it.
        float  dtb = s.calculateGPA();

        // Need ranking??
        // Let student do it.
        int  loai = s.rank();
}
```

Encapsulation

- Dr. Guru advises: **“Tell, Don’t Ask”**
  - Object attributes

    ➔ Hide from outside access.
  - Object keeps data

    ➔ Is responsible to do tasks relating to them.
  - “Don’t ask me information”

    ➔ “Tell me to do the jobs!!”
  - Give me data

    ➔ Please also give me tasks.

Encapsulation

- Practice:

```
// Find triangle centroid??
class Point
{
private:
        float      m_X;
        float      m_Y;
};

class Triangle
{
private:
        Point      m_A;
        Point      m_B;
        Point      m_C;
};
```

```
//  Print excellent students??
// (GPA >= 8.5)
class Student
{
private:
        char       *m_name;
        float      m_math;
        float      m_literature;
};
class StudentList
{
private:
        std::vector<Student>   m_list;
};
```

- The Big Three:
  - Three default methods compiler provides:
    - Default destructor.
    - Default copy constructor.
    - Default assignment.
  - They do not work well with pointers and allocations.
  - Rule of Three: provide explicit ones.
- Encapsulation:
  - Follow Rule of Black Box.
  - “Tell, don’t ask” principle:
    - Do not ask object data to do task.
    - Tell object to do the task instead.

Practice

- Practice 5.1:

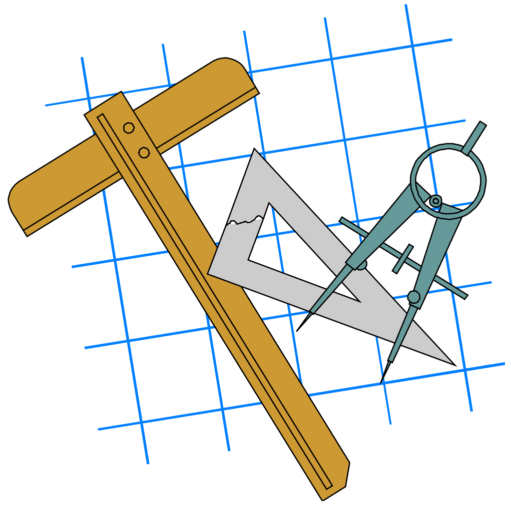
  Construct class **Polynomial** having the followings:

  (Constructors and destructor)

  - Default construction with degree = 0.
  - Construction with degree and array of coefficients.
  - Construction from another polynomial object.
  - Destruction, de-allocate memory.

  (Getters and setters)

  - Get/set degree.
  - Get/set coefficient at a degree.



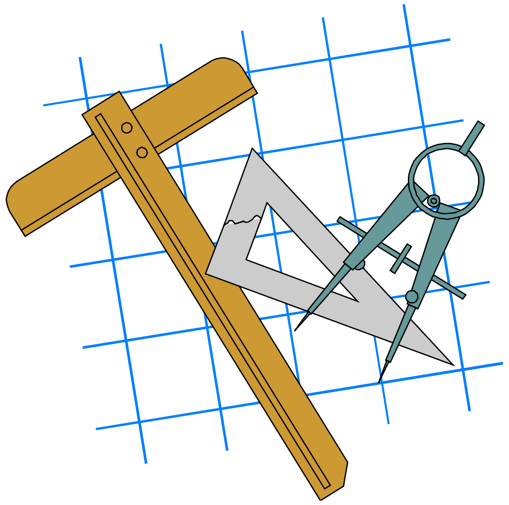
Practice

- Practice 5.1:

  Construct class **Polynomial** (continue):

  (Operators)

  - Arithmetics: +, -, *, /, =.
  - Comparisons: >, <, ==, >=, <=, !=
  - Derivative (!), anti-derivative (~).
  - Input and output: >>, <<.



Practice

- Practice 5.2:

  Construct necessary classes to do the followings on triangle:

  - Calculate triangle perimeter and area.
  - Calculate triangle centroid.
  - Find triangle perpendicular bisector of a side.