한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

Introduction to Computers & Lab

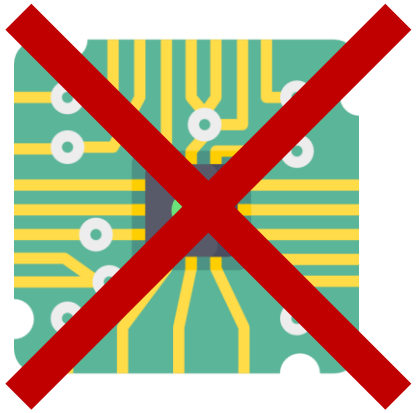
Lab 13

2021.06.03

Prof. Muhammad Bilal
TA. Sohee Jang

Index

Interface



Internal implementation is unknown

Provides an interface
to manipulate TV

For similar reasons, implementation and interface separation are very useful in programming.

Constructor


```
struct Foo {
  public :
    int m_x;
    int m_y;
};

int main() {
  Foo foo1 = {4, 5};
  Foo foo2 = {6, 7};

  return 0;
}
```

If all the members of the structure are public, the initialization list can be used to initialize directly.

However, if a member variable is private, the class can no longer be initialized in the same way because it is private and cannot be accessed directly.

Constructor

A constructor is a special kind of member function that is automatically invoked when an object of that class is instantiated. Generators are typically used when a class' member variable is initialized to an appropriate default value or user-supplied value, or the settings required to use the class (such as opening an ex. file).

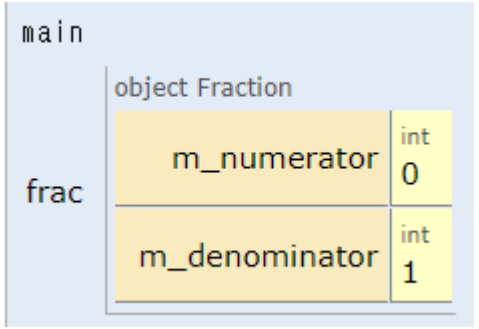
1. The constructor name must be the same as the class.

2. The constructor does not have a return type. It's not void either.

Default Constructor


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

struct Fraction {
  private :
    int m_numerator; //분자
    int m_denominator; //분모
  public :
    Fraction() {
      m_numerator = 0;
      m_denominator = 1;
    }
    int getNumerator(){return m_numerator;}
    int getDenominator(){return m_denominator;}
};

int main() {
  Fraction frac;
  cout << frac.getNumerator() << "/" << frac.getDenominator() << endl;
  return 0;
}
```

Constructors that do not have parameters or all have parameters with default values are called default constructors. When instantiating a struct, the default constructor is called if the user does not provide the initial value.



Constructor with Parameters

The default constructor is useful for setting the default values of struct member variables, but sometimes we want to initialize the values of struct instance-specific member variables to specific values. Fortunately, we can declare parameters on the constructor.

```
public :
  Fraction() {
    m_numerator = 0;
    m_denominator = 1;
  }

  Fraction(int numerator, int denominator = 1) {
    m_numerator = numerator;
    m_denominator = denominator;
  }
```

In the example, there are two constructors in a struct:

1. Default constructor to be called in default case
2. Constructors with two parameters

Overloading

Destructor

한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

An destructor is a member function of a struct that runs automatically when an object is destroyed. The constructor is designed to help initialize the struct, but the destructor is designed to help clean.

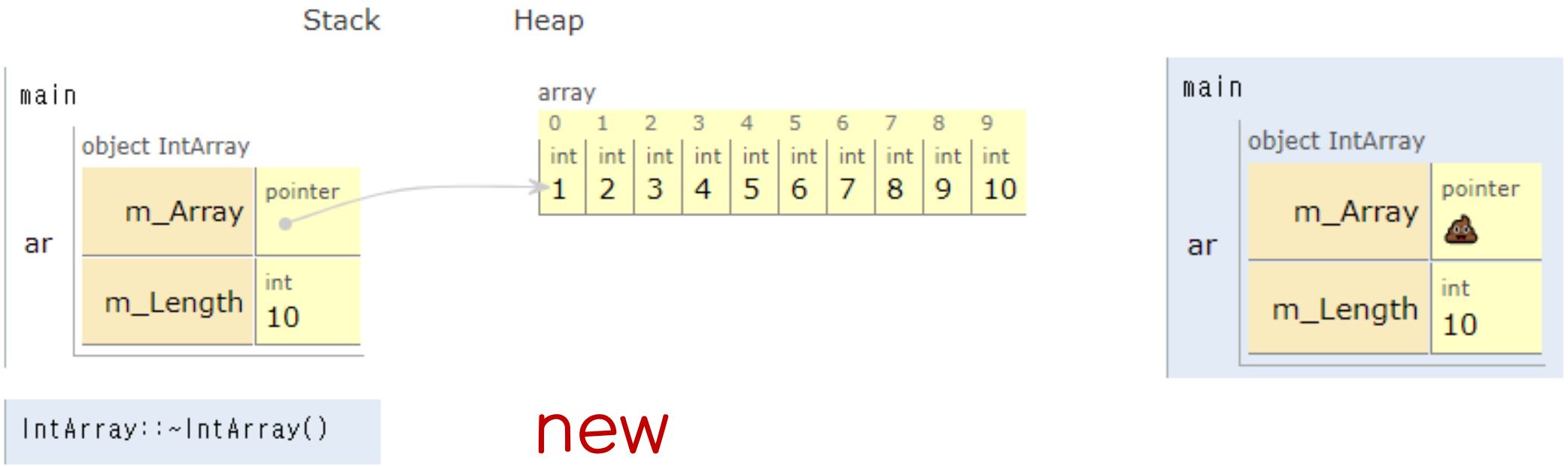
1. The destructor name = the struct name, and must be preceded by ~.

2. Destructors have no arguments.

3. Destructor has no return value.

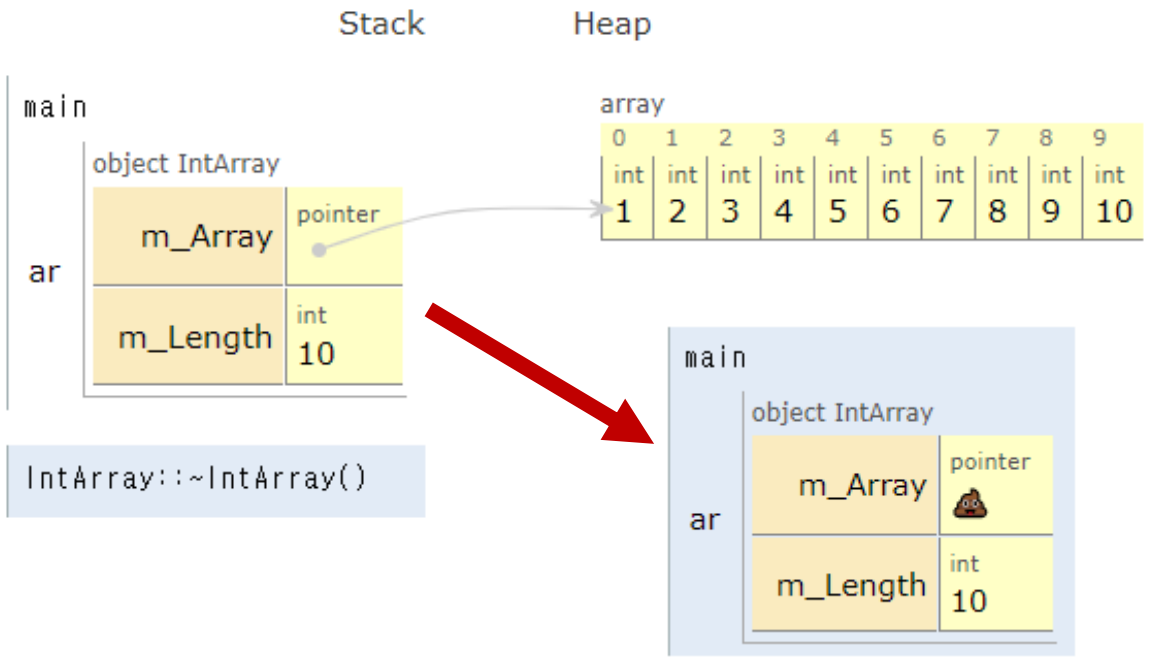
Because of these rules, only one destructor can exist per struct.
Also, there is no explicit invocation of destructors.

New-Delete


Stack
Heap
main
ar
object IntArray
m_Array
pointer
m_Length
int
10
array
0 1 2 3 4 5 6 7 8 9
int int int int int int int int int int
1 2 3 4 5 6 7 8 9 10
IntArray::~IntArray()
new
main
ar
object IntArray
m_Array
pointer
m_Length
int
10
delete

Destructor

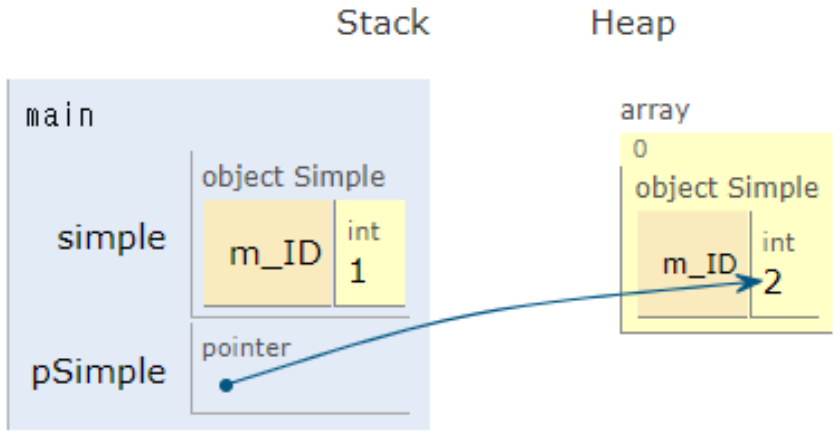
```
#include <iostream>
using namespace std;

struct IntArray {
  private :
    int* m_Array;
    int m_Length;
  public :
    IntArray(int length) { //Constructor
      m_Array = new int[length]{};
      m_Length = length;
    }

    ~IntArray() { //Destructor
      delete[] m_Array;
    }

    void SetValue(int index, int value){m_Array[index] = value;}
    int GetValue(int index){return m_Array[index];}
    int GetLength(){return m_Length;}
};

int main() {
  IntArray ar(10);
  for(int cnt = 0; cnt < ar.GetLength(); ++cnt){
    ar.SetValue(cnt, cnt+1);
  }
  cout << "The value of element 5 is: " << ar.GetValue(5) << endl;
}
```




```
The value of element 5 is: 6
```

Lifecycle of Destructor

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

struct Simple {
  private :
    int m_ID;
  public :
    Simple(int id) : m_ID(id){
      cout << "Constructing " << m_ID << endl;
    }

    ~Simple() {
      cout << "Destructing " << m_ID << endl;
    }

    int GetID(){return m_ID;}
};

int main() {
  Simple simple(1);
  cout << simple.GetID() << endl;
  Simple* pSimple = new Simple(2);
  cout << pSimple->GetID() << endl;

  delete pSimple;
}
```



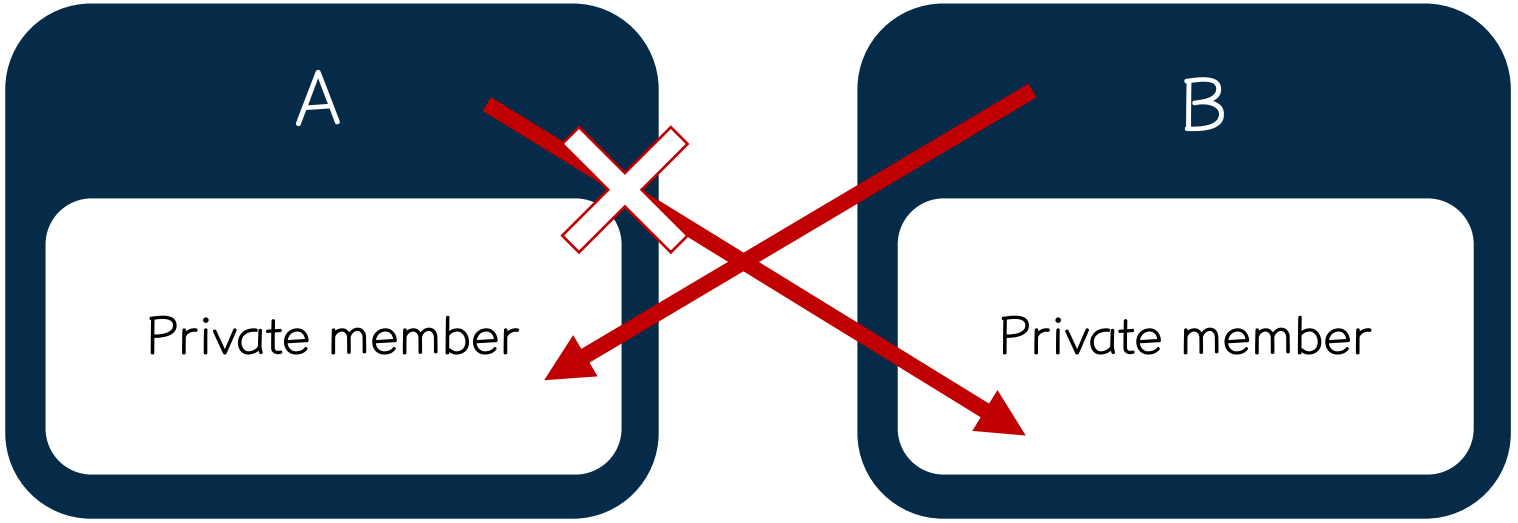

```
Constructing 1
1
Constructing 2
2
Destructing 2
Destructing 1
```

Friend

한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

Let's say class A and B exists. Both A and B have private members. As we know, private members are not accessible from outside. However, if class B is specified as friend within class A, class B will be able to access private members of class A directly. However, the opposite is not possible.

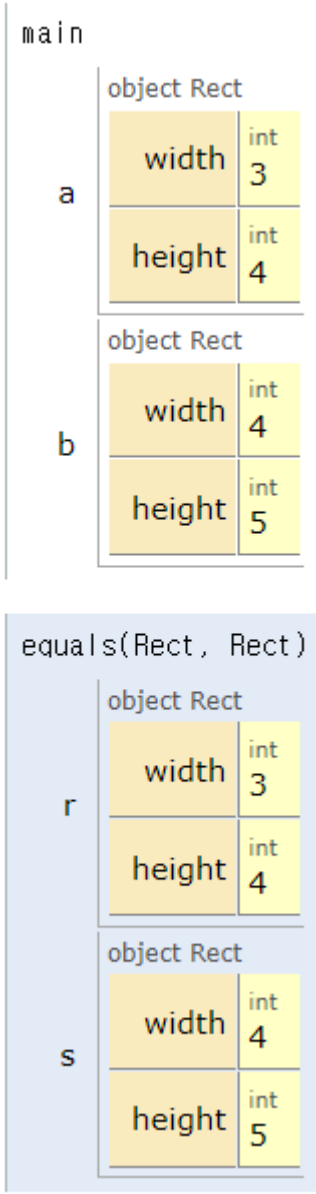
```
class A {
  friend class B;
  ...
}
```



Friend

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

class Rect {
  int width, height;
  public :
    Rect(int width, int height) {
      this->width = width; this->height = height;}
    friend bool equals(Rect r, Rect s);
};

bool equals(Rect r, Rect s) {
  if(r.width == s.width && r.height == s.height)
    return true;
  else
    return false;
}

int main() {
  Rect a(3,4), b(4,5);
  if(equals(a,b)) cout << "equal" << endl;
  else cout << "not equal" << endl;
  return 0;
}
```

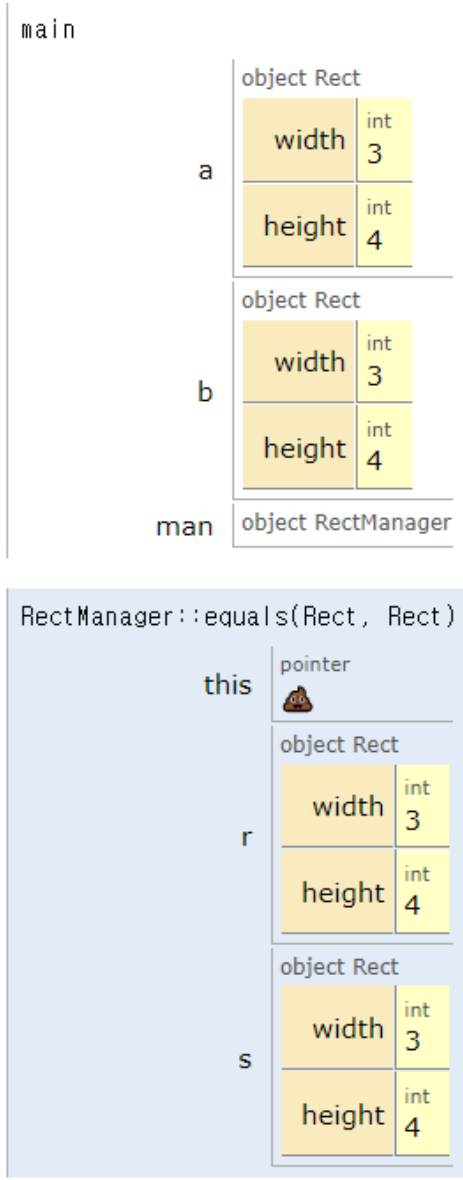
```
not equal
```



Friend

```cpp
#include <iostream>
using namespace std;

class Rect;

class RectManager {
  public:
    bool equals(Rect r, Rect s);
};

class Rect {
  int width, height;
  public:
    Rect(int width, int height) {
      this->width = width;
      this->height = height;
    }
    friend bool RectManager::equals(Rect r, Rect s);
};

bool RectManager::equals(Rect r, Rect s){
  if(r.width == s.width && r.height == s.height) return true;
  else return false;
}

int main() {
  Rect a(3,4), b(3,4);
  RectManager man;

  if(man.equals(a,b))
    cout << "equal" << endl;
  else cout << "not equal" << endl;
  return 0;
}
```

equal



한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

Task 1 : The biggest box

한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

- Create a program by duplicating the constructors of a box class.
    1. default) width = 4.0, length = 4.0, height = 3.2
    2. When receiving parameters width, length, and height

- Create a getVolume() member function to calculate volume.

- Make 3 boxes.
    1. Box1) width = 3.4, length = 5.5, height = 5.0
    2. Box2) default
    3. Box3) width, length, height -> get, set

Task 2 : Width of a circle

Create a program to find the width of a circle.

1. Implement a setRadius (int radius) member function that sets the radius.

2. Implement the getArea() member function that returns by obtaining the area.

3. The circle array is declared by inputting the number of circles, and the radius of each circle is inputted and stored as a setRadius function.

Task 3 : Class in Class

Create a program that checks the steps of the constructor and destructor through the class of employee and date.

```
Date construct with 3 parameters
Date construct with 3 parameters
Date default construct
Date default construct
Employee construct with 3 parameters
tina
1993 9 10
2018 3 1
1993 9 10
Date default construct
Date default construct
Date distructer
Date distructer
Date distructer
Date distructer
Date distructer
Date distructer
```

Task 4 : a + b

Create a program that includes an operator function for a+b using friend. After defining the power class, the operator+ member function is defined as friend.