C++ 맛보기 예제들

숭실대학교
김강희 교수
(khkim@ssu.ac.kr)

순서

- 이론: 객체지향 언어의 문법 기초
  - C++과 Java의 차이점
  - 객체와 클래스
  - 객체 생성
  - 접근 제한자 (public/private/protected)
  - static & const

- 실습:
  - 문자열 조작
  - 문자열과 숫자값 사이의 변환
  - 배열
  - static 선언과 내부 클래스
  - 클래스 상속

용어 사용

- 데이터 필드(data field) ↔ 멤버 변수(member variable)
- 메쏘드(method) ↔ 멤버 함수(member function)

C++과 Java의 차이점 (by Dr. Veerasamy)

C++	Java
Write once, compile everywhere → 타겟마다 별도의 실행 파일을 생성함	**Write once, run anyware → 동일한 실행 파일(class file)이 타겟에 의존적인 JVM 위에서 실행함**
클래스 이름과 파일 이름이 반드시 일치할 필요가 없음 (.h 파일과 .cpp 파일을 따로 코딩함)	클래스 이름(public class)과 파일 이름이 반드시 일치해야 함
기본 데이터 타입과 객체 타입 모두 stack과 heap에 할당 가능함. 배열 할당도 동일한 규칙이 적용됨	기본 데이터 타입은 stack과 heap에 할당 가능하지만 객체 타입은 오직 heap에 할당해야 함. 배열 할당에도 동일한 규칙이 적용됨
포인터, 레퍼런스, pass by value 가 지원됨. 배열 경계 체크가 없음	기본 데이터 타입은 항상 pass by value 이고, 객체 타입은 항상 pass by reference임. 배열 경계 체크가 있음
소멸자(destructor) 지원하여 메모리 관리를 프로그래머가 직접 수행함	소멸자 대신에 자동적인 garbage collection을 지원함
연산자 오버로딩을 지원함	연산자 오버로딩을 지원하지 않음
동적 클래스를 지원하지 않음	동적 클래스를 지원함
함수 호출시 정적 바인딩을 사용함	함수 호출시 동적 바인딩을 사용함

C++과 Java의 차이점: 소스 파일 관리

❖ (Matrix.h, Matrix.cpp) vs. Matrix.java

```cpp
#pragma once
#include <iostream>
#include <cstdlib>

using namespace std;

class Matrix {
private:
  int dy;
  int dx;
  int **array;
  void alloc(int cy, int cx);
public:
  int get_dy();
  int get_dx();
  int** get_array();
  Matrix();
  Matrix(int cy, int cx);
  Matrix(const Matrix *obj);
  Matrix(const Matrix &obj);
  Matrix(int *arr, int col, int row);
  ~Matrix();
  Matrix *clip(int top, int left, int bottom, int right);
  void paste(const Matrix *obj, int top, int left);
  Matrix *add(const Matrix *obj);
  int sum();
  void mulc(int coef);
  Matrix *int2bool();
  bool anyGreaterThan(int val);
  void print();
  friend ostream& operator<<(ostream& out, const Matrix& obj);
  Matrix& operator=(const Matrix& obj);
};
```

```java
package ssu.rubicom.tetrismodel;

public class Matrix {
    private int dy = 0;
    private int dx = 0;
    private int[][] array = null;
    public int get_dy() { return dy; }
    public int get_dx() { return dx; }
    public int[][] get_array() { return array; }
    private void alloc(int cy, int cx) throws MatrixException {...
    public Matrix() throws MatrixException {  alloc( cy: 0,  cx: 0);
    public Matrix(int cy, int cx) throws MatrixException {...}
    public Matrix(Matrix obj) throws MatrixException {...}
    public Matrix(int[][] a) throws MatrixException {...}
    public Matrix clip(int top, int left, int bottom, int right) t
    public void paste(Matrix obj, int top, int left) throws Matrix
    public Matrix add(Matrix obj) throws MatrixException {...}
    public int sum(){...}
    public void mulc(int coef){...}
    Matrix int2bool() throws Exception {...}
    public boolean anyGreaterThan(int val){...}
    public void print(){...}
    // end of Matrix
}
class MatrixException extends Exception {...}
```

변수

- 객체지향 언어에는 두 가지 변수 타입이 존재함
  - 기본 (primitive) 타입

C++	설명
bool	true or false
char	8-bit 2's complement integer
short	16-bit 2's complement integer
int (long)	32-bit 2's complement integer
long long	64-bit 2's complement integer
float	32-bit IEEE 754 floating-point number
double	64-bit IEEE 754 floating-point number
wchat_t	16-bit or 32-bit Unicode

  - 객체 (object) 타입
    - Predefined object type : 라이브러리에서 제공하는 타입
    - User-defined object type : 프로그래머가 새로 정의한 타입

객체와 클래스

- 객체 (object) : 인스턴스(instance)라는 단어와 같은 의미
  - 정의 1: 클래스(class) 타입으로 선언된 변수
    - 참고 : '변수'는 컴퓨터 주메모리에 할당된 기억 공간으로서 크기와 타입을 가짐 (예: "int a"는 변수 a가 정수형 4바이트 공간임을 선언함)
    - 예1: Matrix A(3, 3); // stack 할당
    - 예2: Matrix *A = new Matrix(3, 3); // heap 할당
  - 정의 2: (할당된 메모리 공간을 분해하면) 기본 타입을 가진 변수들의 집합체

객체와 클래스

- 클래스 (class)
  - 객체를 구성하는 변수(data field)들과 이 변수들에 작용하는 절차(method)들을 프로그래밍 언어로 “추상적으로” 정의한 것
    - 각 변수는 또 다른 클래스 타입이거나 기본 타입(primitive type)임
    - 예 :

```
class Matrix {
    int dy; int dx;
    int **array;
    Matrix(int cy, int cx);
    Matrix *add(const Matrix *obj);
}
```

객체와 클래스

- 클래스 상속 (inheritance)
  - 새로운 클래스 C를 정의할 때 다른 클래스 P의 정의의 전부 또는 일부를 재사용하는 기법 (클래스 C에서는 새 변수 및 메쏘드를 추가, 또는 P의 메쏘드를 교체할 수 있음)
  - 클래스 C를 child, derivate, subclass, subtype 등으로 지칭하고 클래스 P를 parent, base, superclass, supertype 등으로 지칭
    - 예 :

```
class MyMatrix : public Matrix {
public:
  MyMatrix() : Matrix() { ... }
  MyMatrix(int cy, int cx) : Matrix(cy, cx) { ... }
  void print() { ... }
};
```

객체 생성

- 객체 생성 방법 (stack)
  - Matrix m(3,3);
    - stack 영역에 객체를 할당하고, 생성자 함수를 호출함
    - 그 결과 dy = 3, dx = 3 이라는 값을 가지게 됨
    - 프로그램 흐름이 객체가 최초 생성된 scope 을 벗어날 때 객체는 자동적으로 할당이 해제됨 (메모리가 반환됨)
- 객체 생성 방법 (heap)
  - Matrix *m = new Matrix(3, 3);
    - 좌변은 객체의 주소를 저장하는 포인터 변수를 정의함
    - 우변은 객체를 heap 영역에 할당하고, 생성자 함수를 호출함
    - 그 결과 dy = 3, dx = 3 이라는 값을 가지게 됨
    - 프로그램 흐름이 객체가 최초 생성된 scope 을 벗어나도 객체는 할당 해제되지 않음 (메모리가 반환되지 않음)
    - 할당 해제를 위해서는 다음 코드를 명시적으로 호출해야 함
      - delete m;

객체 생성

- 질문 1: 다음 두 객체 선언은 서로 다른 객체를 생성하는가?
  - ❖Matrix *m1 = new Matrix(3, 3);
  - ❖Matrix *m2 = m1;

  → 동일한 객체를 가리킨다. 즉 동일한 메모리 공간을 두 개의 포인터 변수 m1과 m2가 동시에 가리킨다.
- 질문 2: m2 객체가 m1 객체의 내용과 동일하되 독립된 객체로서 선언하고자 하면, 어떻게 생성해야 하는가?

  → 다음과 같이 복사 생성자를 사용해야 함
  - ❖Matrix *m2 = new Matrix(m1);

생성자 작성법

- 생성자 작성시 주의할 점
  - 주의사항1: 리턴형이 없어야 함
  - 주의사항2: 반드시 public으로 선언해야 함

```
class Matrix {
private:
  int dy;  int dx;
  int **array;
public:
  Matrix(); // 디폴트 생성자
  Matrix(int cy, int cx); // 일반 생성자
  Matrix(const Matrix *obj); // 복사 생성자1
  Matrix(const Matrix &obj); // 복사 생성자2
}
```

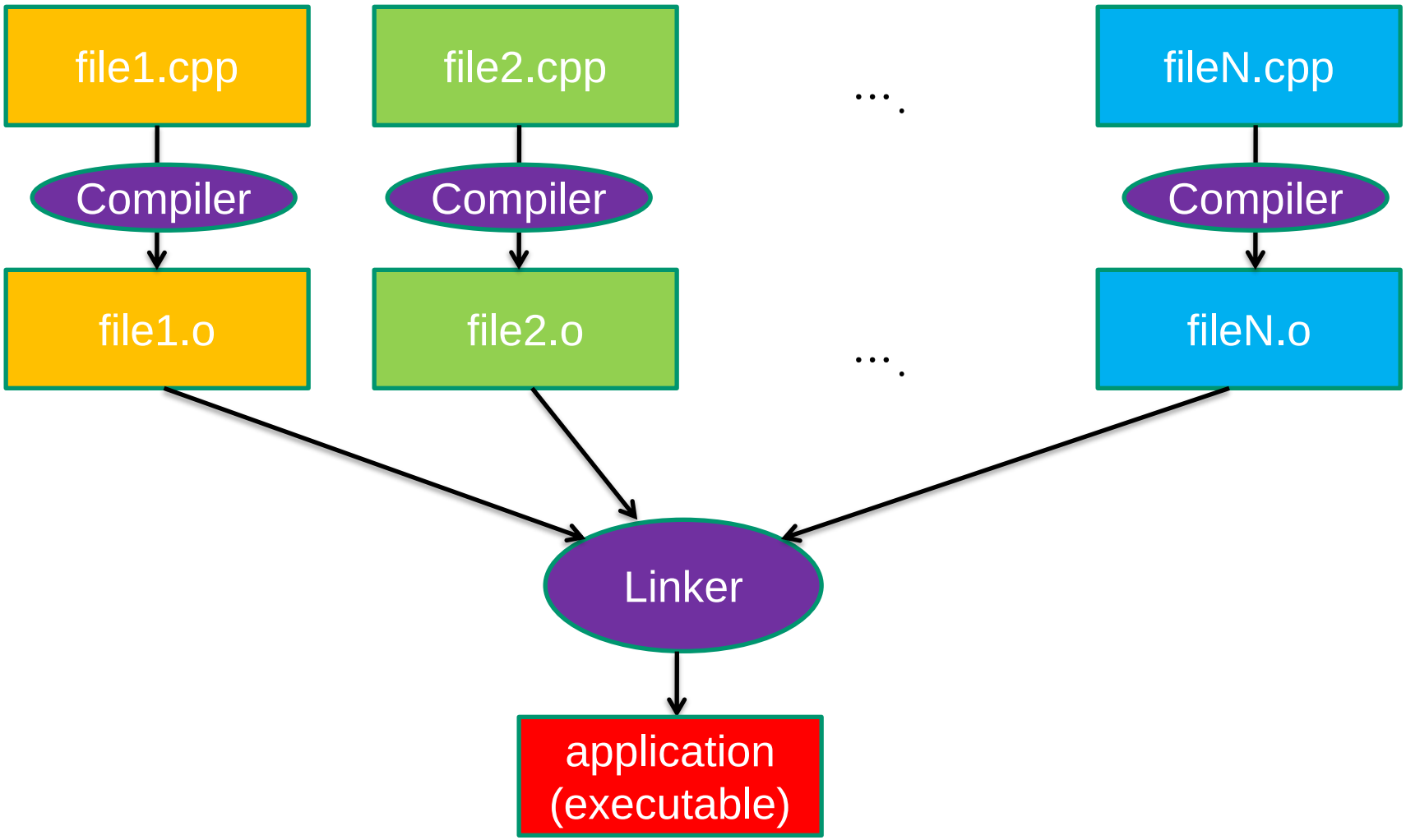
생성자 작성법

- 생성자 작성시 주의할 점
  - 주의사항3: 네트워크 초기화 또는 데이터베이스 초기화와 같은 복잡한 동작을 하지 말아야 함
  - 주의사항4: 서브클래스의 생성자는 수퍼클래스의 생성자를 반드시 호출함

```
class MyMatrix : public Matrix {
public:
  MyMatrix() : Matrix() { ... }
  MyMatrix(int cy, int cx) : Matrix(cy, cx) { ... }
  void print() { ... }
};
```

Lab1 예제들

C++ compiler & linker usage



Build & Run

1. $ mkdir khkim # khkim 폴더 생성하기
2. $ mv ~/Downloads/cpphello.zip khkim # cpphello.zip 파일을 khkim 폴더 내부로 이동시키기
3. $ cd khkim # khkim 폴더 진입하기
4. $ unzip cpphello.zip # 압축 패키지 풀기
5. $ cd cpphello # 폴더 진입하기
6. $ ls –al # 파일 목록 조회하기
7. $ make # Makefile 참조하여 실행 파일 생성하기
8. $ ls –al # 파일 목록 조회하기
9. $ ./Lab1.exe 1 # Lab1.exe 를 인자 1 과 함께 실행하기
10. $ ./Lab1.exe 2 # Lab1.exe 를 인자 2 와 함께 실행하기
11. …
12. $ ./Lab1.exe 5 # Lab1.exe 를 인자 5 와 함께 실행하기
13. $ make clean # 오브젝트 파일들 및 실행 파일들 삭제하기
14. $ ls –al # 파일 목록 조회하기

Makefile

```
1.  # Set compiler to use
2.  CC=g++
3.  CFLAGS=-g -I. -fpermissive -Wno-deprecated
4.  LDFLAGS=
5.  DEBUG=0
6.  all:: Lab1.exe Lab2.exe
7.  Lab1.exe: Lab1.o Matrix.o
8.      $(CC) -o $@ $^ -std=c++11 $(CFLAGS) $(LDFLAGS)
9.  Lab2.exe: Lab2.o Matrix_e.o
10.     $(CC) -o $@ $^ -std=c++11 $(CFLAGS) $(LDFLAGS)
11. %.o: %.c $(DEPS_CHAT)
12.     $(CC) -c -o $@ $< $(CFLAGS)
13. %.o: %.cpp $(DEPS_TET)
14.     $(CC) -c -o $@ $< $(CFLAGS)
15. clean:
16.     rm -f *.exe      rm -f  *.o  *~  *.stackdump
```

Main1 : string 조작

```
int main1(int argc, char *argv[]) {
  string s1 = "Hello, CPP!";
  cout << "s1=" << s1 << endl;
  string s0 = "Hello";
  string s2 = s0 + ", " + “CPP" + '!';
  cout << "s2=" << s2 << endl;
  int len1 = s1.length();
  int len2 = s2.length();
  cout << "len1=" << len1 << endl;
  cout << "len2=" << len2 << endl;
  int b1 = s1.compare(s1); // compare s1 and s2 (in content)
  int b2 = s1.compare(s2); // compare s1 and s2 (in content)
  bool b3 = (s1 == s2); // compare s1 and s2 (in content)
  cout << "s1.compare(s1)=" << b1 << endl;
  cout << "s1.compare(s2)=" << b2 << endl;
  cout << "(s1==s2)=" << b3 << endl;
  return 0;
}
```

```
s1=Hello, C++!
s2=Hello, C++!
len1=11
len2=11
s1.compare(s1)=0
s1.compare(s2)=0
(s1==s2)=1
```

Main2 : string ↔ number 변환

```
int main2(int argc, char *argv[]) {
  string istr = "1234";
  string dstr = "12.34";
  int ival = strtol(istr.c_str(), NULL, 10); // string -> integer value
  double dval = strtod(dstr.c_str(), NULL); // string -> double value
  cout << "before : " << ival << ", " << dval << endl;
  ival = ival + 1111;
  dval = dval + 11.11;
  string istr2 = to_string(ival); // integer value -> string
  string dstr2 = to_string(dval); // double value -> string
  cout << "after  : " << istr2 << ", " << dstr2 << endl;
  return 0;
}
```

```
before : 1234, 12.34
after  : 2345, 23.45
```

Main3 : 배열 조작

```
#include <vector>
#include <iterator>
int main3(int argc, char *argv[]) {
  int* A0 = NULL;  //int A0[] = NULL;
  int A1[] = {1, 2, 3, 4, 5};
  int* A2 = new int[5]; //int[] A2 = new int[5];
  int* A3 = new int[5]{1, 2, 3, 4, 5}; //int A3[] = new int[]{1, 2, 3, 4, 5};

  cout << "A0:";  printArray(A0, 0);
  cout << "A1:";  printArray(A1, 5);
  cout << "A2:";  printArray(A2, 5);
  cout << "A3:";  printArray(A3, 5);
  vector<int> v1(A1, A1 + 5);
  vector<int> v3(A3, A3 + 5);
  cout << "equal(v1, v3)=" << equal(v1.begin(), v1.end(), v3.begin()) << endl;
  cout << "(v1 == v3)=" << (v1 == v3) << endl;
  cout <<  "v1[0] = " << v1[0] << endl;
  return 0;
}
```

```
A0:
A1:1 2 3 4 5
A2:0 0 0 0 0
A3:1 2 3 4 5
equal(v1, v3)=1
```

```
void printArray(int a[], int len) {
  if (a != NULL) {
    for (int i = 0; i < len; i++)
      cout << a[i] << " ";
  }
  cout << endl;
}
```

Main4 : static/dynamic, nested class

```
int main4(int argc, char *argv[]) {
  Nested* m1 = new Nested(1, 2);
  Nested* m2 = new Nested(3, 4);
  cout << "m1->get_dy()=" << m1->get_dy() << ", m1->get_dx()=" << m1->get_dx() << endl;
  cout << "m2->get_dy()=" << m2->get_dy() << ", m2->get_dx()=" << m2->get_dx() << endl;
  Nested::InnerS *s = new Nested::InnerS();
  cout << "s->get_dx()=" << s->get_dx() << endl;
  return 0;
}
```

```
m1->get_dy()=1, m1->get_dx()=4
m2->get_dy()=3, m2->get_dx()=4
s->get_dx()=4
```

Main4 : static/dynamic, nested class

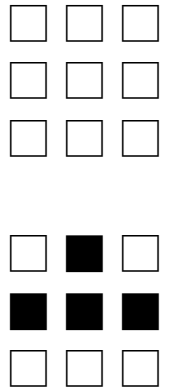
```
class Nested {
public: // private:
  int dy; // dynamic variable
  static int dx; // static variable
public:
  Nested(int cy, int cx) { dy = cy; dx = cx; }
  int get_dy() { return dy; }
  static int get_dx() { return dx; } // can be declared 'dynamic'
  static class InnerS {  // An inner class inherently is a static class
                         // even without 'static' keyword
        public:
        int get_dx() { return dx; }
  };
};
int Nested::dx = 0;
```

Main5 : 상속 & 다형성

```
int main5(int argc, char *argv[]) {
  Matrix* m1 = new Matrix(3,3);
  m1->print(); cout << endl;
  int A[] = { 0, 1, 0, 1, 1, 1, 0, 0, 0 }; // int A[][]
  Matrix* m2 = new Matrix(A, 3, 3); // new Matrix(A)
  m2->print(); cout << endl;
  MyMatrix* m3 = new MyMatrix(3,3);
  m3->print(); cout << endl;
  MyMatrix* m4 = new MyMatrix(A, 3, 3); // new Matrix(A)
  m4->print(); cout << endl;
  m2 = m4; // polymorphism: Matrix covers MyMatrix!!
  m2->print(); cout << endl; // static binding!!
  return 0;
}
```

```
Matrix(3,3)
0 0 0
0 0 0
0 0 0

Matrix(3,3)
0 1 0
1 1 1
0 0 0
```



```
Matrix(3,3)
0 1 0
1 1 1
0 0 0
```

Main5 : 상속 & 다형성

```
#include "Matrix.h"
class MyMatrix : public Matrix {
public:
  MyMatrix() : Matrix() { }
  MyMatrix(int cy, int cx) : Matrix(cy, cx) { }
  MyMatrix(int *arr, int col, int row) : Matrix(arr, col, row) { }
  void print() {
    int dy = get_dy();
    int dx = get_dx();
    int **array = 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;
    }
  }
```

static 선언

- Static 선언이 없으면, (자동으로) dynamic으로 간주함
- 변수 앞에 static 선언이 붙으면,
  - 그 변수가 속한 클래스의 인스턴스들 사이에서 공유되는 변수 (개별 객체 수준에서 static 변수를 위한 공간 할당이 없음)
  - 그 변수는 클래스에 소속된다고 말하며, 객체에 소속되지 않음
- 메쏘드 정의 앞에 static 선언이 붙으면,
  - Static 메쏘드는 서브클래스에서 재정의(override)할 수 없음
  - 그 메쏘드는 클래스에 소속된다고 말하며, (객체의) dynamic 변수들을 접근할 수 없고, 오직 그 클래스의 static 변수들만 접근할 수 있음
- 클래스 정의 앞에는 static 선언이 없어도 static 으로 선언된 효과가 있음!

접근 제한자

- private modifier:
  - 필드 또는 메쏘드 앞에 붙어서, 소속 클래스 외부에서 해당 필드 또는 메쏘드를 참조할 수 없게 한다.
- protected modifier:
  - 필드 또는 메쏘드 앞에 붙어서, 서브클래스가 해당 필드 또는 메쏘드를 참조하는 것을 허용한다.
- public modifier:
  - 필드 또는 메쏘드 앞에 붙어서, 소속 클래스 외부 어디에서든지 해당 필드 또는 메쏘드를 참조하는 것을 허용한다.

const 선언

- 변수(필드,형식인자,지역변수) 앞에 const 선언되면,
  - 그 변수는 (선언과 동시에 또는 생성자 안에서) 한번 값이 정해지면 변경될 수 없음
  - 그 변수는 그 변수를 감싸는 블록 안에서 사용 전에 오직 1회만 초기화 가능함
- 형식인자(parameter) 앞에 const 선언되면,
  - 실제인자의 값은 해당 메쏘드 안에서 변경될 수 없음
  - 다만, 실제인자가 객체이면 객체 내부의 필드들은 변경될 수 있음

결론 (by Dr. Veerasamy)

- C++은 controls/options이 많은 비행기를 운전하는 것이고 Java는 controls/options이 적은 자동차를 운전하는 것이다.
- C++은 잘 사용하면 좋은 성능을 가져오고, 잘못 사용하면 심각한 성능 저하를 가져온다.
- 메모리 누수(memory leak), 배열 경계 초과 접근(out-of-bounds array access) 등은 memory corruption을 가져옴 (이 이슈들은 일반적으로 너무 늦게 발견된다는 것이 문제임)
- native program 를 작성할 때 객체 지향 언어가 필요하다면 반드시 C++ 언어가 필요함

부록

- Linux 의 파일 조작 명령어들
  - cp [source files] [target directory]
  - cp [source file] [target file]
  - mv [source files] [target directory]
  - mv [source file] [target file]
  - rm [files]
  - rm –r [directory]
  - zip –r [target zipped file] [source directory]
  - unzip [source zipped file]
- 경로 표기시 유의할 사항들
  - '~' : home directory 를 의미함
  - '.' : current directory 를 의미함
  - '..' : parent directory 를 의미함
  - 예: "mv ../cpphello.zip . " → parent directory 에 있는 cpphello.zip 을 current directory 로 복사함

부록

- Ubuntu 의 패키지 설치 명령어들
  - sudo apt update → 패키지 목록을 갱신함 (install 전에 실행할 것)
  - sudo apt install [package names]
- Ubuntu 에 vscode 설치하는 방법
  - https://code.visualstudio.com/docs/setup/linux 참조할 것