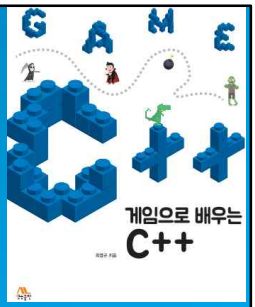
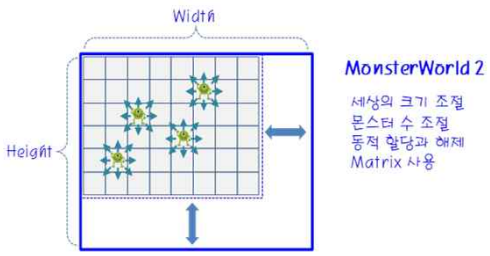
게임으로 배우는 C++



CHAPTER

08 객체와 포인터

몬스터 월드2 (조절되는 세상)



- Monster World 2
  - 몬스터 월드의 크기를 마음대로 조절함.
  - 동적으로 메모리를 할당을 사용
  - 몬스터들도 동적으로 생성하고 해제함

2

8장 학습 목표

- 객체와 포인터의 관계를 이해한다.
- 동적 메모리 할당과 해제를 이해하고 활용할 수 있도록 한다.
- 깊은 복사와 복사 생성자의 관계를 이해한다.
- this 포인터를 이해한다.
- 정적 멤버 변수와 정적 멤버 함수를 이해한다.
- 2차원 배열을 동적으로 할당하는 방법을 이해한다.

3

8.1 객체와 포인터

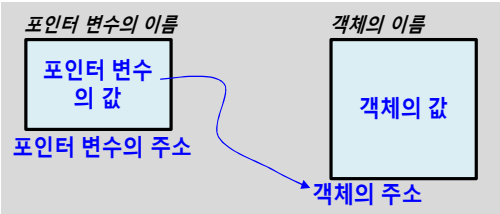
- 포인터란?
- 객체와 포인터
- 포인터와 객체의 멤버 접근
- 정적 메모리 할당과 동적 메모리 할당

4

포인터란?

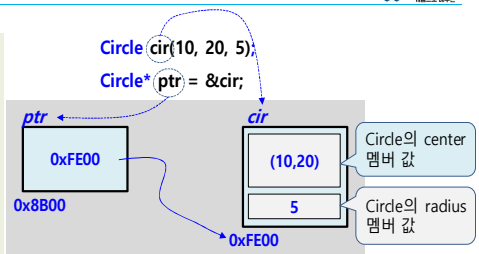
- 포인터: 어떤 주소를 저장하기 위한 변수
  - 이름: 포인터 변수도 적절한 이름을 가짐
  - 주소: 포인터도 변수이므로 메모리 공간과 주소가 있음
  - 값: 메모리 공간에 들어 있는 값. 다른 변수나 객체의 주소 저장.

- 객체와 포인터

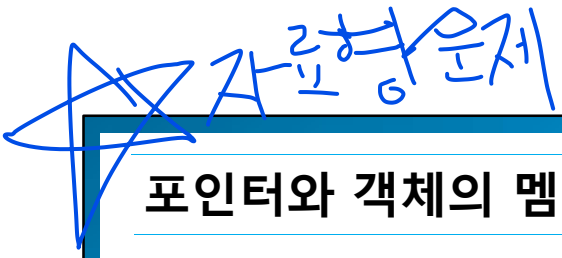


객체와 포인터

```
struct Point {
    int x, y;
    Point(int xx = 0, int yy = 0)
       : x(xx), y(yy) { }
};

class Circle {
    Point center;
    int radius;
public:
    Circle(int cx=0, int cy=0, int r=0)
       : center(cx, cy), radius(r) {}
    double perimeter() { return 2*3.14159*radius; }
};
```




```
Circle cir(10,20,5);  // 중심이 (10,20), 반지름 5인 객체 cir 생성
Circle* ptr = &cir;   // Circle의 포인터 ptr생성. cir의 주소 복사
```

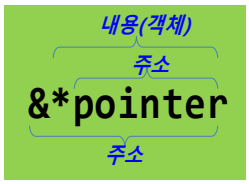


포인터와 객체의 멤버 접근

- 멤버의 접근
  - **객체에서 멤버**를 선택하기 위해 "."
  - 객체의 **포인터에서는 멤버** 추출을 위해 "->"

```
Circle cir(10,20,5);
Circle* ptr = &cir;

double peri = cir.perimeter();
double peri = ptr->perimeter();
```

```
Point center(10,20);
Point* pCenter = &center;

center.x = 40;
pCenter->x = 40;
```

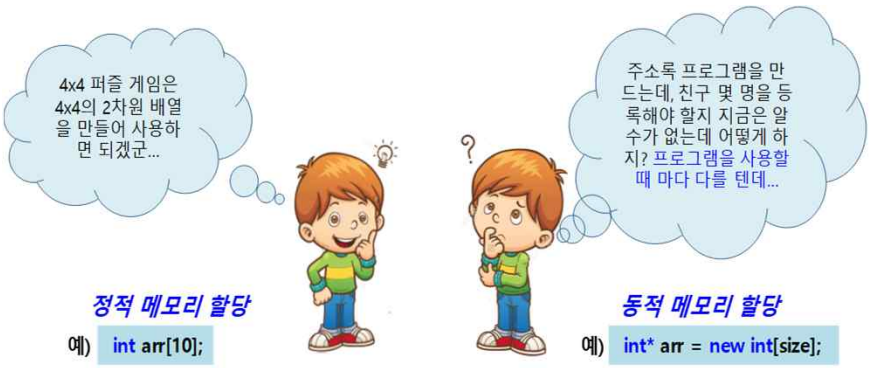


- 역참조 연산자 * :포인터(주소)에 들어있는 값을 반환

```
center.x == pCenter->x == (*pCenter).x == (&center)->x
```

7

정적 메모리 할당과 동적 메모리 할당



8

8.2 동적 메모리 할당과 해제

- 동적 메모리 할당(dynamic memory allocation)
- 동적 메모리 할당과 해제
- 정적/동적 메모리 할당의 비교
- Lab: Vector 클래스

동적 메모리 할당(dynamic memory allocation)

- 프로그램 실행 중에 필요한 메모리의 크기를 시스템에 요청.
- 시스템은 힙(heap)이라고 부르는 큰 공간을 관리하고 있는데, 프로그램에서 요청하는 공간을 확보(할당)하여 시작 주소를 돌려줌.
- 할당된 메모리 시작 주소를 반드시 포인터 변수에 저장
- 이 포인터를 이용해 할당된 메모리를 사용(잃어버리면 안됨)
- 사용이 끝나면 할당된 메모리를 시스템에 돌려줌.

- 다음 코드가 가능할까?

```
int x;
cin >> x;       // x를 입력 받음
int arr[x];     // 동적으로 배열 할당 ? (ERROR!)
```

  - 그럼 어떻게? new, delete

동적 메모리 할당과 해제

- 할당

```
new data_type;
new data_type(인수1, 인수2, ...);
new data_type [size];
```

```
int* pi = new int;
float* pf = new float;
Complex* pc = new Complex(2,3);
*pi = 10;
*pf = 3.14f;
pc->real = 1.0;
cout << pc->imag;
```

```
int* pi = new int[10];
float* pf = new float[10];
Complex* pc = new Complex[5];

*pi = 10;       // pi[0] = 10;
pf[3] = 3.14f;
pc[2].real = 30;
```

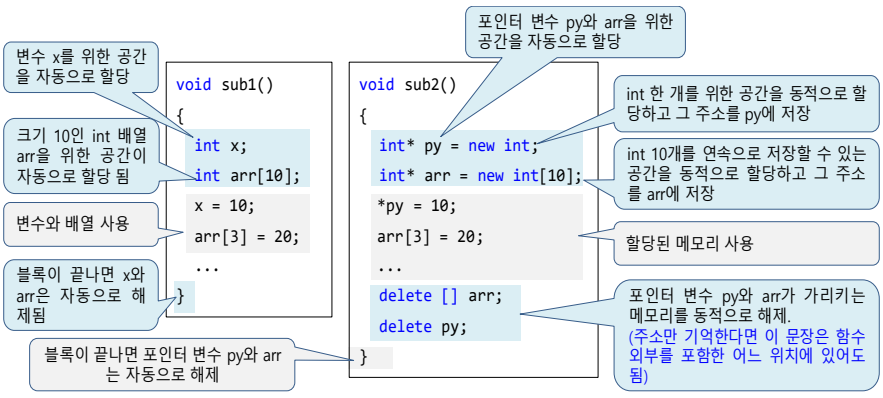
- 해제

```
delete 포인터;
delete [] 포인터;
```

```
delete pi;
delete pf;
delete pc;
```

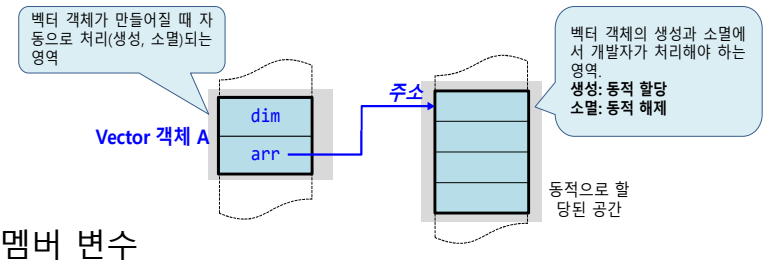
```
delete [] pi;
delete [] pf;
delete [] pc;
```


정적/동적 메모리 할당의 비교



Lab: Vector 클래스

- 벡터 객체의 구조



- 멤버 변수
  - `int dim` : 벡터의 차원은 정수
  - `double* arr` : 각 차원에 데이터로는 실수 값의 배열. 배열의 길이를 고정시킬 수 없으므로 double형 포인터 변수 사용.
- 멤버 함수
  - 생성자, 소멸자, print(), set(), setRand()

↳ 동적할당

벡터 클래스 구현

  - 생성자, 소멸자, print(), set(), setRand()

```
class Vector {
    int dim;
    double* arr;
public:
    Vector(int d = 0)
        : dim(d) { arr = new double[dim]; }
    ~Vector() { delete[] arr; }
    void setRand(int max = 100) {
        for (int i = 0; i<dim; i++)
            arr[i] = rand() % (max * 10) / 10.0;
    }
    void print(char *str = "Vector") {
        cout << str << "[" << dim << "] = < ";
        for (int i = 0; i < dim; i++)
            cout << arr[i] << " ";
        cout << ">\n";
    }
};
```

```
void main()
{
    Vector u(3), v(5), w;
    u.setRand();
    v.setRand();
    u.print(" U ");
    v.print(" V ");
    w.print(" W ");
}
```

```
C:₩WINDOWS₩system32₩cmd....
U [3] = < 4.1 46.7 33.4 >
V [5] = < 50 16.9 72.4 47.8 35.8 >
W [0] = < >
```

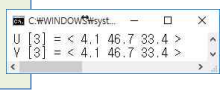
8.3 객체의 얕은 복사 문제

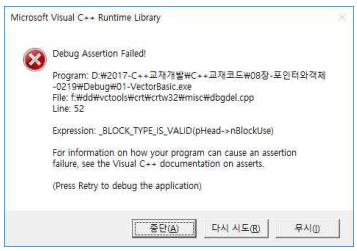
- 얕은 복사 문제 상황들
- 얕은 복사에서의 실행 오류 발생 상황

얕은 복사 문제 상황들

- 실행 오류? 대입 연산자?

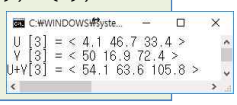
```
void main()
{
    Vector u(3), v(3);
    u.setRand();
    v = u;
    u.print(" U ");
    v.print(" V ");
}
```

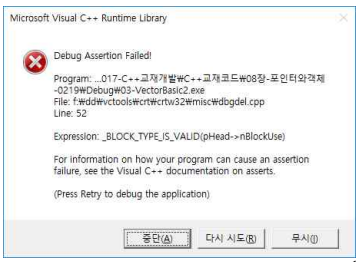




- 실행 오류? 복사 생성자?

```
  void add(Vector a, Vector b) {
    for (int i = 0; i < dim; i++)
      arr[i] = a.arr[i]+b.arr[i];
  }
void main()
{
    Vector u(3), v(3), w(3);
    ...
    w.add(u,v);
    ...
}
```

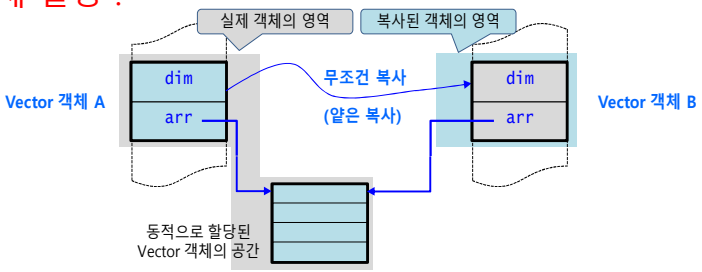




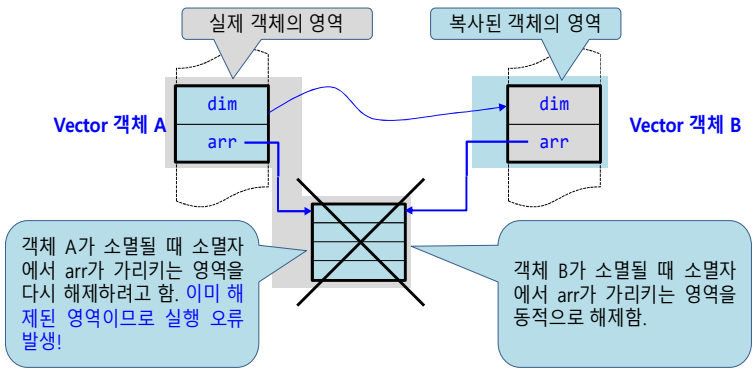
얕은 복사(디폴트 복사생성자,대입연산자)

- 얕은 복사

- 문제 발생 !



얕은 복사에서의 실행 오류 발생 상황



8.4 동적 메모리와 깊은 복사

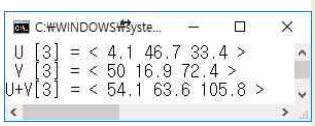
- 소심한 해결 방안
- 근본적인 해결 방안 → 깊은 복사
- 깊은 복사 구현 방법

소심한 해결 방안

- 대입 연산자나 복사 생성자가 호출되지 않도록 함
  → 참조자 또는 포인터를 매개변수로 사용함
- 참조자 매개변수

```
void add(Vector& a, Vector& b) {...}
```

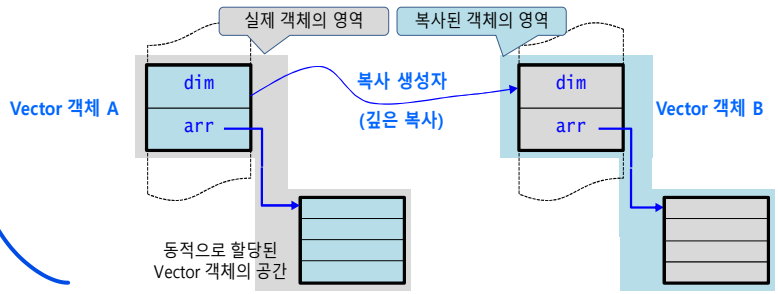
- 포인터 매개변수




```
class Vector {
    ...
    void add(Vector* a, Vector* b) {
        for (int i = 0; i < dim; i++)
            arr[i] = a->arr[i] + b->arr[i];
    }
};
void main() {
    Vector u(3), v(3), w(3);
    ...
    w.add(&u, &v);
    ...
}
```

근본적인 해결 방안 → 깊은 복사

- 동작 방법



21

깊은 복사 구현 방법

```
class Vector {
    ...
    void clone(Vector& a) {
        if (dim > 0) delete[] arr;
        dim = a.dim;
        arr = new double[dim];
        for (int i = 0; i < dim; i++)
            arr[i] = a.arr[i];
    }

    void operator = (Vector& a) { clone(a); }

    Vector(Vector& a) : dim(0) { clone(a); }
};
```

```
C:₩WINDOWS₩system3...
U [3] = < 4.1 46.7 33.4 >
V [3] = < 4.1 46.7 33.4 >
U+V[3] = < 8.2 93.4 66.8 >
```

```
void main()
{
    Vector u(3), v, w(3);
    u.setRandom();
    v = u;
    w.add(u, v);
    u.print(" U ");
    v.print(" V ");
    w.print("U+V");
}
```

22

8.5 this 포인터

- this 포인터란?

this 포인터란?

- 객체 자신의 메모리상의 주소를 나타내는 포인터
  - 그 함수를 실행하는 현재 객체의 주소

```
class Point {
    int x, y;
public:
    Point(int x = 0, int y = 0) {
        this->x = x;
        this->y = y;
    }
    void print(char* msg = "P=") {
        printf("%s(%d,%d)\n", msg,
                this->x, this->y);
    }
    void whereAmI() {
        printf("주소=%x\n", this);
    }
};
```

```
P = (1,2)
주소=e5f8cc
Q = (3,4)
주소=e5f8bc
```

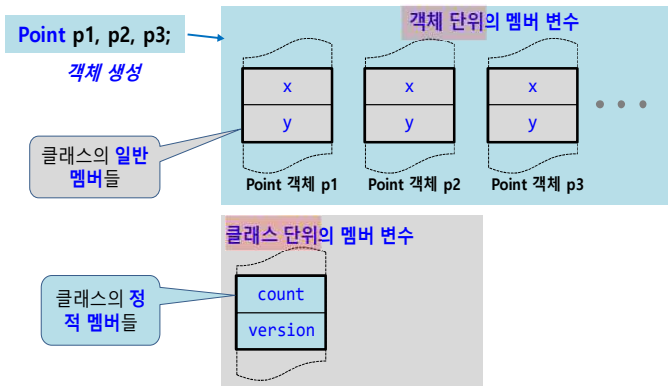
```
void main()
{
    Point p(1, 2), q(3, 4);
    p.print(" P = ");
    p.whereAmI();
    q.print(" Q = ");
    q.whereAmI();
}
```

8.6 정적 멤버

- 클래스의 일반 멤버와 정적 멤버
- 정적 멤버 변수
- 정적 멤버 함수
- Lab: Point 객체의 개수

25

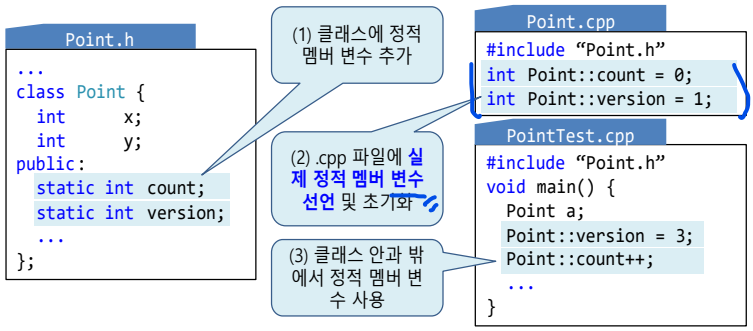
클래스의 일반 멤버와 정적 멤버



26

정적 멤버 변수

- 사용 예
  - 현재 Point 객체의 개수
  - Point 클래스의 버전이나 개발 정보



정적 멤버 함수

- 특징
  - "객체에서 호출되지 않는다"
  - 일반 멤버 함수와 달리 "나" 또는 "내 객체"가 없음.
    - This 포인터를 사용 불가.
    - 일반 멤버 변수를 사용 불가
  - 상속에서 함수의 재정의(overriding)를 사용할 수 없음

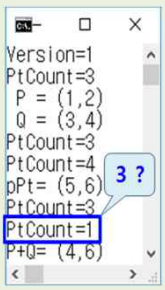
- Java 언어
  - Math 클래스
  - Sin(), cos(), ...
  - Math::sin()

Lab: Point 객체의 개수

- 객체의 개수 관리 방법
  - 맨 처음에는 Point::count 변수를 0으로 초기화
  - 생성자: count를 1 증가, 소멸자: count를 1 감소

```
class Point  {
  int x, y;
  static int count;

public:
  static void printCount() { printf("PtCount=%d\n", count); }

  Point(int x0 = 0, int y0 = 0) : x(x0), y(y0) { count++;  }
  ~Point() { count--; }
  void print(char* msg="P=") { printf("%s(%d,%d)\n",msg,x,y); }
  void add(Point a, Point b) { x = a.x + b.x; y = a.y + b.y; }
};
```


```
int Point::count = 0;     // Point 객체의 개수

void main()
{
    Point p(1,2),q(3,4),r;
                          Point::printCount(); // 3

    p.print(" P = ");
    q.print(" Q = ");

    Point* pPt;           Point::printCount(); // 3
    pPt = new Point(5,6); Point::printCount(); // 4
    pPt->print("pPt= ");

    delete pPt;           Point::printCount(); // 3

    r.add(p, q);          Point::printCount(); // 3? 1?
    r.print("P+Q= ");
}
```




```
  Point(const Point& p) : x(p.x), y(p.y) { count++; }
```

8.7 응용: 2차원 배열의 동적 할당

- Lab: 2차원 배열의 동적 할당
- 2차원 배열의 동적 할당 구조
- 동적 할당 및 해제 함수
- Lab: Matrix 클래스

31

Lab: 2차원 배열의 동적 할당(심화)

- 다음 코드가 가능할까?

```
int **arr = new int [rows][cols];     // 잘못된 코드
arr[i][j] = 0;
...
delete [][] arr;                      // 잘못된 코드
```

- 함수의 원형: 좋지 않은 설계

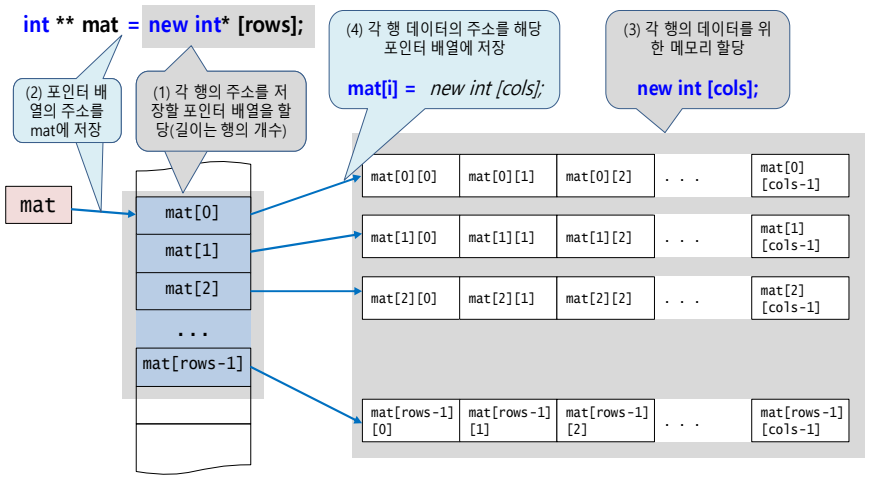
```
int findMaxPixel( int a[][5], int h, int w );
```

- 함수의 원형: 좋은 설계

```
int findMaxPixel( int** a, int h, int w );
```

32

2차원 배열의 동적 할당 구조



동적 할당 및 해제 함수

- 동적 할당

```
int** alloc2DInt (int rows, int cols) {
    if( rows <= 0 || cols <= 0 ) return NULL;
    int** mat = new int* [ rows ];
    for (int i=0 ; i<rows ; i++ )
        mat[i] = new int [cols];
    return mat;
}
```

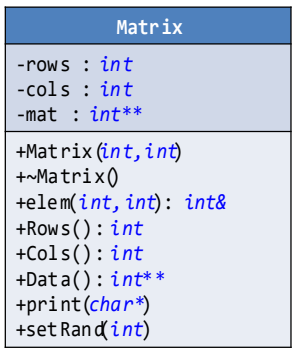
- 동적 해제

```
void free2DInt ( int **mat, int rows, int cols=0) {
    if( mat != NULL ) {
        for( int i=0 ; i<rows ; i++ )
            delete [] mat[i];
        delete [] mat;
    }
}
```

Lab: Matrix 클래스

- 속성
  - 행렬의 크기는 멤버 변수 rows와 cols가 나타낸다.
  - 행렬에는 int 정수가 저장 → int** mat;

- 행위
  - 생성자, 소멸자
  - Rows(), Cols(), Data()
  - print()
  - setRand()



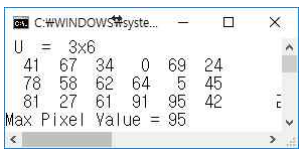
35

Matrix 클래스 구현

```
class Matrix {
    int rows, cols;
    int** mat;
public:
    Matrix(int r=0, int c=0)
      : rows(r),cols(c),mat(NULL) {
        ...
    }
    ~Matrix() {...}
    int& elem(int x,int y){...}
    int Rows() { return rows; }
    int Cols() { return cols; }
    int** Data(){ return mat; };
    void print(char *str = "Mat"){...}
    void setRand(int val = 100){...}
};
int findMaxPixel(int** a,int rows,int cols)
{
    int max = a[0][0];
    for (int i=0; i<rows; i++)
    for (int j=0; j<cols; j++)
      if (max < a[i][j]) max=a[i][j];
    return max;
}
```

동적 할당과 해제 코드

```
void main()
{
    Matrix u(3, 6);
    u.setRand();
    u.print(" U = ");
    cout << "Max Pixel Value = "
         << findMaxPixel(u.Data(),
                u.Rows(), u.Cols())
         << endl;
}
```

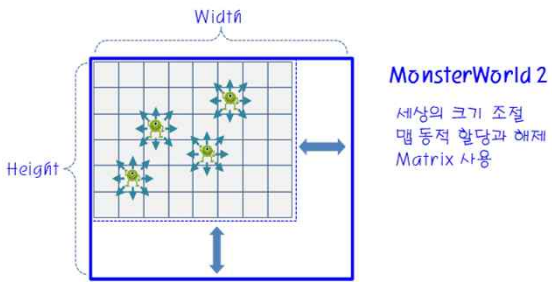


36

8.8 응용: MonsterWorld 2

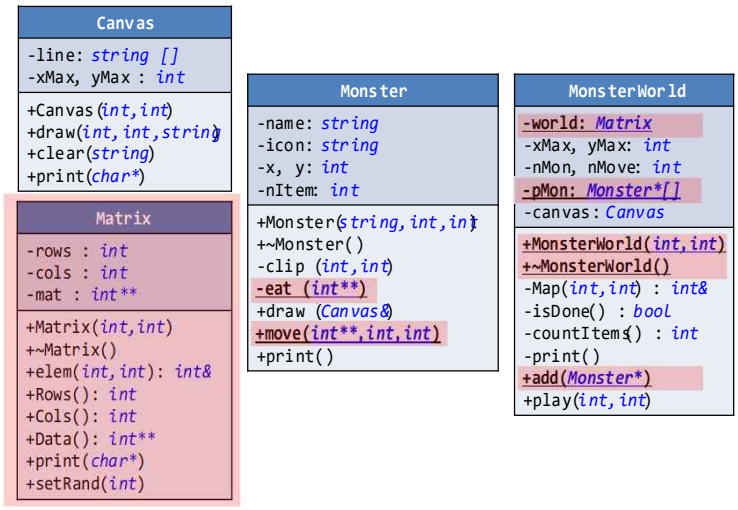
- 몬스터 월드2 : 세상의 크기 조절
- 클래스 수정
- 고찰

Monster World 2: 세상의 크기 조절



- 동적으로 할당한 맵 사용 → Matrix 클래스를 사용
- 몬스터도 동적으로 할당 → 몬스터 포인터의 배열을 사용
- 게임이 종료되면 맵과 몬스터를 동적으로 해제
- 프로그램의 동작과 화면 구성은 7장과 동일

클래스 다이어그램



클래스 수정

- Monster

```
        void eat(int map[DIM][DIM]);
        void move(int map[DIM][DIM], int maxx, int maxy);
```

```
        void move(int** map, int maxx, int maxy) {
        void eat(int** map) {
```

- MonsterWorld

```
class MonsterWorld
{
        Matrix world;
        Monster* pMon[MAXMONS];
        ...
        int& Map(int x, int y) { return world.elem(x,y); }
        void print() {
                ...
                for (int i = 0; i < nMon ; i++)
                        pMon[i]->draw(canvas);
                ...
```

고찰

- 세상의 크기 변경 → 동적 메모리 할당
- 몬스터도 동적으로 생성해 추가
  - 게임이 진행되는 도중에 몬스터를 추가하거나 기존의 몬스터를 제거할 수 있는 틀을 제공
- Matrix와 MonsterWorld에서 깊은 복사를 사용하려면?
  - 기본 복사를 사용할 수 없음
  - 깊은 복사를 하는 복사생성자와 대입연산자를 구현해야 함
- 정적 멤버 변수와 함수 사용
  - 현재 몬스터의 수 관리

41

8장 요약문제, 연습문제, 실습문제

42

감사합니다!

43