FBD로부터 C프로그램 코딩 가이드라인을 준수하는 C코드 변환규칙

C code conversion rules complying with C program coding guidelines from FBD

송승현*, 이동아, 유준범

Dependable software laboratory
건국대학교 컴퓨터공학과

union93@konkuk.ac.kr, ldalove@konkuk.ac.kr, jbyoo@konkuk.ac.kr

목차

1. 서론
2. 배경지식
3. FBD에서 C코드로의 변환규칙 개선
   - 3-1 기존 자동변환의 C코딩 가이드라인 위반 사항 분석
   - 3-2 가이드라인을 준수하도록 변환규칙 개선
4. 사례연구
5. 결론

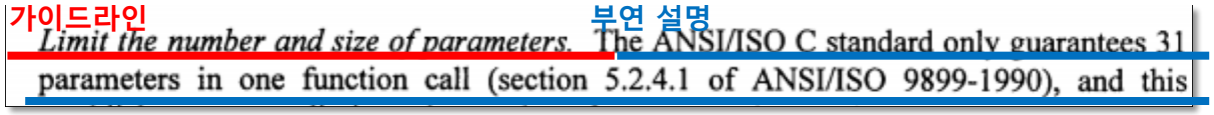
서론

- 원자력 발전소와 같은 안전필수 시스템에서는 안전 최우선으로 시스템을 구성함
- Function Block Diagram을 안전 시스템 소프트웨어 디자인에 사용하도록 권장하고 있음
- 기존 연구에서 FBD를 소프트웨어 개발을 위해 C언어 코드로 자동 변환하는 도구를 제작하였음
- 그러나 자동 변환도구(FBDtoC)의 결과물이 C언어 코딩 가이드라인을 준수하는지 확인 할 수 없었음

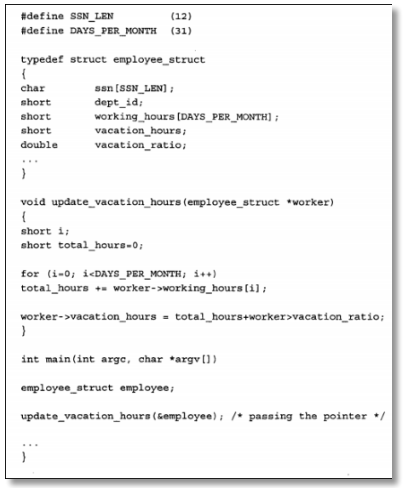
배경지식

NUREG/CR-6463

- 원자력 시스템의 구현에 사용되는 프로그램 언어에 대한 가이드라인으로 각 언어별 코딩 가이드라인과 해당 가이드라인을 준수하지 않을 시 발생하는 이슈를 기술하고 있다. 다음과 같이 가이드라인과 샘플이 포함 되어있다.



Ex)가이드라인: 매개변수의 숫자를 제한 할 것



```
#define SSN_LEN         (12)
#define DAYS_PER_MONTH  (31)

typedef struct employee_struct
{
char        ssn[SSN_LEN];
short       dept_id;
short       working_hours[DAYS_PER_MONTH];
short       vacation_hours;
double      vacation_ratio;
...
}

void update_vacation_hours(employee_struct *worker)
{
short i;
short total_hours=0;

for (i=0; i<DAYS_PER_MONTH; i++)
total_hours += worker->working_hours[i];

worker->vacation_hours = total_hours+worker>vacation_ratio;
}

int main(int argc, char *argv[])

employee_struct employee;

update_vacation_hours(&employee); /* passing the pointer */

...
}
```

Ex)매개변수 숫자제한에 관한 샘플

각 챕터는 다음과 같이 구성되어 있다

1) Reliability: 코드의 신뢰성
2) Robustness: 코드의 내구성
3) Traceability: 코드의 추적성
4) Maintainability: 코드의 유지 보수성

이번 연구에서는 C코드 작성과 직접 연관성이 높은 코드 Reliability와 Maintainability 챕터에 대한 가이드라인을 중점적으로 분석하였음

FBD를 C코드로 변환하는 규칙 개선

기존 자동변환의 C코딩 가이드라인 위반 사항 분석

기존의 FBDtoC의 FBD 프로그램 자동변환 결과물을 *실행/스타일/외부* 방식으로 구분하여 위반여부를 판단하였다.

실행: 결과물인 C코드를 가이드라인이 제시하는 환경,조건에서 실행하여 컴파일 혹은 실행에 문제가 있는 경우 *실행-가이드라인* 위반으로 판단

스타일: 결과물인 C코드가 가이드에서 제시하는 코딩 스타일을 준수하지 않을 경우 *스타일-가이드라인* 위반으로 판단

외부: 가이드라인이 결과물인 C코드 외적인 버전관리도구 혹은 문서 등에 관하여 가이드라인을 제시하고 있으나 지켜지지 않은 경우 *외부-가이드라인* 위반으로 판단

기존 자동변환의 C코딩 가이드라인 위반 사항 분석

가이드 번호	내용	위반 유형
4.1.1.3	루틴을 지나는 매개변수 숫자를 보장가능한 수준으로 제어할 것	실행
4.1.2.6	데이터 타입을 값의 범위에 따라 제한하여 사용할 것	실행
4.1.2.7	부동소수점 정확도가 높은 데이터 타입을 사용할 것	실행
4.2.2.1	예외를 해당 흐름내에서 처리할 것	실행
4.2.2.2	외부 제어흐름을 보존하는 코딩을 할 것	실행
4.3.3	버전 관리자 툴의 사용할 것	외부
4.4.1.2	식별자 이름 설명을 명확하게 할 것	스타일
4.4.1.3	주석 및 내부 문서를 보강할 것	스타일
4.4.1.6	모호하거나 미묘한 프로그래밍 구조를 회피 할 것	스타일
4.4.1.8	리터럴 사용을 최소화 할 것	스타일
4.4.5.1	플랫폼 기반의 언어 사용자제	스타일

가이드라인을 준수하도록 변환규칙 개선

실행: 변환 결과물이 가이드라인에서 제시하는 환경, 조건에서 컴파일, 실행이 원활하게 되도록 변환 규칙을 개선

스타일: 변환 결과물이 가이드라인에서 제시하는 코딩 스타일을 따르도록 변환 규칙을 개선

외부: 가이드라인에서 요구하는 외부 사항을 충족되도록 변환 규칙을 개선

가이드라인을 준수하도록 변환규칙 개선

가이드 번호	내용	위반 유형	개선 사항
4.1.1.3	루틴을 지나는 매개변수 숫자를 보장가능한 수준으로 제어할 것	실행	매개변수를 31개로 제한
4.1.2.6	데이터 타입을 값의 범위에 따라 제한하여 사용할 것	실행	Int 타입 사용을 제한, short long과 같은 데이터 타입 사용
4.1.2.7	부동소수점 정확도가 높은 데이터 타입을 사용할 것	실행	Float 대신 Double 자료형 사용
4.2.2.1	예외를 해당 흐름내에서 처리할 것	실행	오버플로우, 데이터타입에 관한 예외 처리 추가
4.2.2.2	외부 제어흐름을 보존하는 코딩을 할 것	실행	외부 제어를 침해하지 않도록 플로우 설계
4.3.3	버전 관리자 툴의 사용할 것	외부	버전 관리도구 사용 지침 마련
4.4.1.2	식별자 이름 설명을 명확하게 할 것	스타일	차이점이 이름의 앞에 오도록 명명법 개선
4.4.1.3	주석 및 내부 문서를 보강할 것	스타일	주석 자동 생성 추가
4.4.1.6	모호하거나 미묘한 프로그래밍 구조를 회피 할 것	스타일	삼항연산자 형태의 조건문 if-else 문으로 대체
4.4.1.8	리터럴 사용을 최소화 할 것	스타일	리터럴의 값에 소괄호 추가
4.4.5.1	플랫폼 기반의 언어 사용자제	스타일	Int 타입 사용을 제한, short long과 같은 데이터 타입 사용

가이드라인을 준수하도록 변환규칙 개선

FBD 정수 자료형의 C 정수 자료형 변환 규칙 개선

FBD 데이터타입		C 데이터 타입
SINT (1 byte) INT (2 byte)	->	signed short (2 byte)
DINT (4 byte)	->	signed long (4 byte)
LINT (8 byte)	->	signed long long(8 byte)
USINT (1 byte) UINT (2 byte)	->	unsigned short (2 byte)
UDINT (4 byte)	->	unsigned long (4 byte)
ULINT (8 byte)	->	unsigned long long(8 byte)

비교 연산자 코딩 스타일의 개선

* CP : Comparison Operator(비교연산자)	
기존의 비교조건함수 작성 규칙	개선된 비교조건함수 작성 규칙

function(A,B){
return (A CP B) A:B ;
}
	function(A,B){
	if(A CP B){
	return A;
	}
	else{
	return B;
	}
	}

리터럴 작성 규칙 개선

기존의 리터럴 작성규칙	개선된 리터럴 작성규칙
#define value;	#define (value);

사례연구

사례연구

- RPS(Reactor Protection System)은 원자력 플랜트 보호 시스템으로 안전 요구사항이 강력하게 요구되는 시스템이다. FBD를 사용하여 RPS 시스템 소프트웨어를 디자인하도록 되어있다. 이번 연구에서는 RPS 시스템의 Fix_Rising, Fix_Falling, Variable_Rising, Variable_Falling FBD 프로그램을 개선된 변환규칙을 사용하여 변환 후 가이드라인을 준수하게 되었는지 확인하였다.

사례연구

정수형 데이터 타입의 개선 결과

<Variable_Rate_Rising> 정수형 데이터 타입 개선

개선 전	개선 후
int RNG_MAX = 29400; int RNG_MIN = 600;	unsigned short RNG_MAX = 29400; unsigned short RNG_MIN = 600;

리터럴 코드의 스타일 개선

<Header.h>에 정의된 리터럴 코드의 스타일 개선

개선 전	개선 후
#define bool int #define true 1 #define false 0	#define bool (int) #define true (1) #define false (0)

모호한 표현과 정수형 데이터 타입개선 결과

<Fix_Falling> 모호한 비교조건문 코드 스타일 개선 & 정수형 데이터 타입 개선

개선 전	개선 후

bool LT_INT_2(int IN0 , int IN1){
return (IN0 < IN1)? true:false;
}
	bool LT_INT_2(unsigned short IN0 , unsigned short IN1){
	if(IN1 > IN0){
	return true;
	}
	else{
	return false;
	}
	}

결론

- FBDtoC의 변환규칙을 개선함으로써 FBDtoC의 결과물 C코드가 NUREG/CR-6463 C언어 가이드라인을 준수하게 되었다. 따라서 FBD로 디자인된 안전시스템을 변환하였을 때 C코딩 가이드라인을 미준수 함으로써 발생하는 안전이슈를 방지할 수 있게되었다.
- 향후 연구에서는 개선된 변환규칙을 FBDtoC도구에 적용하여 FBD로부터 안전 가이드라인이 준수된 C코드를 변환하는 자동변환 도구를 개발할 것이다.