**프로그래밍언어론 Report**

201524404 강민진

**- Pointer를 잘못 사용하면 생길 수 있는 오류와 이를 해결하는 각 프로그래밍 언어가 취하는 방법을 설명하라**

**Dangling Pointer.** 어떤 것을 가리키고 있는데 쓰레기값을 가리키거나 아무것도 가리키지 않는 포인터이다.(해제된 메모리 영역을 가리키고 있는 것)

**초기화 시.**

int \*a;

\*a = 3; // ERROR!

C에서 포인터를 선언 초기화하지 않고 사용하면 프로세서가 할당한 메모리 범위를 벗어난 공간 혹은 침범하면 안되는 곳의 메모리의 주소값을 가질 수 있어 심각한 오류를 초래할 수 있다.

해결 : C++ 에서 이 문제를 해결하기 위해서 레퍼런스 (&)를 도입하여서 이러한 문제가 생기는 것을 방지하였다.(레퍼런스(&)은 선언과 동시에 초기화를 시켜줘야 하고, 수정이 불가능하다.) 또한 최신의 컴파일러들은 초기화하지 않은 변수의 사용을 런타임에 체크해주어 이러한 문제가 일어나는 것을 방지한다.

**원인**

1) 동적으로 할당받은 메모리가 해제되었는데도 포인터 변수가 이 해제된 메모리 주소를 가리키고 있는 경우

int \*p1, \*p2;

p1 = (int \*)malloc(sizeof(int));

p2 = p1;

free(p1);

여기서 끝나게 되면 p2는 사라진 p1의 주소를 가리키게 되므로 dangling pointer 문제가 발생하게 된다.

2) 지역변수의 주소를 함수에서 리턴할 경우

int function(){

int a = 1;

return &a; // 함수가 사라지면 지역변수는 사라지기 때문에 리턴받은 값은 Dangling.

}

앨리어싱

하나 이상의 포인터가 같은 메모리 영역을 가리키고, 그 중 하나가 해제된 경우

int \*p1 = (int \*)malloc(sizeof(int));

\*p1 = 5;

int \*p2;

p2 = p1;

free(p1);

\*p2 = 10; // p2는 Dangling Pointer.

이러한 Dangling Pointer는

1) 메모리 접근 시 접근이 되었을때(원치않는 곳을 접근하는 것이므로) 예측 불가능한 동작이 발생할 수 있고.

2) 메모리 접근 불가시 Segment Fault 발생할 수 있고

3) 잠재적인 보안 위험을 초래한다.

Dangling Pointer 해결 방안 :

JAVA : JAVA에서는 메모리 할당, 해제를 명시적으로 하는 매커니즘이 없어서 Dangling pointer가 발생하지 않는다. 대신 Garbage Collector가 객체가 어떤 참조에도 닿지 않을때 메모리를 해제한다.

Python, PHP, C#은 JAVA와 같이 Garbage Collector을 통해 메모리 관리를 자동으로 수행해준다.

(but, Garbage Collector는 속도가 느려지게 하는 주 원인이다. 메모리를 정확히 언제 반환해야 하는지 모르기 때문에 주기적으로 객체가 필요한지 아닌지 확인하고, 포인터가 생성되고 제거될때마다 참조 횟수를 확인하며 이 두 방법은 CPU와 메모리를 사용하기 때문에 프로그램 수행속도가 느려진다. 따라서 속도가 중요한 C++, Rust와 같은 언어에는 Garbage Collector가 지원된다고 하더라도 적합하지 않을 것이다.)

Pascal : 변수의 주소를 포인터에 대입하는 자체를 금지시킨다.

ada : 명시적 deallocation을 지원한다.

Rust : 성능과 메모리 안정성을 모두 달성하기 위해 소유권이라는 개념을 도입. 소유자가 범위를 벗어나면 값이 삭제된다. (변수가 범위를 벗어나면 자동으로 해당 메모리를 해제한다)

C/C++ 자체적으로는

1) 메모리 해제 후 포인터를 NULL로 설정.

2) free() 함수를 대체할 적절한 함수를 만들어 사용

예를 들면 free()후에 포인터에 NULL을 넣어주는 함수.

3) 런타임 시스템, 디버깅 시스템이 이러한 제어가 가능하도록 하는 방법 등이 있다.

**- C언어에서 Complie time 에 Type checking 이 불가능한 예를 보여라**

**union**

Union이 C언어에서 타입체킹을 불가능하게 하는 요소이다. Union은 Union안에 속한 변수 타입크기의 최대의 크기로 메모리를 할당받고 그 메모리를 공유한다. 그래서 Union의 각 멤버에 접근할 때 같은 주소값을 가리키지만, 선언된 타입에 따라 메모리 형식을 다르게 결정한다.

이로써 Compile time 에 type checking 을 하지 않음을 알 수 있다.

**void pointer**

int형, char 형 포인터 자리에 void 형 포인터를 써도 에러가 발생하지 않는다. 또한 void pointer의 type은 runtime 시에 checking 되므로 Compile time에는 checking 이 불가능하다. 포인터의 형이 유동적으로 변할 수 있을 때 사용되도록 만들었다.

**Enum(열거형)**

enum color{red = 1, blue, green}; 이라고 설정하면 원래는 0부터 1, 2 이렇게 증가하지만 런타임에 red부터 1, 2, 3 이렇게 증가되어 나타난다. 이는 Compile Time 에 checking 이 불가능하다.

**자동 형변환**

int a;

double b = 1.5

a = b;

이런 식으로 일반적인 변수의 연산결과를 compile time 에는 checking 할 수 없다.