포인터와 동적 배열 2
(Chapter 9 Pointers and Dynamic Arrays)

숭실대학교
김강희 교수
(khkim@ssu.ac.kr)

순서

- 이론:
  - 포인터
  - 동적 배열
  - this 포인터
  - 소멸자
  - 복사 생성자

배열 변수들

- 배열의 원소들은 연속된 메모리 주소들에 저장된다.
  - 배열 변수는 일종의 포인터 변수이다.
- 예시:

```
int a[10];
int * p;
```

  - a 와 p 는 둘 다 포인터 변수이다!
- 그러나, 다음과 같은 차이점이 있다.

```
int a[10];
typedef int* IntPtr;
IntPtr p;
p = a;        // legal
a = p;        // illegal: a 는 상수 포인터이다. 즉 "const int * " 타입이다.
```

동적 배열들

- 배열 변수들
  - 사실상 포인터 변수들임!
- 표준 배열
  - 고정된 크기를 가짐
- 동적 배열
  - 코딩 시점에서는 크기가 정해지지 않음
  - 실행 시점에 크기가 결정됨

동적 배열들

- 할당 예시:

```
typedef double * DoublePtr;
DoublePtr d;
d = new double[10];     //Size in brackets
```

- 소멸 예시:

```
delete [] d; // 여기서 bracket 은 d 가 array 를 가리키고 있음을 의미함
d = NULL; // 좋은 습관!
```

함수가 배열을 리턴하는 방법

- 배열 타입은 함수의 리턴 타입이 될 수 없다.
- 예시:
  int [] someFunction(); // illegal
- 대안:
  int* someFunction(); // legal

포인터 산수

- 포인터는 오직 덧셈과 뺄셈만 가능하다.
  - 곱셈과 나눗셈은 불가하다.
  - ++ 또는 -- 연산자를 사용할 수 있다.
- 예시:

```
for (int i = 0; i < arraySize; i++)
        cout << *(d + i) << " " ;
```

- 동등한 예시:

```
for (int i = 0; i < arraySize; i++)
        cout << d[i] << " " ;
```

다차원 동적 배열들

- 포인터들의 배열을 만들어, 다차원 동적 배열을 만들 수 있다.
- 예시: // 3x4 크기의 2차원 배열을 만들자

```
typedef int* IntArrayPtr;
IntArrayPtr *m = new IntArrayPtr[3];
for (int i = 0; i < 3; i++)
    m[i] = new int[4];
```

클래스와 포인터

- '->' 연산자
  - dereference operator '*' 와 dot operator '.' 를 합친 의미이다.
  - 예시:
    ```
    MyClass *p;
    p = new MyClass;
    p->grade = "A";
    (*p).grade = "A"; // '->' 표현과 동등한 표현이다.
    ```
- this 포인터
  - 멤버 함수는 calling object 의 포인터(this)가 필요한 경우가 있다.
  - ```
    Class Matrix {
    private:    int dy; int dx; int **array;
    public:     int print_dy();
    };
    ```
  - 멤버 함수 안에서 다음 두 표현은 동일한 의미이다.
    - cout << dy;
    - cout << this->dy;
    ```

‘=‘ 연산자 오버로딩

- 할당 연산자는 레퍼런스를 리턴한다.
  - 객체 a 와 b 에 대해서 ‘a = b‘ 표현은 ‘a.=(b)’ 로 바꾸어 이해한다.
  - 다음과 같은 할당 체인도 가능하다.
    - 예: a = b = c;
      - a 와 b 에 c 값을 할당한다.
      - ‘(a = b) = c‘ 로 이해할 때 c 의 관점에서는 ‘a = c‘ 와 같은 표현이다.
      - 따라서 ‘(a = b)’ 표현은 레퍼런스 a 를 리턴해야 한다.

'=' 연산자 오버로딩

- string class 예시:

```
StringClass& StringClass::operator=(const StringClass& rtSide)
{
        if (this == &rtSide)        // 우변과 좌변이 같은 객체일 때
                return *this;
        else
        {
                capacity = rtSide.length;
                length = rtSide.length;
                delete [] a;
                a = new char[capacity];
                for (int i = 0; i < length; i++)
                        a[i] = rtSide.a[i];
                return *this;
        }
}
```

'=' 연산자 오버로딩

- Matrix class 예시:

```
219  Matrix& Matrix::operator=(const Matrix& obj)
220  {
221    if (this == &obj) return *this;
222    if ((dx != obj.dx) || (dy != obj.dy)) {
223      if (array != NULL) dealloc();
224      alloc(obj.dy, obj.dx);
225    }
226    for (int y = 0; y < dy; y++)
227      for (int x = 0; x < dx; x++)
228        array[y][x] = obj.array[y][x];
229    return *this;
230  }
```

부록: Matrix 생성자와 소멸자 (수정 버전)

```
16  void Matrix::alloc(int cy, int cx) {
17    if ((cy <= 0) || (cx <= 0)) {
18      dy = 0;
19      dx = 0;
20      array = NULL;
21      nAlloc++;
22      return;
23    }
24    dy = cy;
25    dx = cx;
26    array = new int*[dy];
27    for (int y = 0; y < dy; y++)
28      array[y] = new int[dx];
29    for (int y = 0; y < dy; y++)
30      for (int x = 0; x < dx; x++)
31        array[y][x] = 0;
32
33    nAlloc++;
34  }
35
36  Matrix::Matrix() { alloc(0, 0); }
```

```
38  void Matrix::dealloc() {
39    if (array != NULL) {
40      for (int y = 0; y < dy; y++)
41        delete[] array[y];
42      delete[] array;
43      array = NULL;
44    }
45
46    nFree++;
47  }
48
49  Matrix::~Matrix() { dealloc(); }
```

얕은 복사와 깊은 복사

- shallow copy
  - 객체 간에 member-wise copy 를 수행하는 경우이다.
  - default assignment 연산과 default copy constructor 에 적용된다.
- deep copy
  - 객체 안에 포인터 멤버가 존재하면 shallow copy 는 문제를 일으킨다.
    - 메모리 누수와 이중 소멸
  - 포인터 멤버가 가리키는 데이터를 실질적으로 얻을 수 있도록, 할당 연산자와 복사 생성자를 직접 코딩해야 한다.

소멸자

- 생성자의 반대 동작을 수행한다.
  - 스택 객체가 out-of-scope 에 도달하면 자동으로 호출된다.
  - 힙 객체가 delete 되면 자동으로 호출된다.
  - 디폴트 버전은 오직 멤버 변수들만을 소멸시킨다.
  - 예시:

```
38  void Matrix::dealloc() {
39    if (array != NULL) {
40      for (int y = 0; y < dy; y++)
41        delete[] array[y];
42      delete[] array;
43      array = NULL;
44    }
45
46    nFree++;
47  }
48
49  Matrix::~Matrix() { dealloc(); }
```

복사 생성자

- 다음 경우에 자동 호출된다.
  1. Matrix B(A) 와 같이 객체 선언할 때
  2. B = myfunc() 와 같이 함수가 객체를 리턴할 때
  3. call-by-value 호출 방식을 사용하는 함수의 인자로 객체가 전달될 때, 즉 myfunc2(B)
- 디폴트 생성자는 member-wise copy, 즉 shallow copy 를 수행한다.
- deep copy 가 필요하면, 자신만의 복사 생성자를 코딩하라.
  - 예시:

```
58  Matrix::Matrix(const Matrix *obj) {
59    alloc(obj->dy, obj->dx);
60    for (int y = 0; y < dy; y++)
61      for (int x = 0; x < dx; x++)
62        array[y][x] = obj->array[y][x];
63  }
64
65  Matrix::Matrix(const Matrix &obj) {
66    alloc(obj.dy, obj.dx);
67    for (int y = 0; y < dy; y++)
68      for (int x = 0; x < dx; x++)
69        array[y][x] = obj.array[y][x];
70  }
```