



The Norm

Version 3

Summary: 본 문서는 42에서 적용 가능한 표준(Norm)을 설명합니다. 프로그래밍 표준은 코드를 작성할 때에 따라야하는 규칙들을 정의합니다. Norm은 기본적으로 커먼 코어 내의 모든 C 프로젝트와 지정된 모든 프로젝트에 적용됩니다.

Contents

I	머리말	2
II	Why ?	3
III	The Norm	4
III.1	명명	4
III.2	서식	5
III.3	함수	7
III.4	자료형, 구조체, 열거형(enum)과 공용체(union)	8
III.5	헤더 - a.k.a 파일 포함하기	9
III.6	42 헤더 - a.k.a 멋지게 파일 시작하기	10
III.7	매크로와 전처리기	11
III.8	금지 사항!	12
III.9	주석	13
III.10	파일	14
III.11	Makefile	15

Chapter I

머리말

norminette은 파이썬으로 작성되었으며 오픈 소스입니다.
리포지터리는 다음 주소에서 확인할 수 있습니다. <https://github.com/42School/norminette>
풀 리퀘스트, 제안과 이슈를 환영합니다!

Chapter II

Why ?

Norm은 많은 교육학적 요구를 충족시키기 위해 신중하게 만들어졌습니다. 아래의 모든 선택에 대한 이유는 다음과 같습니다.

- **시퀀싱:** 코딩은 크고 복잡한 작업을 긴 일련의 기본 명령으로 나누는 것을 의미합니다. 이 모든 명령은 차례대로 실행될 것입니다. 소프트웨어를 만들기 시작하는 초보자는 프로젝트를 위해 모든 개별 명령과 정확한 실행 순서를 완전히 이해한 단순하고 명확한 아키텍처가 필요합니다. 동시에 여러 명령을 수행하는 암호 언어 구문은 분명히 혼란스러우며 코드의 동일한 부분에 혼합된 여러 작업을 처리하려는 함수는 오류의 원인입니다.
The Norm은 각 조각의 고유한 작업을 명확하게 이해하고 검증할 수 있고, 실행된 모든 명령의 순서가 의심의 여지가 없는 곳에 간단한 코드 조각을 만들 것을 요구합니다. 이것이 우리가 함수에서 최대 25줄을 요구하는 이유이고, 또한 `for`, `do .. while`, 또는 삼항 연산자를 금지하는 이유입니다.
- **룩 앤드 필:** 일반적인 동료 학습 과정과 동료 평가 중에 친구 및 동료와 교환하는 동안 코드를 해독하는 데 시간을 쓰지 않고 코드 조각의 논리에 대해 직접 이야기하고 싶어 합니다.
The Norm은 많은 곳에서 함수와 변수의 이름 지정, 들여쓰기, 중괄호 규칙, 템 및 띄어쓰기에 대한 지침을 제공하는 특정 룩 앤드 필을 사용하도록 요구합니다. 이것은 익숙해 보이는 다른 사람의 코드를 부드럽게 살펴볼 수 있으며 코드를 이해하기 전에 코드를 읽는 데 시간을 쓸 필요 없이 바로 요점을 파악할 수 있게 합니다. The Norm은 또한 트레이드마크입니다. 42 커뮤니티의 일원으로서, 노동시장에 있을 때 다른 42학생이나 졸업생들이 작성한 코드를 알아볼 수 있을 것입니다.
- **장기 비전:** 이해할 수 있는 코드를 작성하려는 노력은 그것을 유지하는 가장 좋은 방법입니다. 당신을 포함한 다른 누군가가 버그를 수정하거나 새로운 기능을 추가해야 할 때마다 이전에 올바른 방식으로 작업을 수행했다면 그것이 무엇을 하는지 알아내려고 귀중한 시간을 낭비할 필요가 없습니다. 이렇게 하면 시간이 오래 걸린다는 이유만으로 코드 조각들이 더 이상 유지되지 않는 상황을 피할 수 있고, 시장에서 성공적인 제품을 가지고 있는 것에 대해 말할 때 차이를 만들 수 있습니다. 그렇게 하는 법은 빨리 배울수록 좋습니다.
- **표준:** 당신은 어떤 규칙이나 모든 규칙이 임의적이라고 생각할 수도 있지만, 우리는 실제로 무엇을 어떻게 해야 할지 생각하고 읽었습니다. 우리는 왜 함수가 짧고 한 가지 일만 해야 하는지, 변수 이름이 의미가 있어야 하는지, 줄 너비가 80열을 넘지 않아야 하는지, 함수가 많은 매개 변수를 사용하지 않아야 하는지, 주석이 유용해야 하는 이유 등을 구글에서 검색하는 것을 강력히 권장합니다.

Chapter III

The Norm

III.1 명명

- 구조체의 이름은 `s_`로 시작해야만 합니다.
- typedef의 이름은 `t_`로 시작해야만 합니다.
- 공용체(union)의 이름은 `u_`로 시작해야만 합니다.
- 열거형(enum)의 이름은 `e_`로 시작해야만 합니다.
- 전역 변수의 이름은 `g_`로 시작해야만 합니다.
- 변수와 함수의 이름에는 소문자, 숫자 및 '`_`' (Unix Case)만이 포함될 수 있습니다.
- 파일 및 디렉터리의 이름에는 소문자, 숫자 및 '`_`' (Unix Case)만이 포함될 수 있습니다.
- 표준 ASCII 코드표에 없는 문자는 금지됩니다.
- 변수, 함수 및 기타 식별자는 스네이크 케이스를 사용해야 합니다. 대문자는 없고 각 단어는 밑줄 문자로 구분됩니다.
- 모든 식별자(함수, 매크로, 자료형, 변수 등)는 영어여야만 합니다.
- 객체(변수, 함수, 매크로, 자료형, 파일 또는 디렉터리)는 가능한 가장 명시적이거나 가장 연상되는 이름을 가져야 합니다.
- 프로젝트에서 명시적으로 허용하지 않는 한 상수(const) 및 정적(static)이 아닌 전역 변수 선언은 금지되며 Norm 오류로 간주됩니다.
- 파일은 컴파일이 가능해야 합니다. 컴파일되지 않는 파일은 Norm을 통과할 수 없을 것입니다.

III.2 서식

- 들여쓰기는 네 칸 크기의 텁으로 이루어져야 합니다. 일반적인 공백 네 칸이 아니라 진짜 텁을 말합니다.
- 각 함수는 함수 자체의 중괄호를 제외하고 최대 25줄이어야 합니다.
- 각 줄은 주석을 포함해 최대 80자의 열 너비를 가집니다. 주의: 텁 들여쓰기는 한 열로 계산하지 않으며, 텁이 해당되는 공백의 수 만큼으로 계산됩니다.
- 각 함수는 줄 바꿈으로 구분해야 합니다. 모든 주석과 전처리기 명령은 함수 바로 위에 있을 수 있습니다. 줄 바꿈은 이전 함수 다음에 와야 합니다.
- 한 줄에 한 명령만이 존재할 수 있습니다.
- 빈 줄은 공백이나 텁 들여쓰기 없이 비어 있어야 합니다.
- 줄은 공백이나 텁 들여쓰기로 끝날 수 없습니다.
- 두 개의 연속된 공백이 있을 수 없습니다.
- 모든 중괄호나 제어 구조 뒤는 줄바꿈으로 시작돼야 합니다.
- 줄의 끝이 아니라면 모든 콤마와 세미콜론 뒤에는 공백 문자가 따라와야 합니다.
- 모든 연산자나 피연산자는 하나의 공백으로 구분해야 합니다.
- 각 C 키워드 뒤에는 공백이 있어야만 합니다. 자료형 키워드(int, char, float, 등)와 sizeof는 제외됩니다.
- 각 변수 선언은 해당 스코프와 같은 열로 들여쓰기 되어야만 합니다.
- 포인터와 함께 쓰이는 별표는 변수 이름에 붙어있어야만 합니다.
- 한 줄에 한 개의 변수 선언만이 가능합니다.
- 선언과 초기화는 같은 줄에서 작성될 수 없습니다. 다음 경우에는 제외됩니다. 전역 변수(허용 될때에), 정적 변수, 그리고 상수.
- 선언문은 함수의 처음에 존재해야 합니다.
- 함수 내의 변수 선언문과 이후 함수 사이에는 빈 줄이 존재해야만 합니다. 다른 빈 줄은 함수 내에서 허용되지 않습니다.
- 다중 대입은 엄격하게 금지됩니다.
- 명령문이나 제어 구조 다음에 새 줄을 추가할 수도 있습니다. 그러기 위해서는 들여쓰기와 함께 중괄호나 대입 연산자를 추가해야 합니다. 연산자는 줄의 시작에 있어야만 합니다.
- 제어문(if, while..)에는 한 줄인 경우를 제외하고 중괄호가 존재해야 합니다
- 함수, 선언문, 제어 구조 다음에 오는 중괄호 앞 뒤에는 줄바꿈이 있어야만 합니다.

일반적인 예시]:

```
int          g_global;
typedef struct s_struct
{
    char     *my_string;
    int      i;
}           t_struct;
struct      s_other_struct;

int      main(void)
{
    int      i;
    char     c;

    return (i);
}
```

III.3 함수

- 한 함수에는 최대 4개의 명명된 매개변수를 가질 수 있습니다.
- 인자를 받지 않는 함수는 "void"라는 단어를 인자로 명시적으로 프로토타입 돼야 합니다.
- 함수 프로토타입 안의 매개 변수는 명명되어야만 합니다.
- 각 함수는 빈 줄로 다음 함수와 구분되어야 합니다.
- 각 함수에서 5개를 초과하여 변수를 선언할 수 없습니다.
- 함수의 리턴은 괄호 사이에 있어야 합니다.
- 각 함수의 리턴 자료형과 함수 이름 사이에는 한 번의 탭 들여쓰기가 있어야 합니다.

```
int my_func(int arg1, char arg2, char *arg3)
{
    return (my_val);
}

int func2(void)
{
    return ;
}
```

III.4 자료형, 구조체, 열거형(enum)과 공용체(union)

- 구조체, 열거형, 공용체를 선언 할 때는 템 들여쓰기를 넣습니다.
- 구조체, 열거형, 공용체의 변수를 선언할 때에는 자료형 안에 공백 문자 하나를 넣습니다.
- 구조체, 열거형, 공용체를 typedef와 함께 선언할 때에는 모든 들여쓰기 규칙이 적용됩니다.
- typedef 이름은 템이 앞에 있어야만 합니다.
- 모든 구조체의 이름은 해당 스코프의 같은 열에 들여쓰기 해야만 합니다.
- .c 파일 내에서 구조체를 선언할 수 없습니다.

III.5 헤더 - a.k.a 파일 포함하기

- 헤더파일에서 허용되는 것들: 헤더 인클루드(시스템 헤더 또는 유저 헤더), 선언문, defines, 프로토타입과 매크로.
- 모든 인클루드는 파일의 시작에 작성되어야 합니다.
- C 파일을 포함할 수 없습니다.
- 헤더 파일은 중복 인클루드를 방지해야만 합니다. 만약 파일 이름이 ft_foo.h라면 인클루드 가드 매크로 이름은 FT_FOO_H 가 되어야 합니다.
- 사용하지 않은 헤더의 인클루드는 금지됩니다.
- .c / .h 파일의 모든 헤더 인클루드는 정당한 이유가 있어야만 합니다.

```
#ifndef FT_HEADER_H
#define FT_HEADER_H
#include <stdlib.h>
#include <stdio.h>
#define FOO "bar"

int           g_variable;
struct        s_struct;

#endif
```

III.6 42 헤더 - a.k.a 멋지게 파일 시작하기

- 모든 .c 및 .h 파일은 즉시 표준 42 헤더로 시작해야 합니다. : 유용한 정보를 포함하는 특별한 형식의 멀티 라인 주석. 표준 헤더는 다양한 텍스트 편집기를 위한 클러스터의 컴퓨터에서 자연스럽게 사용할 수 있습니다. (emacs : C-c C-h 입력, vim : :Stdheader 또는 F1, 등... 입력).
- 42 헤더에는 로그인과 이메일이 포함된 작성자, 생성일, 로그인 및 마지막 업데이트 날짜를 포함한 여러 최신 정보가 포함되어야 합니다. 파일이 디스크에 저장될 때마다 정보가 자동으로 업데이트되어야 합니다.

III.7 매크로와 전처리기

- 매크로 상수(또는 #define)는 리터럴이나 상수값에만 사용 가능합니다.
- Norm을 우회하거나 코드 가독성을 낮추는 모든 #define은 금지됩니다. 이 부분은 사람에 의해 검사되어야 합니다.
- 표준 라이브러리의 매크로는 프로젝트에서 사용이 허가되었을 경우에만 사용 가능합니다.
- 여러 줄에 걸친 매크로는 금지됩니다.
- 매크로 이름은 모두 대문자여야만 합니다.
- #if, #ifdef, #ifndef 다음 문자들은 들여쓰기 해야만 합니다.

III.8 금지 사항!

- 다음 구문은 사용이 금지됩니다:
 - for
 - do...while
 - switch
 - case
 - goto
- 다음과 같은 삼항 연산자 ‘?’
- VLA - 가변 길이 배열.
- 자료형을 명시하지 않은 변수 선언

```
int main(int argc, char **argv)
{
    int     i;
    char   string[argc]; // 가변 길이 배열
    i = argc > 5 ? 0 : 1 // 삼항 연산자
}
```

III.9 주석

- 주석은 함수 내부에 있을 수 없습니다. 주석은 줄 끝에 있거나 별개의 줄에 있어야만 합니다.
- 주석은 영어여야만 합니다. 그리고 유용해야만 합니다.
- 주석은 "쓰레기 같은" 함수를 정당화할 수 없습니다.

III.10 파일

- .c 파일을 인클루드할 수 없습니다.
- 하나의 .c 파일에 함수를 5개보다 많이 정의할 수 없습니다.

III.11 Makefile

Makefile은 Norm에서 확인하지 않으며, 반드시 학생이 평가 중에 확인해야만 합니다.

- 다음 규칙은 필수적입니다. \$(NAME), clean, fclean, re and all
- Makefile이 리링크(relink)되면, 프로젝트는 작동하지 않는 것으로 간주됩니다.
- 실행 파일이 여러 개인 프로젝트의 경우, 위의 규칙 이외에도 컴파일된 각각의 실행 파일에 대한 특정 규칙 뿐만 아니라 실행 파일들을 모두 컴파일하는 규칙이 있어야만 합니다.
- 비-시스템 라이브러리(예: libft)에서 함수를 호출하는 프로젝트의 경우 Makefile은 반드시 이 라이브러리를 자동으로 컴파일해야만 합니다.
- 프로젝트를 컴파일하기 위해 필요한 모든 소스파일들은 Makefile에 반드시 명시해야만 합니다.