Matrix Class 사용하기

숭실대학교
김강희 교수
(khkim@ssu.ac.kr)

순서

- 이론: 문법 기초
  - 객체 소멸
  - C++ Exception 처리

- 실습:
  - Matrix 클래스에 객체 개수 count하기
  - Matrix 클래스에 exception 적용하기
  - Matrix 클래스의 함수들 이해하기

Main1 : 객체 소멸

```
int main1(int argc, char *argv[]) { // what about stack allocations?
  Matrix *m[1000] = { new Matrix(10,10), };
  for (int i=1; i<1000; i++) m[i] = new Matrix(10, 10);
  for (int i=0; i<999; i++) delete m[i];
  cout << "nAlloc=" << m[1000]->get_nAlloc() << endl;
  cout << "nFree=" << m[1000]->get_nFree() << endl;
  return 0;
}
class Matrix {
private:
  static int nAlloc;
  static int nFree;
  void alloc(int cy, int cx); // nAlloc++
public:
  int get_nAlloc() const;
  int get_nFree() const;
  ~Matrix(); // nFree++
```

```
nAlloc=1000
nFree=999
```

Exception 처리

- Exception 처리의 필요성
  - 코드 상에서 에러가 발생한 즉시 사용자에게 에러 발생 지점과 에러 원인 메시지를 출력하여 디버깅을 도와주기 위함
- 동작 방식
  - 함수들의 예외(또는 에러) 조건의 처리를 도와주기 위해 exception이라는 메커니즘(또는 클래스)를 제공함
  - 한 함수에서 에러가 발생하면 그 지점에서 적절한 exception 객체를 생성하여 throw함
    - 해당 함수는 함수 이름 뒤에 "throws xxxException"이라는 문구를 반드시 포함해야 함 → 함수 signature에 포함됨
    - xxxException 이름은 그 함수 안에서 발생 가능한 모든 exception들을 cover할 수 있는 general exception class 이어야 함
  - Throw된 exception 객체는 발생 지점 이후에 처음으로 만나게 되는 try-catch 블록에서 처리됨
    - Catch 블록에서는 exception 객체의 타입을 식별하여 원하는 exception 객체만 catch할 수 있음
    - Throw된 exception 객체를 포함하는 superclass의 exception 타입을 명세하면 보다 많은 exception들을 하나의 catch 블록에서 처리할 수 있음

Main2 : 오류 처리

```
int main2(int argc, char *argv[]) {
  int arrayBlk[3][3] = { ... };
  try {
    Matrix *currBlk = new Matrix((int *) arrayBlk, 3, 3);
    Matrix *tempBlk = new Matrix(5,5);
    //Matrix *tempBlk = new Matrix(-1,-1); // falls into the second catch
    Matrix *tempBlk2;
    tempBlk2 = tempBlk->add(currBlk); // falls into the first catch
    delete tempBlk;
    tempBlk2->print();
  } catch(MismatchedMatrixException& e) {
    cout << "at first catch: " << e.getMessage() << endl;
  } catch(MatrixException& e) {
    cout << "at second catch: " << e.getMessage() << endl;
  }
  return 0;
}
```

Main2 : 오류 처리

```
class Matrix {
private:
  ...
  void alloc(int cy, int cx) throw(MatrixException);
public:
  Matrix(); // why no throw?
  ...
  Matrix *clip(int top, int left, int bottom, int right) throw(MatrixException);
  void paste(const Matrix *obj, int top, int left) throw(MatrixException);
  Matrix *add(const Matrix *obj) throw(MismatchedMatrixException);
  ...
}
void Matrix::alloc(int cy, int cx) throw(MatrixException) {
  if ((cy < 0) || (cx < 0)) throw MatrixException("wrong matrix size");
  // do allocation;
}
Matrix *Matrix::add(const Matrix *obj) throw(MismatchedMatrixException) {
  if ((dx != obj->dx) || (dy != obj->dy))
    throw MismatchedMatrixException("Mismatched Matrices");
  // do addition;
}
```

Main2 : 오류 처리

```
class MatrixException {
public:
  MatrixException(): message("Matrix Exception") { }
  MatrixException(string msg): message(msg) {}
  string getMessage() const { return message; }
protected:
  string message;
};

class MismatchedMatrixException : public MatrixException {
public:
  MismatchedMatrixException(): MatrixException("Mismatched Matrix
Exception") {}
  MismatchedMatrixException(string msg): MatrixException(msg) {}
};
```

Matrix 클래스 이해

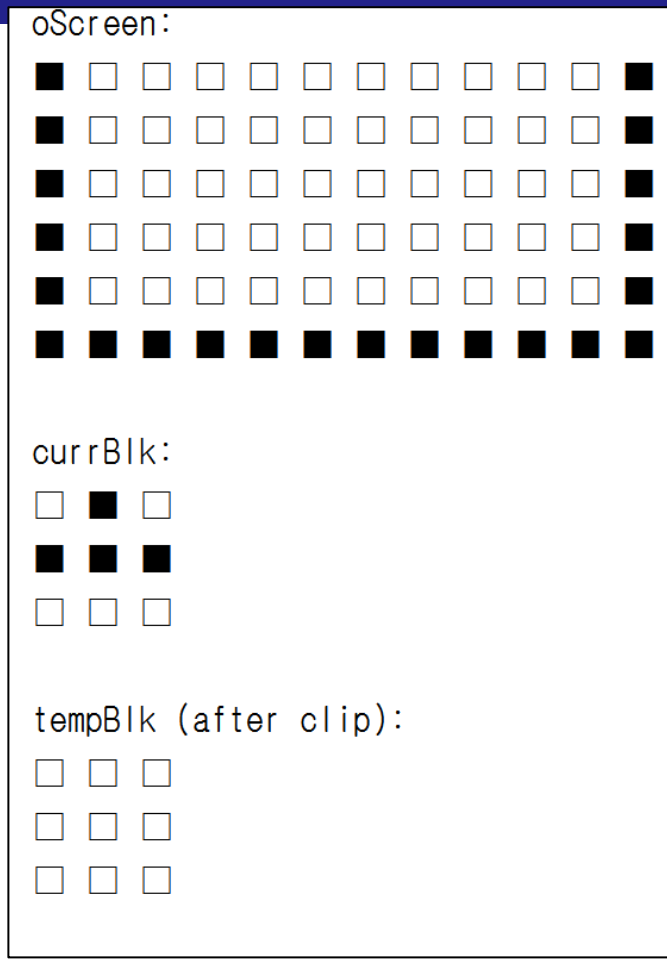
- Matrix class의 함수들
  - M.clip(top, left, bottom, right) : (top, left, bottom, right)로 정의되는 사각형에 포함되는 원소들을 또 다른 행렬로 리턴한다.
  - M.add(obj): 행렬 M와 행렬 obj를 원소 대 원소로 더한 결과를 또 다른 행렬로 리턴한다.
  - M.paste(object, top, left): 행렬 M의 (top, left) 좌표를 좌상 꼭지점으로 삼아 행렬 object를 행렬 M 안에 복사한다.
  - M.sum(): 행렬 M의 모든 원소들의 합을 리턴한다.
  - M.mulc(coef): 행렬 M의 모든 원소에 coef를 곱한다.
  - M.anyGreaterThan(val): 행렬 M 안에 val 값보다 큰 값을 갖는 원소가 하나라도 있으면 true를 리턴한다.

Main3 : Matrix functions

```
int main3(int argc, char *argv[]) {
  int arrayScreen[6][12] = { ... };
  int arrayBlk[3][3] = { ... };
  Matrix *oScreen = new Matrix((int *)arrayScreen, 6, 12);
  cout << "oScreen:" << endl;
  drawMatrix(oScreen); cout << endl;

  Matrix *currBlk = new Matrix((int *) arrayBlk, 3, 3);
  cout << "currBlk:" << endl;
  drawMatrix(currBlk); cout << endl;

  int top = 0;
  int left = 4;
  Matrix *tempBlk = oScreen->clip(top, left,
            top+currBlk->get_dy(), left+currBlk->get_dx());
  cout << "tempBlk (after clip):" << endl;
  drawMatrix(tempBlk); cout << endl;
```



Main3 : Matrix functions

```
Matrix *tempBlk2 = tempBlk->add(currBlk); delete tempBlk;
cout << "tempBlk (after add):" << endl;
drawMatrix(tempBlk2); cout << endl;

oScreen->paste(tempBlk2, top, left);
cout << "oScreen (after paste):" << endl;
drawMatrix(oScreen); cout << endl;

cout << "currBlk->sum()=" << currBlk->sum() << endl;
cout << endl;

tempBlk2->mulc(2);
cout << "tempBlk (after mulc):" << endl;
tempBlk2->print(); cout << endl;

cout << "currBlk->anyGreaterThan(1)=" << currBlk->anyGreaterThan(1) << endl;
cout << "tempBlk->anyGreaterThan(1)=" << tempBlk->anyGreaterThan(1) << endl;
}
```

```
tempBlk (after add):
□ ■ □
■ ■ ■
□ □ □

oScreen (after paste):
■ □ □ □ □ ■ □ □ □ □ □ ■
■ □ □ □ ■ ■ ■ □ □ □ □ ■
■ □ □ □ □ □ □ □ □ □ □ ■
■ □ □ □ □ □ □ □ □ □ □ ■
■ □ □ □ □ □ □ □ □ □ □ ■
■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

currBlk.sum()=4

tempBlk (after mulc):
Matrix(3,3)
0 2 0
2 2 2
0 0 0

currBlk.anyGreaterThan(1)=false
tempBlk.anyGreaterThan(1)=true
```

Main3 : Matrix functions

```
void drawMatrix(Matrix *m) {
  int dy = m->get_dy();
  int dx = m->get_dx();
  int **array = m->get_array();
  for (int y=0; y < dy; y++) {
    for (int x=0; x < dx; x++) {
      if (array[y][x] == 0) cout << "□ ";
      else if (array[y][x] == 1) cout << "■ ";
      else cout << "X ";
    }
    cout << endl;
  }
}
```