Topic 8
Classes – Constructors and Destructor

Constructor

- **Constructor overloading**
- **Default constructor**
- **Member initializer**
- **Copy constructor**

Destructor

Object Constructions

- Uninitialized instance can cause an unpredictable behavior
- Construction consists of two steps
  - allocation of appropriate memory space for the object
  - initialization function is called automatically to initialize the memory space
- Ensure when an object is created, it is always placed in a predictable state.

Object Construction in C++

- A constructor is a specialized member function
  - Only used for initializing object
  - Automatically invoked whenever an object is created
  - has the same name as the class itself
  - has no return type

```
class Rectangle {
private:
   int leftTopX, leftTopY ;
   int rightBottomX, rightBottomY ;
public:
   Rectangle(int x1, int y1, int x2, int y2) {
      leftTopX = x1 ; leftTopY = y1 ;
      rightBottomX = x2 ; rightBottomY = y2 ;
   }
   …
```

Constructor

```
# include <iostream>
using namespace std ;
class Rectangle {
private:
   int leftTopX, leftTopY ;
   int rightBottomX, rightBottomY ;
public:
   Rectangle(int x1, int y1, int x2, int y2) {
      leftTopX = x1 ; leftTopY = y1 ;
      rightBottomX = x2 ; rightBottomY = y2 ;
      // 대신에set()을호출하는것도가능함
   }
   void set(int x1, int y1, int x2, int y2) {
      leftTopX = x1 ; leftTopY = y1 ;
      rightBottomX = x2 ; rightBottomY = y2 ;
   }
   void getLeftTop(int& x, int& y) {
      x = leftTopX ; y = leftTopY ;
   }
   void getRightBottom(int& x, int& y) {
      x = rightBottomX ; y = rightBottomY ;
   }
   int getArea() {
      return (rightBottomX - leftTopX)
      * (rightBottomY - leftTopX) ;
   }
} ;
```

```
int main() {
    int x1, y1, x2, y2 ;
    cin >> x1 >> y1 >> x2 >> y2 ;

    Rectangle r1(x1, y1, x2, y2) ;
    // r1.set(…)을 하지 않음

    int x3, y3, x4, y4 ;
    r1.getLeftTop(x3, y3) ;
    r1.getRightBottom(x4, y4) ;

    Rectangle r2(x3, y3, x4, y4) ;
    // r2.set(…)을 하지 않음

    cout << endl << r1.getArea()
        << '\t' << r2.getArea() << endl ;
}
```

Constructor Invocation

```
int main() {
	int x1, y1, x2, y2 ;
	cin >> x1 >> y1 >> x2 >> y2 ;

	Rectangle r1(x1, y1, x2, y2) ;

	int x3, y3, x4, y4 ;
	r1.getLeftTop(x3, y3) ;
	r1.getRightBottom(x4, y4) ;

	Rectangle* pR = new Rectangle(x3, y3, x4, y4) ;

	cout << endl << r1.getArea() << '\t' << pR->getArea() << endl ;

	delete pR ;
}
```

Constructor Overloading

```
class Point {
public:
    int x, y ;
    Point(int x, int y) { this->x = x ; this->y = y ; }
} ;
class Rectangle {
private:
    int leftTopX, leftTopY ;
    int rightBottomX, rightBottomY ;
public:
    // 1) 번생성자
    Rectangle(int x1, int y1, int x2, int y2) { set(x1, y1, x2, y2) ; }
    // 2) 번생성자
    Rectangle(int x, int y) { set(x, y, 0, 0) ; }
    // 3) 번생성자
    Rectangle(const Point& leftTop, const Point& rightBottom) {
        set(leftTop.x, leftTop.y, rightBottom.x, rightBottom.y) ;
    }
    // 4번생성자
    Rectangle(const Point& leftTop) { set(leftTop.x, leftTop.y, 0, 0) ; }
    void set(int x1, int y1, int x2, int y2) {
        leftTopX = x1 ; leftTopY = y1 ;
        rightBottomX = x2 ; rightBottomY = y2 ;
    }
```

Constructor Overloading

```
int main() {
    int x1, y1, x2, y2 ;
    cin >> x1 >> y1 >> x2 >> y2 ;

    // 1) 번생성자호출
    Rectangle r1(x1, y1, x2, y2) ;

    // 2) 번생성자호출
    Rectangle* pR2 = new Rectangle(x1+10, y1+10) ;

    Point p1(10, 10), p2(20, 20) ;
    // 3) 번생성자호출
    Rectangle r3(p1, p2) ;

    // 4) 번생성자호출
    Rectangle* pR4 = new Rectangle(Point(30, 30)) ;

    delete pR2 ;
    delete pR4 ;
}
```

Constructor Overloading

- Overloading can be simplified by default arguements

```
class Point {
public:
    int x, y ;
    Point(int x, int y) { this->x = x ; this->y = y ; }
} ;
class Rectangle {
private:
    int leftTopX, leftTopY ;
    int rightBottomX, rightBottomY ;
public:
    // 1) 번생성자
    Rectangle(int x1, int y1, int x2=0, int y2=0) { set(x1, y1, x2, y2) ; }
    // 2) 번생성자: 불필요
    // Rectangle(int x, int y) { set(x, y, 0, 0) ; }
    // 3) 번생성자
    Rectangle(const Point& leftTop, const Point& rightBottom=Point(0,0)) {
        set(leftTop.x, leftTop.y, rightBottom.x, rightBottom.y) ;
    }
    // 4번생성자: 불필요
    // Rectangle(const Point& leftTop) { set(leftTop.x, leftTop.y, 0, 0) ; }
    void set(int x1, int y1, int x2, int y2) {
        leftTopX = x1 ; leftTopY = y1 ;
        rightBottomX = x2 ; rightBottomY = y2 ;
    }
```

Default constructor

- A constructor without no argument

```
class Rectangle {
private:
    int leftTopX, leftTopY ;
    int rightBottomX, rightBottomY ;
public:
    // 기본(default) 생성자
    Rectangle() { set(0, 0, 0, 0) ; }
    …
    …
} ;
```

```
int main() {
    int x1, y1, x2, y2 ;
    cin >> x1 >> y1 >> x2 >> y2 ;

    // 기본생성자호출
    Rectangle r ;
    // Rectangle r()로하면Rectangle을
    // return하는함수r()에대한선언과혼동됨
    cout << r.getArea() << endl ;

    Rectangle* pR = new Rectangle() ;
    cout << pR->getArea() << endl ;

    delete pR ;
}
```

Default constructor

- A compiler will generate default constructor, if there are no constructors.

```
class Rectangle {
private:
    int leftTopX, leftTopY ;
    int rightBottomX, rightBottomY ;
public:
    // 생성자가 없음
    …
    void set(int x1, int y1, int x2, int y2) {
        …
    }
    …
    …
} ;
```

```
int main() {
    int x1, y1, x2, y2 ;
    cin >> x1 >> y1 >> x2 >> y2 ;

    // 자동으로 생성된 기본 생성자가 호출됨
    Rectangle r ;
    cout << r.getArea() << endl ;

    Rectangle* pR = new Rectangle() ;
    cout << pR->getArea() << endl ;

    delete pR ;
}
```

Initialization of Objects Array

```
class Point {
public:
    int x, y ;
    Point(int x, int y) { this->x = x ; this->y = y ; }
} ;
class Rectangle {
    …
public:
    // 기본(default) 생성자
    Rectangle() { … }
    // 1) 번생성자
    Rectangle(int x1, int y1, int x2, int y2) { ...}
    // 2) 번생성자
    Rectangle(int x, int y) { set(x, y, 0, 0) ; }
    // 3) 번생성자
    Rectangle(const Point& leftTop,
       const Point& rightBottom) { … }
    // 4번생성자
    Rectangle(const Point& leftTop) { … }
    …
}
```

```
int main() {
   // 기본생성자호출
   Rectangle retangles1[10] ;

   Rectangle rectangles2[] = {
      Rectangle(),    // 기본생성자
      Rectangle(10, 10, 20, 20),
      Rectangle(10, 10),
      Rectangle(Point(10, 10), Point(20,20)),
      Rectangle(Point(10,10))
   } ;

   int rectNo ;
   cin >> rectNo ;
   // 기본생성자호출됨
   Rectangle* pRectangles =
      new Rectangle[rectNo] ;

   delete [] pRectangles ;
}
```

Member Initializer

- 데이터 멤버 초기화 위치

```
class Rectangle {
  int leftTopX, leftTopY ;
  int rightBottomX, rightBottomY ;
public:
  Rectangle(int x1=0, int y1=0, int x2=0, int y2=0)
    : leftTopX(x1), leftTopY(y1) // 멤버 초기화 목록
  { // 함수 본문
    rightBottomX = x2 ; rightBottomY = y2 ;
  }
```

Initialiation of Member Objects

```
# include <iostream>
using namespace std ;
class Point {
   int x, y ;
public:
   Point(int x=0, int y=0) { this->x = x ; this->y = y ; }
} ;
class Rectangle {
   // rightBottom, leftTop의 순으로 호출됨
   Point rightBottom, leftTop  ;
public:
   // 1) 번생성자
   Rectangle(const Point& p1, const Point& p2=Point(0,0))
      // member initializer
      : leftTop(p1), rightBottom(p2) {}
   // 2) 번생성자
   Rectangle(int x1, int y1, int x2=0, int y2=0)
      // member initializer
      : leftTop(x1, y1), rightBottom(x2, y2) {}
   // 기본생성자
   Rectangle()
      // 생략가능함; default constructor를호출하므로
      : leftTop(), rightBottom() {}
} ;
```

```
int main() {
   // 기본생성자
   Rectangle r1 ;

   Point p ;
   // 1) 번생성자호출
   Rectangle r2(p, p) ;
   Rectangle r3(p) ;

   // 2) 번생성자호출
   Rectangle* pR4 =
      new Rectangle(100, 200) ;
   Rectangle* pR5 =
      new Rectangle(100) ;

   delete pR4 ;
   delete pR5 ;
}
```

Member Initializer의 용도

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

enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;

class Student ;

class School {
    // const 멤버는반드시member initializer로초기화해야함
    const string name ;
    // 멤버객체는반드시member initializer로초기화해야함
    vector<Student*> students ;
    float budget ;
public:
    School(const string& _name, int size) :
      name(_name), students(size) { budget = 0 ; }
} ;
```

Member Initializer의 용도

```
class Student {
    string name ;
    Grade grade ;
    // reference 멤버는 반드시 member initializer로 초기화 해야함
    const School& school ;
public:
    Student(const School& _school, const string& _name="")
        : school(_school) { this->name = name ; grade = FRESH ; }
} ;
```

복사 생성자

```
class Point {
    int x, y ;
public:
    Point(int x=0, int y=0) { this->x = x ; this->y = y ; } // 일반 생성자
    Point(const Point& pt) { x = pt.x ; y = pt.y ; }  // 복사 생성자
    int getX() const { return x ; }
    int getY() const { return y ; }
} ;
```

복사 생성자의 호출 상황

```
Point readPoint() {
    int x, y ;
    cin >> x >> y ;
    return Point(x, y) ;
}
void print(const Point pt) {
    cout << pt.getX() << ", " << pt.getY() << endl ;
}

int main() {
    Point pt1 ;

    pt1 = readPoint() ; // 3)
    Point pt2(pt1) ; // 1)

    print(pt2) ; // 2)
}
```

```
const int DEFAULT_MAX_SIZE = 10 ;
class MyStack {
    char* items ;
    int top ;
    const int maxSize ;
public:
    MyStack(int _maxSize=DEFAULT_MAX_SIZE) : maxSize(_maxSize) {
        items = new char[maxSize] ;
        top = 0 ;
    }
    MyStack(const char* const str) : maxSize(strlen(str)+DEFAULT_MAX_SIZE) {
        items = new char[maxSize] ;
        for ( int i = 0 ; i < strlen(str) ; i ++ )
                items[i] = str[i] ;
        top = strlen(str) ;
    }
    void push(char ch) ;
    char pop() ;
    …
} ;
```

기본 복사 생성자

- 복사 생성자가 명시되지 않았다면 컴파일러가 복사 생성자를 자동으로 생성한다.
- 자동으로 생성된 복사 생성자는 데이터 멤버 별로 대입문을 실행한다.

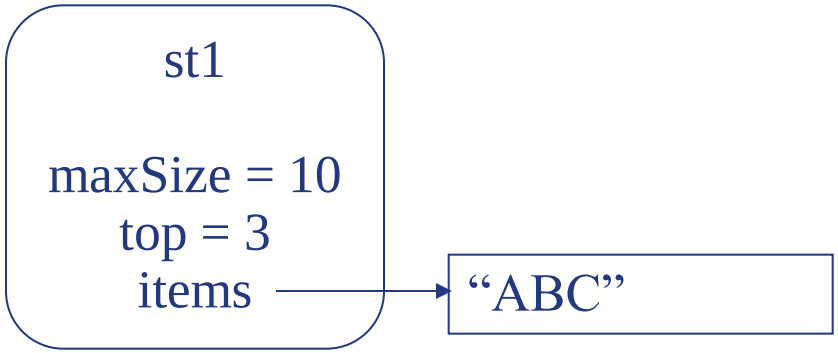
```
// 자동으로 생성된 복사 생성자
Point(const Point& pt) { x = pt.x ; y = pt.y ; }
```

기본 복사 생성자의 문제점

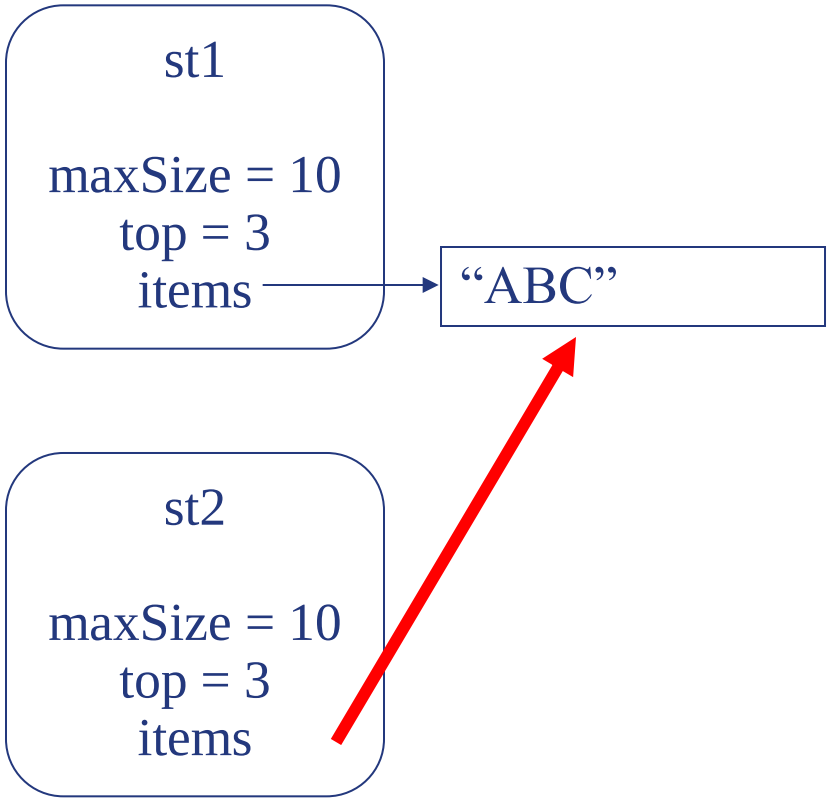
```
int main() {
    MyStack st1(“ABC”) ;
    MyStack st2(st1) ;
    st2.push(‘D’) ;
    cout << st1.pop() ; // ‘D’ not ‘C’ ;
}
```

Shallow Copy

MyStack st1("ABC") ;

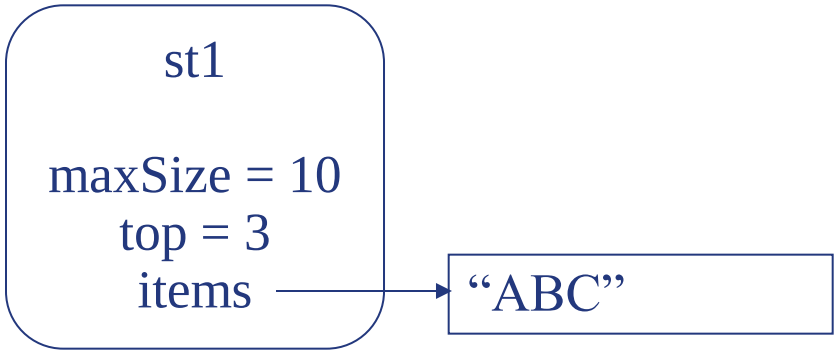


MyStack st2(st1) ;

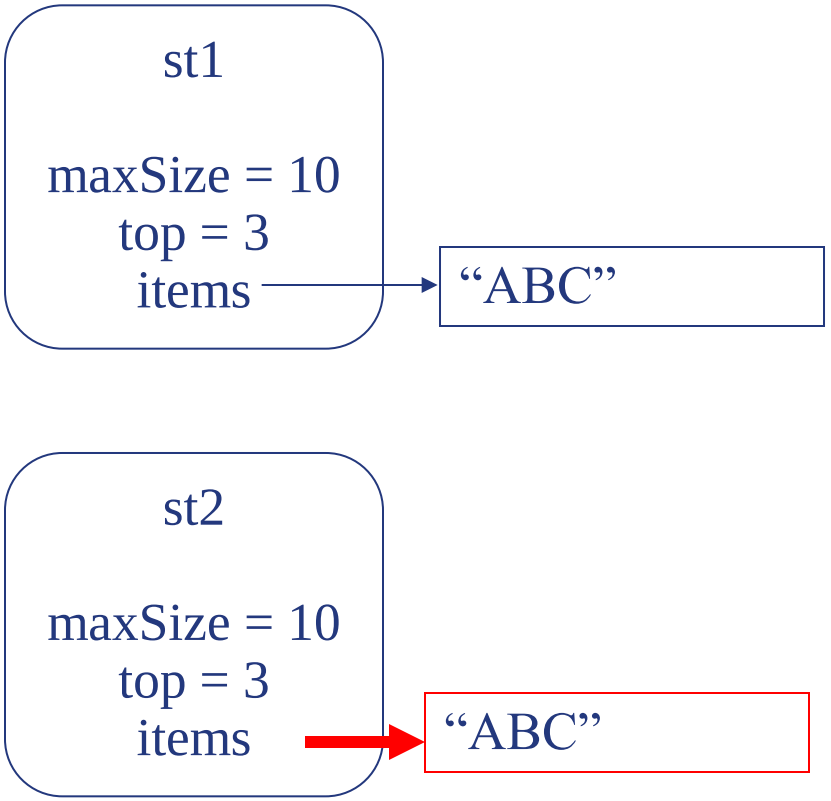


Deep Copy

MyStack st1("ABC") ;



MyStack st2(st1) ;



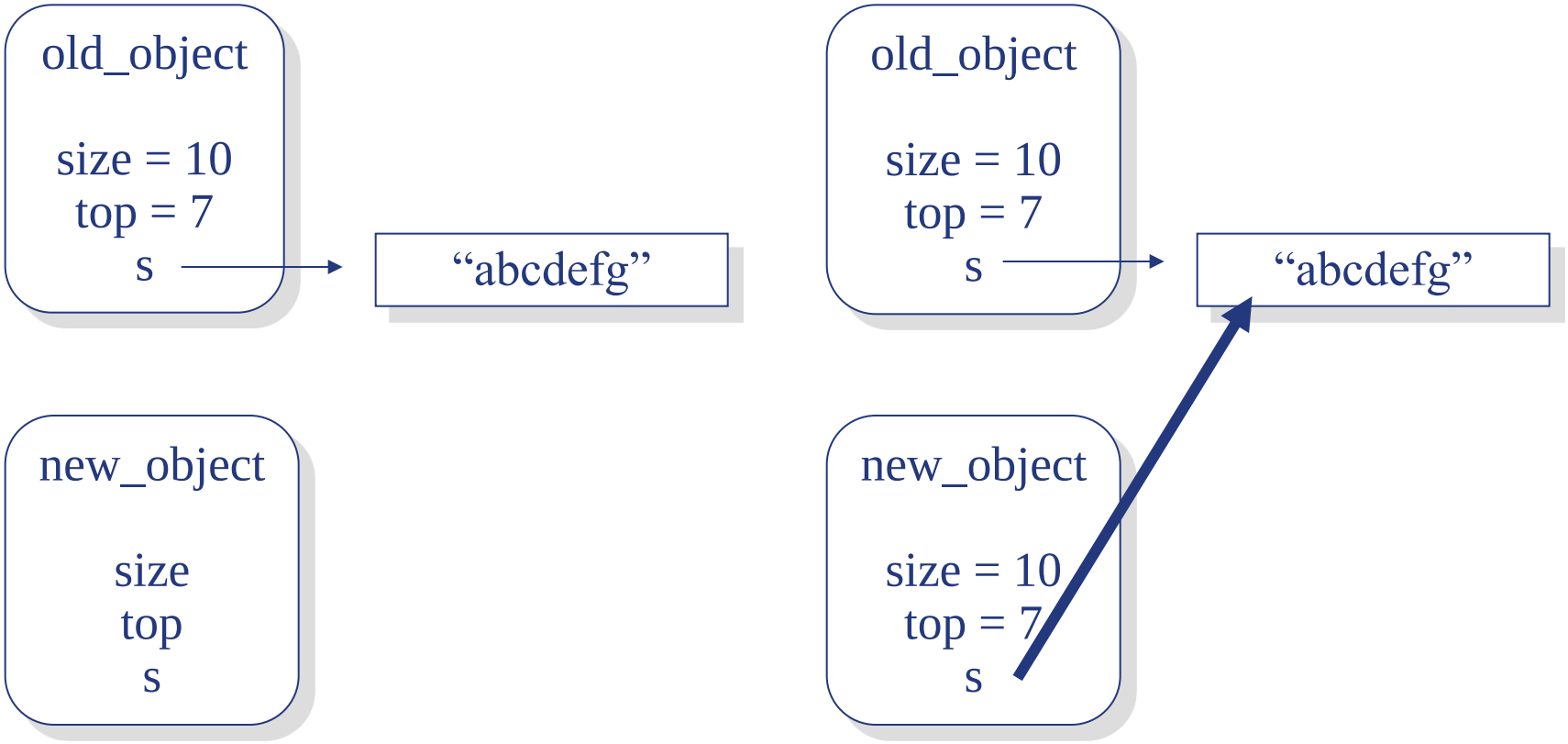
Copy Constructor의 필요성

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

class CharStack {
  private:
    int size ;
    int top ;
    char* s ;
  public:
    CharStack(int sz) {
        top = 0 ; s = new char[size=sz];
    }
    void push(char c) { s[top++] = c ; }
    char pop() { char r = s[--top ]; s[top] = '\0' ;
return r ; }
    void print() const {
        for ( int i = 0 ; i < top ; i ++ ) cout << s[i]  ;
        cout << endl ;
    }
} ;
```

```
int main() {
    CharStack s1(10) ;
    s1.push('a') ;
    s1.push('b') ;
    s1.print() ; // a b

    CharStack s2(s1) ;
    s2.print() ; // ab

    s1.pop() ;
    s1.print() ; // a
    s2.print() ; // a; why not ab

    s1.push('c') ;
    s1.print() ; // ac
    s2.print() ; // ac; why not ab

}
```

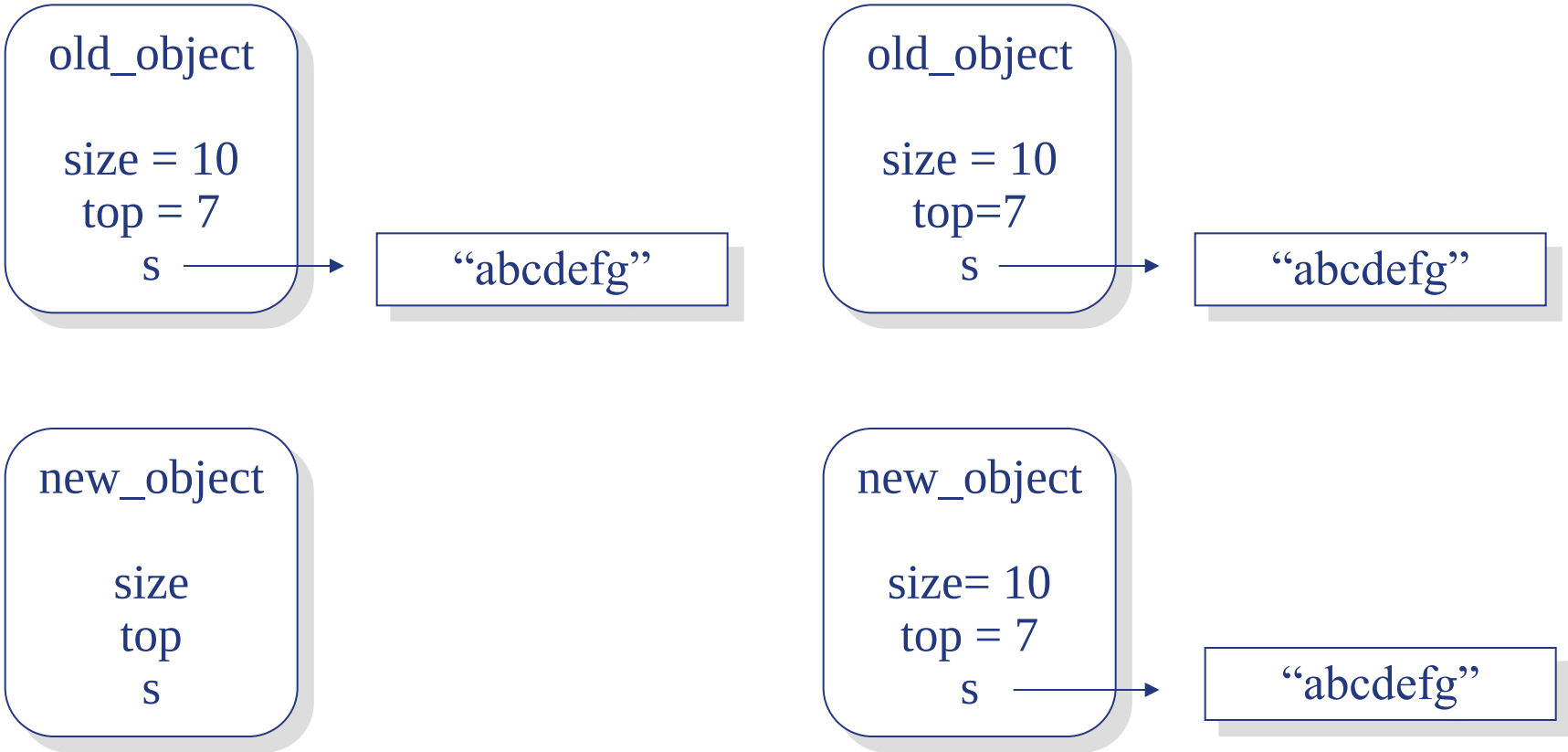
Shallow Copy

- Copying the addresses only
- CharStack new_object(old_object)



Deep Copy

- Copying the entire contents
- CharStack new_object(old_object)



Copy Constructor

- Default action of assignment: memberwise copy
- The Problem: shallow copy
- The Solution: copy constructor for deep copying
- The Copy Constructor for class X
  - X::X(const X&)

Copy Constructor

```
#include <iostream>
using namespace std ;
class CharStack {
  private:
    int size ;
    int top ;
    char* s ;
  public:
    CharStack(int sz) { top = 0 ; s = new char[size=sz]; }
   CharStack(const CharStack& another)
    : size(another.size), top(another.top) {
       s = new char[size] ;
       for ( int i = 0 ; i <= top ; i ++ ) s[i] = another.s[i] ;
   }
    void push(char c) { s[top++] = c ; }
    char pop() { char r = s[--top ]; s[top] = '\0' ; return r ; }
   void print() const {
    for ( int i = 0 ; i < top ; i ++ ) cout << s[i]  ;
    cout << endl ;
   }
} ;
```

```
int main() {
    CharStack s1(10) ;
    s1.push('a') ;
    s1.push('b') ;
    s1.print() ; // a b

    CharStack s2(s1) ;
    s2.print() ; // ab

    s1.pop() ;
    s1.print() ; // a
    s2.print() ; // ab

    s1.push('c') ;
    s1.print() ; // ac
    s2.print() ; // ab

}
```

Copy Constructor의 호출 상황

```
#include <iostream>
using namespace std ;
class CharStack {
    …
  public:
    CharStack(int sz) { top = 0 ; s = new char[size=sz]; }
    CharStack(const CharStack& another)
        : size(another.size), top(another.top) {
            s = new char[size] ;
            for ( int i = 0 ; i <= top ; i ++ ) s[i] = another.s[i] ;
            cout << "copy constructor invoked." << endl ;
    }
    …
} ;
void printStack(const CharStack s) {
    s.print() ;
}
CharStack createStack(int size) {
    CharStack newStack(size) ;
    return newStack ;
}
```

```
int main() {
    CharStack s1(10) ;
    s1.push('a') ;
    s1.push('b') ;
    s1.print() ;

    // 새 객체 초기화할때
    CharStack s2(s1) ;

    // 함수에서 반환할때
    createStack(10) ;

    // 함수의 인자로 전달될때
    printStack(s2) ;
}
```

Destructor

- The destructor ensures proper cleanup of objects
- In particular, many classes use some memory from the free store that is allocated by a constructor and de-allocate by a destructor

Destructor

```
# include <iostream>
using namespace std ;
class Point {
   int x, y ;
public:
   Point(int x=0, int y=0) {
      this->x = x ; this->y = y ;
      cout << "\tPoint " ; print() ; cout << " constructed." << endl ;
   }
   ~Point() { cout << "\tPoint " ; print() ; cout << " destructed." << endl ; }
   void print() const { cout << "(" << x << ", " << y << ")" ; }
} ;
```

Destructor

```
Point gP(100, 100) ;
int main() {
    cout << "P1\n" ;
    Point p1 ;

    Point* pP2 ;
    {
       cout << "\nP2\n" ;
       Point p3(3, 3) ;
       cout << "\nP3\n" ;
       pP2 = new Point(2, 2) ;
    }
    delete pP2 ;
}
```

```
 Point (100, 100) constructed.
P1
        Point (0, 0) constructed.

P2
        Point (3, 3) constructed.

P3
        Point (2, 2) constructed.
        Point (3, 3) destructed.
        Point (2, 2) destructed.
        Point (0, 0) destructed.
        Point (100, 100) destructed.
```

Destructor of Member Objects

```
# include <iostream>
using namespace std ;
class Point {
   int x, y ;
public:
   Point(int x=0, int y=0) {
      this->x = x ; this->y = y ;
      cout << "\tPoint " ; print() ; cout << " constructed." << endl ;
   }
   ~Point() { cout << "\tPoint " ; print() ; cout << " destructed." << endl ; }
   void print() const { cout << "(" << x << ", " << y << ")" ; }
} ;
class Rectangle {
   Point rightBottom, leftTop  ;
public:
   Rectangle(const Point& p1, const Point& p2=Point(0,0))
      : leftTop(p1), rightBottom(p2) {
         cout << "Rectangle: " ; print() ; cout << " constructed." << endl ;
      }
   Rectangle(int x1, int y1, int x2=0, int y2=0)
      : leftTop(x1, y1), rightBottom(x2, y2) {
      cout << "Rectangle: " ; print() ; cout << " constructed." << endl ;
   }
   Rectangle() { cout << "Rectangle: " ; print() ; cout << " constructed." << endl ; }
   ~Rectangle() { cout << "Rectangle: " ; print() ; cout << " destructed." << endl ; }
   void print() const { leftTop.print() ; rightBottom.print() ; }
} ;
```

Destructor of Member Objects

```
int main() {
    Point p(2, 2) ;
    Rectangle r2(p) ;
}
```

```
Point (2, 2) constructed.
      Point (0, 0) constructed.
Rectangle: (2, 2)(0, 0) constructed.
      Point (0, 0) destructed.
Rectangle: (2, 2)(0, 0) destructed.
      Point (2, 2) destructed.
      Point (0, 0) destructed.
      Point (2, 2) destructed.
```

생성자/소멸자 호출 순서 요약

```
class Rectangle {
   Point rightBottom ;
   Point leftTop  ;
public:
   Rectangle(int x1, int y1, int x2=0, int y2=0)
      : leftTop(x1, y1), rightBottom(x2, y2)
      { /*  */  }
   ~Rectangle() { /*  */ }
} ;
```

rightBottom

leftTop

Rectangle() 본문

~Rectangle()

leftTop

rightBottom

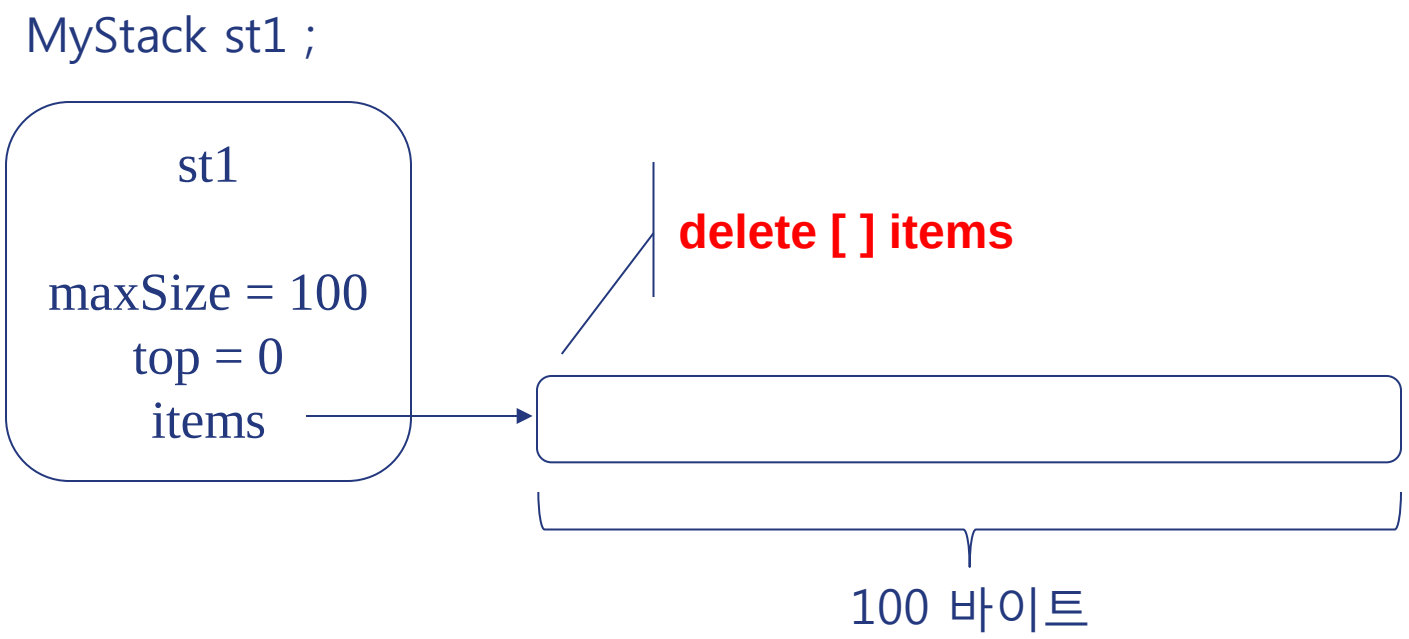
Destructor의 용도

- What if no destructor for CharStack?

```
#include <iostream>
using namespace std ;
class CharStack {
  private:
    int size ;
    int top ;
    char* s ;
  public:
    CharStack(int sz) {
    top = 0 ; s = new char[size=sz];
    cout << "constructor invoked for Stack[" << size << "]" << endl ;
  }
  CharStack(const CharStack& another)
    : size(another.size), top(another.top) {
      s = new char[size] ;
      for ( int i = 0 ; i <= top ; i ++ ) s[i] = another.s[i] ;
  }
  ~CharStack() {
    delete [] s ;
    cout << "destructor invoked for Stack[" << size << "]" << endl ;
  }
  …
} ;
```

```
int main() {
    CharStack s1(100000) ;
    {
        CharStack s2(20000) ;
    }
}
```

소멸자의 역할



Exercise

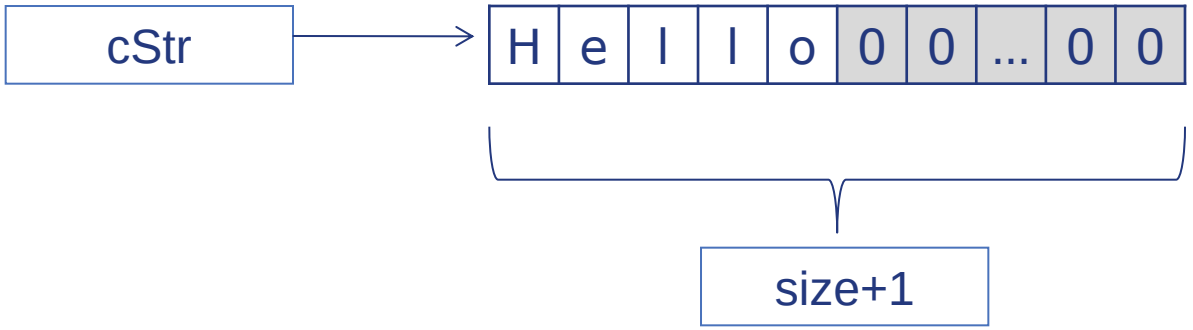
- Implement class MyString equivalent to STL string.

```
class MyString {
    char* str ;
    int size ;
public:
    …
} ;
```

```
int main() {
    // constructor test
    MyString str1("ABC") ;
    str1.print() ;

    // copy constructor test
    {
        MyString str2(str1) ;
        // set test
        str2.set(0, 'D') ;
        str2.print() ;
    }
    str1.print() ;

    // length, at test
    for ( int i = 0 ;i < str1.length() ; i ++ )
        cout << str1.at(i) ;
}
```

```
ABC
DBC
ABC
ABC
```

MyStringList



MyStringList

```
class MyStringList {
private:
    MyString** ppMyString ;
    int size ;
public:
    …
} ;
```