제 1 부 — 기초

2장

프로그램 구조와 기본 문법

Free Form, Explicit Intent

소스 형식 · 프로그램 골격 · 식별자 · implicit none

PROGRAM SKELETON

```
program name
   use module
   implicit none
   [선언부]

   [실행부]

end program name
```

학습 목표

이 장에서 익히는 네 가지

01

자유형식 익히기

현대 Fortran의 자유형식으로 코드를 쓰고, 고정형식은 레거시를 읽기 위한 지식으로만 배운다.

02

프로그램의 골격

프로그램 단위와 문장이 어떻게 구성되는지, 선언부와 실행부의 순서를 파악한다.

03

식별자 명명

변수·상수·프로그램 이름에 적용되는 명명 규칙과 대소문자 비구분 특성을 익힌다.

04

implicit none 의무

모든 단위에 implicit none을 두어 선언하지 않은 변수를 원천 차단한다.

도입 질문

프로그램의 모양을 들여다보면

?

컴파일러는 프로그램이 어디서 시작하고 어디서 끝나는지 어떻게 아는가?

?

레거시 코드는 왜 줄 앞이 늘 비어 있고 들여쓰기가 고정되어 있는가?

?

어떤 코드는 .f90 이고 어떤 코드는 .f 인데, 둘은 무엇이 다른가?

?

변수명에 오타가 났는데 컴파일은 통과했고 답만 틀렸다. 왜 그런가?

이 장을 마치면 네 의문이 풀리고, 이후 모든 장에서 쓸 기본 골격을 갖추게 된다.

2.1 소스 형식

자유형식과 고정형식

자유형식

free form

Fortran 90에서 도입된 현대적 형식. 문장을 어느 칸에서 시작하든 상관없고 들여쓰기가 자유롭다.

```
.f90   .f95   .f03   .f08
```

현대 코드 — 본문 전체

고정형식

fixed form

FORTRAN 77 시절의 형식. 천공카드 시대의 흔적으로, 글자가 놓이는 칸 위치 자체에 문법적 의미가 있다.

```
.f   .for   .ftn
```

레거시 코드 — 읽기 전용

gfortran은 파일 확장자만 보고 소스 형식을 판별한다 — 자유형식 코드를 실수로 .f 로 저장하면 고정형식으로 해석되어 엉뚱한 오류가 난다.

2.1 자유형식

자유형식의 핵심 규칙 다섯

1. **최대 132자**
   한 줄은 표준상 132자까지 쓸 수 있다.

2. **주석은 !**
   느낌표부터 그 줄 끝까지 주석으로 인식된다.

3. **한 줄 여러 문장은 ;**
   세미콜론으로 문장을 구분해 나열한다.

4. **줄잇기는 &**
   줄 끝에 앰퍼샌드를 붙이면 다음 줄로 이어진다.

5. **대소문자 비구분**
   count, Count, COUNT는 모두 같은 이름이다.

```
! 이 줄 전체가 주석이다
a = 1; b = 2; c = 3

print *, "sum =", a + b + &
         c
```

가독성 먼저

세미콜론으로 한 줄에 여러 문장을 욱여넣으면 코드를 읽기 어려워진다. 기본 원칙은 한 줄에 한 문장이다.

2.1 [예제] 레거시 읽기

고정형식은 칸(column)이 곧 문법이다

```
%%writefile fixed_demo.f
C234567890  (column ruler)
C     Fixed-form: read-only legacy style
      program fixed_demo
      implicit none
      integer :: total
      total = 21 +
     &        21
      print *, "total =", total
      end program fixed_demo
```

실행 결과 `total =  42`

1열	C, * , ! 가 오면 그 줄 전체가 주석
1~5열	문장 라벨(번호) 자리. GOTO·FORMAT이 참조
6열	빈칸이 아닌 문자가 있으면 앞 줄에서 이어지는 문장
7~72열	실제 문장을 적는 영역. 73열부터는 완전히 무시

[흔한 실수] 오류 메시지가 statement label을 말하면 십중팔구 소스 형식을 잘못 지정한 것이다 — Non-numeric character in statement label

2.2 프로그램 단위와 문장 구조

프로그램은 단위(program unit)로 이루어진다

주 프로그램

`main program`

실행이 시작되는 지점. 한 실행 파일에 정확히 하나만 있어야 한다.

모듈

`module`

자료형, 상수, 프로시저를 모아 두는 단위.

외부·내부 프로시저

`procedure`

함수와 서브루틴을 정의하는 단위.

주 프로그램은 선언부와 실행부로 나뉘며, 반드시 선언부가 먼저 온다

1 선언부

모듈 사용(use), implicit none, 변수·상수 선언이 들어간다. 실제 동작은 일어나지 않는다.

2 실행부

대입, 계산, 입출력 등 실제 동작을 수행한다.

컴퓨터가 변수 공간의 크기와 개수를 완전히 파악한 뒤에야 연산을 시작할 수 있기 때문이다.

2.2 문법 규격

주 프로그램의 뼈대

```
program program_name
   ! 선언부
   use module_name
   implicit none
   [declarations]

   ! 실행부
   [statements]

[contains
   internal_procedures]
end program program_name
```

use 는 맨 앞에

use 문은 반드시 implicit none보다 먼저, 선언부의 최상단에 위치해야 한다.

end program 이름

end 만 써도 문법상 문제는 없지만, 가독성을 위해 항상 이름까지 적어 닫는다.

contains 는 나중에

contains 이후의 내부 프로시저는 10장에서 따로 다룬다.

이 순서는 Fortran 프로그래밍의 가장 기본적인 뼈대다. 코드를 쓸 때 이 순서를 엄격히 지킨다.

2.2 [예제]

선언부와 실행부 — 하루는 몇 초인가

```
%%writefile seconds_per_day.f90
program seconds_per_day
   implicit none
   integer :: hours, minutes, seconds, total

   hours   = 24
   minutes = 60
   seconds = 60
   total   = hours * minutes * seconds

   print *, "seconds in a day =", total
end program seconds_per_day
```

```
 seconds in a day =       86400
```

빈 줄 한 칸의 힘

선언부와 실행부 사이에 빈 줄을 하나 두면 코드의 구조가 한눈에 들어온다. 좋은 코딩 습관이다.

읽는 순서

- program 이름으로 시작
- implicit none 선언
- 정수 변수 네 개를 선언
- 값을 대입하고 곱해 출력

선언부가 모두 끝난 뒤에만 실행부 코드를 적을 수 있다 — 순서는 문법이다.

2.2 [예제] 문장 구조

세미콜론과 줄잇기

```
%%writefile statement_demo.f90
program statement_demo
   implicit none
   integer :: a, b, c

   a = 1; b = 2; c = 3          ! 한 줄, 세 문장
   print *, "sum =", a + b + &  ! 다음 줄로 이어짐
            c
end program statement_demo
```

```
                    sum =           6
```

; 세미콜론

문장과 문장 사이에 넣으면 한 줄에 여러 개의 독립된 명령을 나열할 수 있다. 다만 가독성이 떨어지므로 권장하지 않는다.

& 줄잇기 문자

문장 끝에 붙이면 다음 줄이 앞 문장의 연속으로 인식된다. & 뒤에 꼬리 주석을 붙여도 줄잇기는 정상 동작한다.

[흔한 실수] & 없이 식을 다음 줄로 넘기면 식이 잘려 Syntax error in expression 오류가 난다.

2.3 주석

소스 코드는 기계를 위한 것, 주석은 사람을 위한 것

```
! This whole line is a comment.
total = a + b    ! a trailing comment
```

1 **의도 전달**
코드가 '무엇을' 하는지는 코드가 말해 주지만, '왜' 그렇게 썼는지는 주석만이 설명할 수 있다.

2 **유지보수 비용 절감**
복잡한 알고리즘이나 특이한 예외 처리를 남겨 두면, 나중에 분석하는 시간을 크게 줄여 준다.

3 **협업의 효율화**
동료가 내 코드를 읽을 때 훌륭한 가이드가 되어 불필요한 질문과 시행착오를 막는다.

주석은 시간이 흐른 뒤의 자신과 동료를 위한 이정표다

좋은 코드는 주석이 필요 없을 만큼 명확해야 한다. 그러나 복잡한 로직이나 의사결정의 배경을 남기는 주석은 여전히 필수적이다.

`!` 부터 줄 끝까지는 컴파일러가 무시한다

2.3 식별자

변수·상수·프로그램 이름 짓기

1 시작 문자

반드시 영문자로 시작한다. 숫자로 시작할 수 없다.

2 허용 문자

영문자, 숫자, 밑줄(_)만 쓸 수 있다.

3 길이 제한

최대 63자까지 (Fortran 2003 이후).

4 금지 항목

공백과 특수문자(-, +, ., 한글 등)는 쓸 수 없다.

5 대소문자 무관

count, Count, COUNT는 모두 같은 식별자다.

유효한 이름

```
Speed        time_step        temp_1
```

무효한 이름

```
2nd_run    숫자로 시작
delta-t    빼기로 해석됨
n value    토큰 두 개로 쪼개짐
```

키워드는 예약어가 아니다

integer :: if 처럼 키워드와 같은 이름의 변수를 문법적으로는 선언할 수 있다. 그러나 가독성을 크게 해치고 혼란을 부르므로 절대 쓰지 않는다.

2.3 [예제]

대소문자를 구분하지 않는다

```
%%writefile case_demo.f90
program case_demo
   implicit none
   integer :: count

   count = 10
   COUNT = COUNT + 5    ! count 와 COUNT 는 같은 이름
   print *, "count =", count
end program case_demo
```

```
                    count =          15
```

무슨 일이 일어났나

C나 파이썬이라면 두 단어를 다른 변수로 보고 에러를 냈겠지만, Fortran은 count와 COUNT를 하나의 같은 변수 방으로 안내한다. 그래서 값이 누적되어 15가 나온다.

그래도 섞어 쓰지 말 것

편해 보이지만, 대소문자를 섞어 쓰면 가독성이 급격히 떨어진다. 하나의 이름은 처음부터 끝까지 하나의 표기법으로 일관되게 쓴다.

2.4 implicit none

암묵적 형 지정이라는 함정

선언하지 않은 변수를 쓰면, 이름의 첫 글자만 보고 자료형이 자동으로 결정된다.

I J K L M N

`integer` 로 자동 선언

그 밖의 모든 글자

`real` 로 자동 선언

타이핑은 줄여 주지만, 치명적인 함정을 만든다

변수 이름에 오타를 내면 컴파일러가 오류를 내는 대신 값이 0인 새 변수를 조용히 만들어 버린다. 프로그램은 멀쩡히 실행되면서 틀린 답을 낸다.

해결책은 하나 — 모든 프로그램·모듈·프로시저에 예외 없이 `implicit none` 을 둔다.

오류 학습 [예제]

같은 오타, 전혀 다른 결말

implicit none 없이

```
program no_implicit_bug
   fahrenheit = 100.0
   celsius = (fahreheit - 32.0) * 5.0 / 9.0
   print *, "celsius =", celsius
end program no_implicit_bug
```

경고만 뜨고 실행됨 `celsius =  -17.7777786`

오타 fahreheit 가 값 0.0인 새 실수 변수로 조용히 만들어졌다. (0.0 − 32.0) × 5.0 ÷ 9.0 이 계산되어 −17.78이 나왔다. 정답은 37.78이다.

implicit none 을 넣으면

```
program implicit_fixed
   implicit none
   real :: fahrenheit, celsius
   fahrenheit = 100.0
   celsius = (fahreheit - 32.0) * 5.0 / 9.0
```

컴파일 거부 `Symbol 'fahreheit' has no IMPLICIT type`

컴파일러는 did you mean 'fahrenheit'? 까지 제안한다. 조용한 논리 버그가 요란한 컴파일 오류로 바뀌었고, 개발자는 그 자리에서 고칠 수 있다.

implicit none 의 자리는 use 문 다음, 변수 선언문 앞 — 예외는 없다.

오류 학습 [예제]

선언부와 실행부의 순서 위반

```
program decl_after_exec
   implicit none
   integer :: a
   a = 10
   integer :: b      ! 실행문 뒤의 선언
   b = 20
   print *, a + b
end program decl_after_exec
```

```
Error: Unexpected data declaration statement at (1)
Error: Symbol 'b' at (1) has no IMPLICIT type
```

왜 안 되는가

모든 변수 선언은 실제 연산을 수행하는 명령보다 물리적으로 앞서야 한다. 컴퓨터가 변수 공간의 크기와 개수를 완전히 파악한 뒤에야 연산을 시작할 수 있기 때문이다.

고치는 법 — 선언을 선언부로 모은다

```
implicit none
integer :: a, b
a = 10
b = 20
```

[오류 메시지 읽기] gfortran은 줄 번호와 1 표시로 위치를 정확히 가리키고, 종종 원인과 올바른 철자까지 제안한다. 첫 번째 메시지부터 끝까지 읽는 습관이 디버깅 시간을 줄인다.

오류 학습

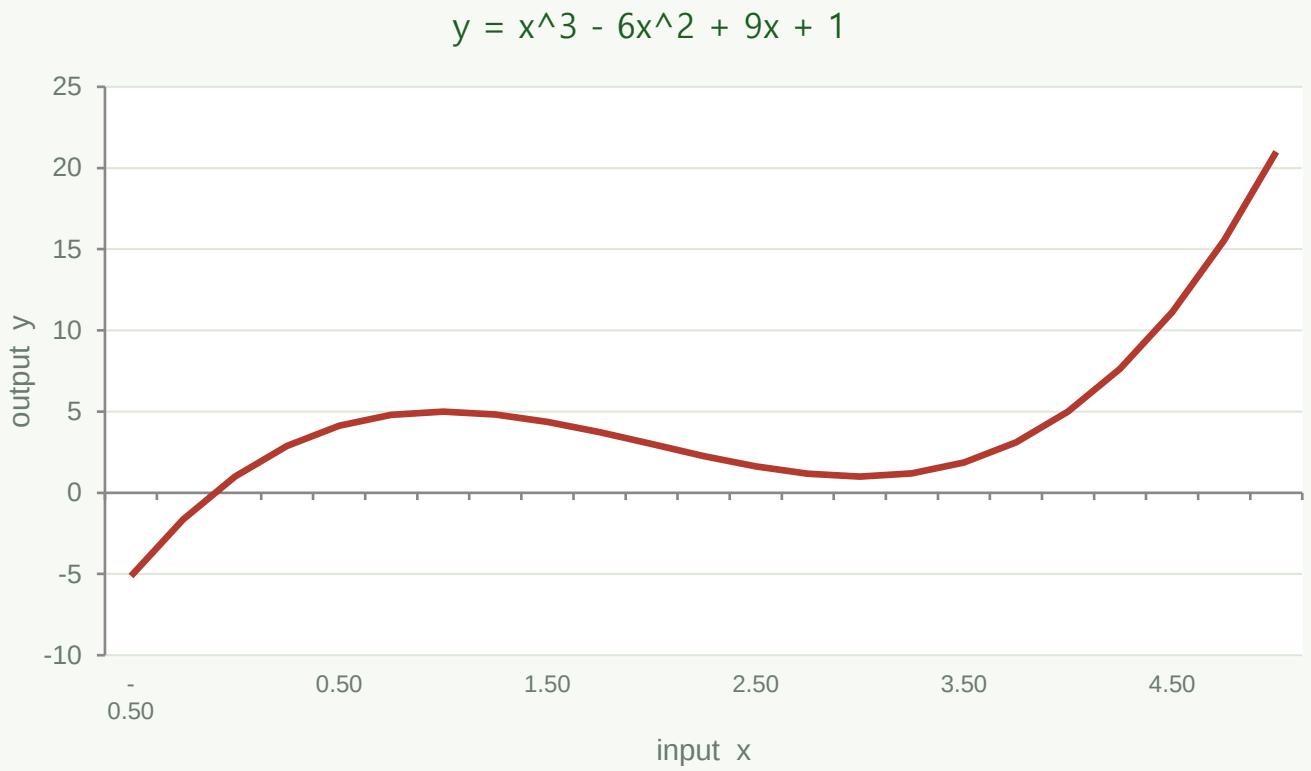
이 메시지가 뜨면, 이걸 의심하라

`Non-numeric character in statement label`	**소스 형식 오지정**	자유형식 코드를 .f 확장자로 저장했다. .f90 으로 바꾼다.
`Syntax error in expression`	**줄잇기 & 누락**	식을 다음 줄로 넘길 때 줄 끝에 & 를 붙이지 않았다.
`Unexpected data declaration statement`	**선언 순서 위반**	실행문 뒤에 선언문이 왔다. 선언을 선언부로 옮긴다.
`Symbol '...' has no IMPLICIT type`	**미선언 또는 오타**	implicit none 아래에서 선언되지 않은 이름을 썼다.
`is used uninitialized [-Wall 경고]`	**암묵적 형 지정 발동**	implicit none 이 없어 오타가 새 변수로 만들어졌다.

좋은 오류 메시지는 훌륭한 선생이다 — 일부러 실수를 만들어 컴파일러의 경고를 해석하는 훈련이 실력이 된다.

2.4 [예제] 입력에서 출력으로

3차 함수의 흐름을 직접 계산하기




```
do i = 0, n
   x = -0.5 + real(i) * 0.05
   y = x**3 - 6.0*x**2 &
       + 9.0*x + 1.0
   write(u,'(F8.3,",",F10.4)') x, y
end do
```

읽는 법

거듭제곱 ** 은 곱셈 * 보다 우선순위가 높다. 그래서 괄호를 많이 쓰지 않아도 수학 표기와 거의 같게 읽힌다.

x = 1 부근에서 봉우리(y ≈ 5), x = 3 부근에서 골(y ≈ 1)을 지나 다시 가파르게 치솟는 3차 함수 특유의 형태가 드러난다.

알아두기

컴파일과 빌드는 어떻게 다른가

컴파일 (Compile)

소스 코드를 CPU가 실행할 수 있는 기계어로 번역하는 단일 작업 . hello.f90 을 목적 파일(.o)로 바꾸는 순수한 번역 과정이다.

빌드 (Build)

소스 코드가 실행 가능한 완성된 프로그램이 되기까지의 모든 과정. 컴파일은 빌드에 포함되는 하나의 핵심 단계다.

빌드의 세 단계

1 **전처리**

`Preprocessing`

코드 안의 매크로와 설정을 먼저 정리한다.

2 **컴파일**

`Compile`

코드를 기계어로 번역해 부품(목적 파일)을 만든다.

3 **링크**

`Link`

부품들과 외부 수학 라이브러리(BLAS, LAPACK)를 엮어 실행 파일을 만든다.

간단한 예제에서는 명령 한 줄로 실행 파일까지 나오므로 두 말을 섞어 쓰지만, 파일과 라이브러리가 수백 개 얽힌 대규모 계산 프로그램에서는 둘을 명확히 구분한다.

2장 정리

요약

형식과 골격

- 현대 코드는 자유형식 .f90 — 고정형식 .f 는 읽기용. gfortran은 확장자로 형식을 판별한다.
- 골격: program 이름 → 선언부(use, implicit none, 선언) → 실행부 → end program 이름.
- 한 줄 여러 문장은 ; , 줄잇기는 줄 끝의 & , 주석은 ! , 거듭제곱은 ** .

이름과 안전장치

- 식별자는 영문자로 시작, 이후 영문자·숫자·밑줄, 최대 63자, 대소문자 비구분.
- implicit none 은 모든 프로그램·모듈·프로시저에 예외 없이 — 조용한 오타 버그를 컴파일 오류로 바꾼다.
- 오류 진단은 gfortran 메시지의 줄 번호와 위치 표시를 끝까지 읽는 데서 시작한다.

한 줄 요약: 자유형식으로 쓰고, 선언부를 먼저 두고, implicit none 을 빠뜨리지 않는다.

연습 문제

1. program hello ... end program hello 골격에 implicit none 을 넣고 자기 이름을 출력하라.

2. Speed, 2nd_run, time_step, delta-t, n value, temp_1 이 유효한 식별자인지 판정하라.

3. width, height 에 7과 5를 넣고 넓이를 출력하라. 선언부와 실행부를 빈 줄로 나눠라.

4. 한 줄에 세 개의 대입문을 ; 로 잇고, 긴 print 문 하나를 & 로 두 줄에 걸쳐 써라.

5. 고정형식 코드의 출력을 손으로 예측한 뒤, 같은 동작의 자유형식 .f90 으로 다시 써라.

6. area = base * height / 2 ! 주석 에서 컴파일러가 실제로 처리하는 부분만 옮겨 적어라.

7. c = 37.0 을 화씨로 바꿔 출력하라. 9 / 5 로 적으면 결과가 어떻게 달라지는지 설명하라.

8. implicit none 을 빼고 오타 든 프로그램을 만들어, 경고 메시지와 오류 메시지를 비교하라.

9. 분 단위 시간을 "몇 시간 몇 분"으로 나눠 출력하라. 정수 나눗셈과 mod 를 쓴다.

10. parameter 로 선언한 pi 에 값을 다시 대입하면 어떤 메시지가 나오는가? 이유를 설명하라.

다음 장에서는 자료형과 변수 선언, 정밀도(kind)와 상수를 본격적으로 다룬다.