

RUST 언어 기본 및 HIGHT 구현

송민호

유튜브: <https://youtu.be/XcHIMGeJ8IE>

RUST

- 안전하고 빠른 시스템 개발을 위한 언어

- 안전성

- 메모리 안전성과 스레드 안전성을 강조하는 언어
- RUST 컴파일러는 실행 시간 오류를 줄이기 위해 메모리 오류를 검출

- 높은 성능

- 꽤 많은 경우에서 C와 비슷한 수준의 성능
- LLVM을 사용하여 코드를 컴파일하고 최적화하여 빠른 실행 속도 제공

- 확장성

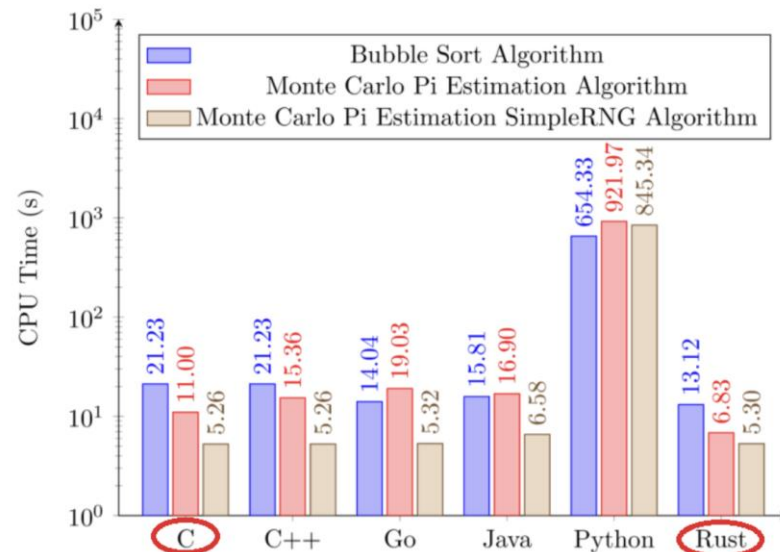
- 다양한 운영체제와 아키텍처를 지원
- 크로스 플랫폼 개발에 적합하며 다양한 분야에서 사용

- 생산성

- 코드의 가독성을 강조
- 표준 라이브러리와 함께 제공되는 cargo를 통해 의존성 관리와 빌드 자동화를 지원

- 지속 가능성

- 대형 기업들이 개발하고 사용하고 있음
- 다양한 커뮤니티와 생태계가 활발하게 운영



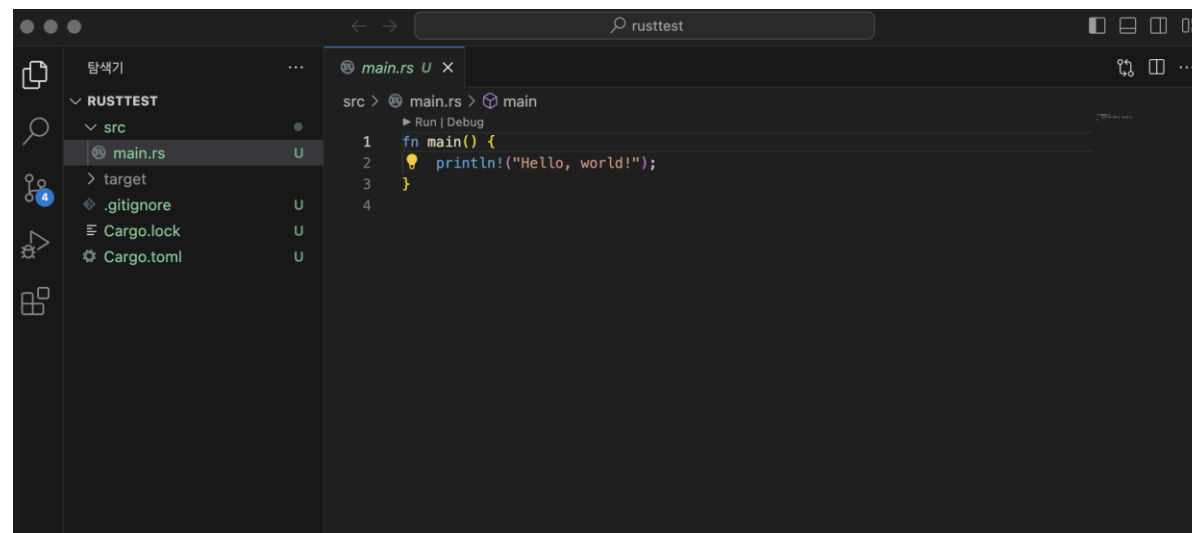
CPU time 벤치마크

사용 방법

- <https://play.rust-lang.org/>

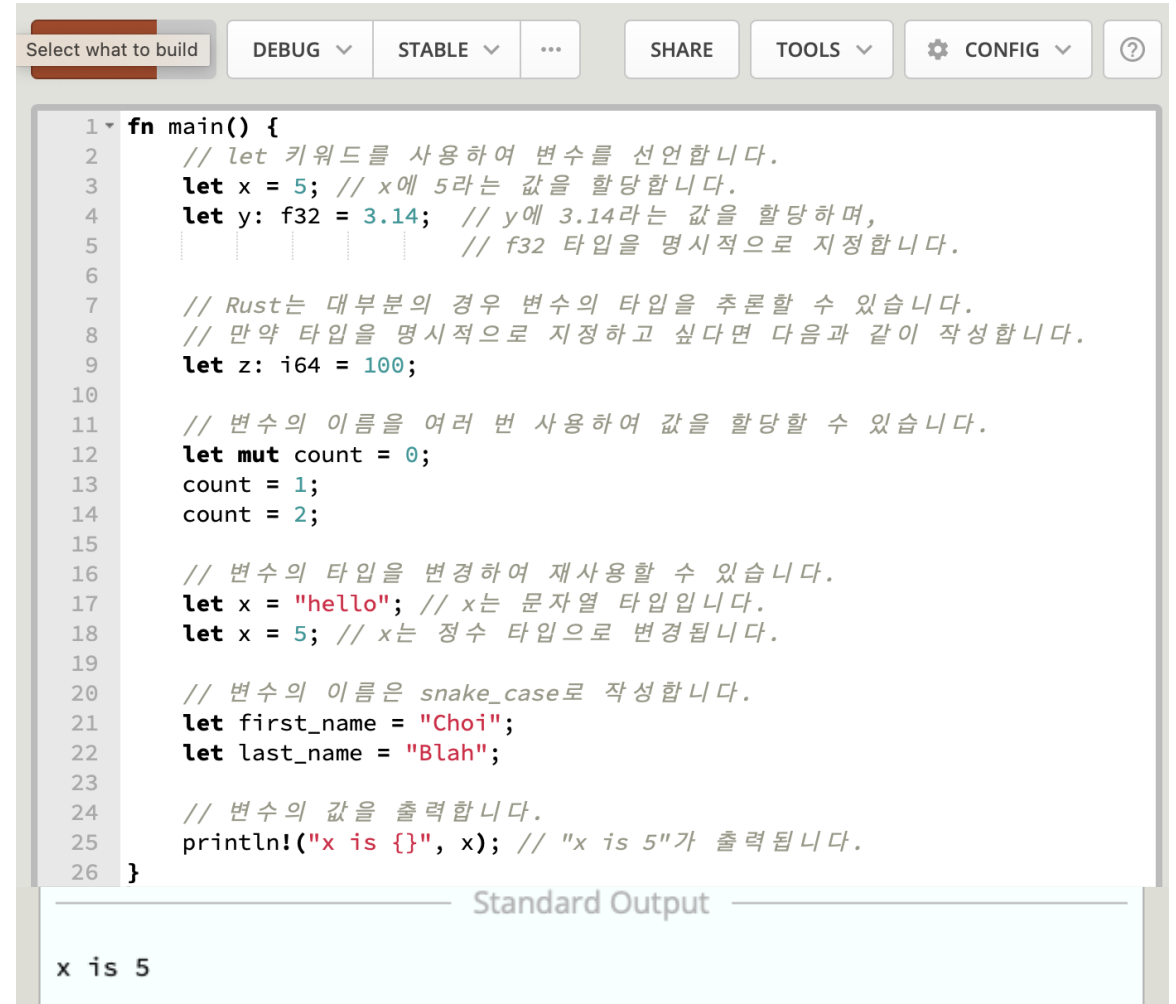


- `$ curl https://sh.rustup.rs -sSf | sh`
- `Cargo new` 프로젝트 이름
- Vscode에서 오픈



변수

- let 키워드를 사용하여 변수 선언
- 값을 할당할 때, RUST는 거의 대부분 변수의 타입을 추론할 수 있음
- RUST가 추론하지 못하면 변수의 선언 시 타입을 추가할 수 있음
 - let x = 5;
 - let y: f32 = 3.14;
 - let z = 7u8;
- 변수의 이름을 여러 번 사용하여 값 할당 가능
 - 변수의 타입은 재할당될 때마다 변경됨
- 변수의 이름은 항상 **snake_case**로 작성



The screenshot shows a Rust IDE interface. At the top, there are buttons for 'Select what to build', 'DEBUG', 'STABLE', 'SHARE', 'TOOLS', 'CONFIG', and a help icon. The code editor displays a Rust function `main()` with the following content:

```
1 fn main() {  
2     // let 키워드를 사용하여 변수를 선언합니다.  
3     let x = 5; // x에 5라는 값을 할당합니다.  
4     let y: f32 = 3.14; // y에 3.14라는 값을 할당하며,  
5     // f32 타입을 명시적으로 지정합니다.  
6  
7     // Rust는 대부분의 경우 변수의 타입을 추론할 수 있습니다.  
8     // 만약 타입을 명시적으로 지정하고 싶다면 다음과 같이 작성합니다.  
9     let z: i64 = 100;  
10  
11    // 변수의 이름을 여러 번 사용하여 값을 할당할 수 있습니다.  
12    let mut count = 0;  
13    count = 1;  
14    count = 2;  
15  
16    // 변수의 타입을 변경하여 재사용할 수 있습니다.  
17    let x = "hello"; // x는 문자열 타입입니다.  
18    let x = 5; // x는 정수 타입으로 변경됩니다.  
19  
20    // 변수의 이름은 snake_case로 작성합니다.  
21    let first_name = "Choi";  
22    let last_name = "Blah";  
23  
24    // 변수의 값을 출력합니다.  
25    println!("x is {}", x); // "x is 5"가 출력됩니다.  
26 }
```

Below the code editor, there is a 'Standard Output' window showing the result of the program execution:

```
x is 5
```

변수 업데이트

- RUST는 변수가 변경 가능한지 여부에 대해 많은 주의를 기울임

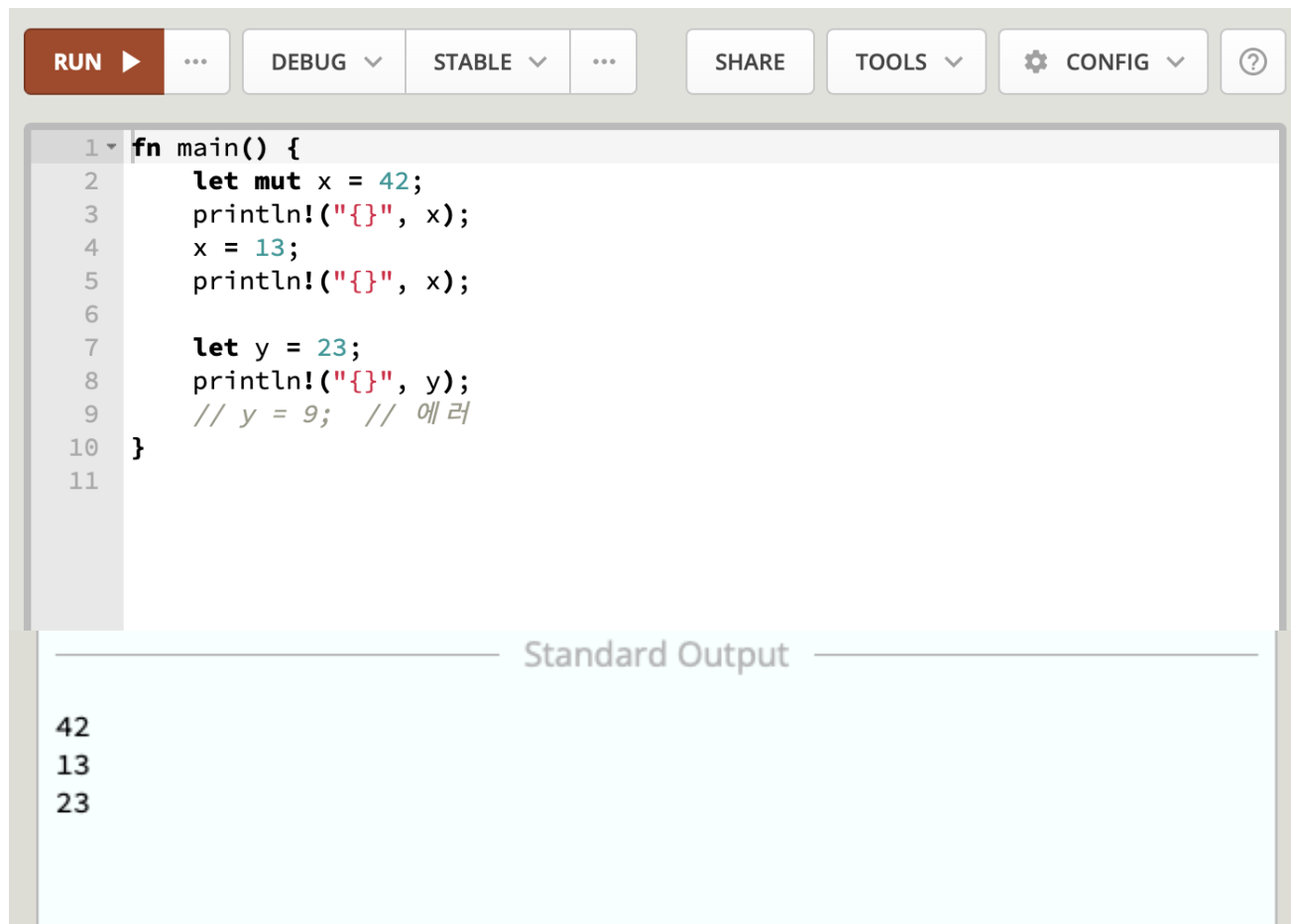
- Mutable(가변)

- 컴파일러는 해당 변수에 대해
쓰거나 읽을 수 있도록 허용

- Immutable(불변)

- 컴파일러는 해당 변수에 대해
읽기만 가능하도록 허용

- 가변 값은 mut 키워드로 구분됨



```
1 fn main() {  
2     let mut x = 42;  
3     println!("{}", x);  
4     x = 13;  
5     println!("{}", x);  
6  
7     let y = 23;  
8     println!("{}", y);  
9     // y = 9; // 에러  
10 }  
11
```

Standard Output

```
42  
13  
23
```

데이터 타입

- Booleans – bool로 참/거짓(1 byte)
- Unsigned integers – 음이 아닌 정수
 - u8, u16, u32, u64, u128
- Signed integers – 정수
 - i8, i16, i32, i64, i128
- Pointed sized integers
 - 메모리에서 색인과 항목의 크기를 나타냄
 - usize, isize(보통 4 or 8 bytes)
- Floating point
 - f32, f64
- Tuple – 고정된 값의 시퀀스를 스택에 전달
 - (value, value, ...)

RUN ▶

...

DEBUG ▾

STABLE ▾

⋮

SHARE

TOOLS ▾

⚙️ CONFIG ▾

?

```
1 ▾ fn main() {  
2     let x = 12; // by default this is i32  
3     let a = 12u8;  
4     let b = 4.3; // by default this is f64  
5     let c = 4.3f32;  
6     let bv = true;  
7     let t = (13, false);  
8     let sentence = "hello world!";  
9 ▾ println!(  
10         "{} {} {} {} {} {} {} {} {}",  
11         x, a, b, c, bv, bv as u8, t.0, t.1, sentence  
12     );  
13 }  
14
```

Standard Output

```
12 12 4.3 4.3 true 1 13 false hello world!
```

배열

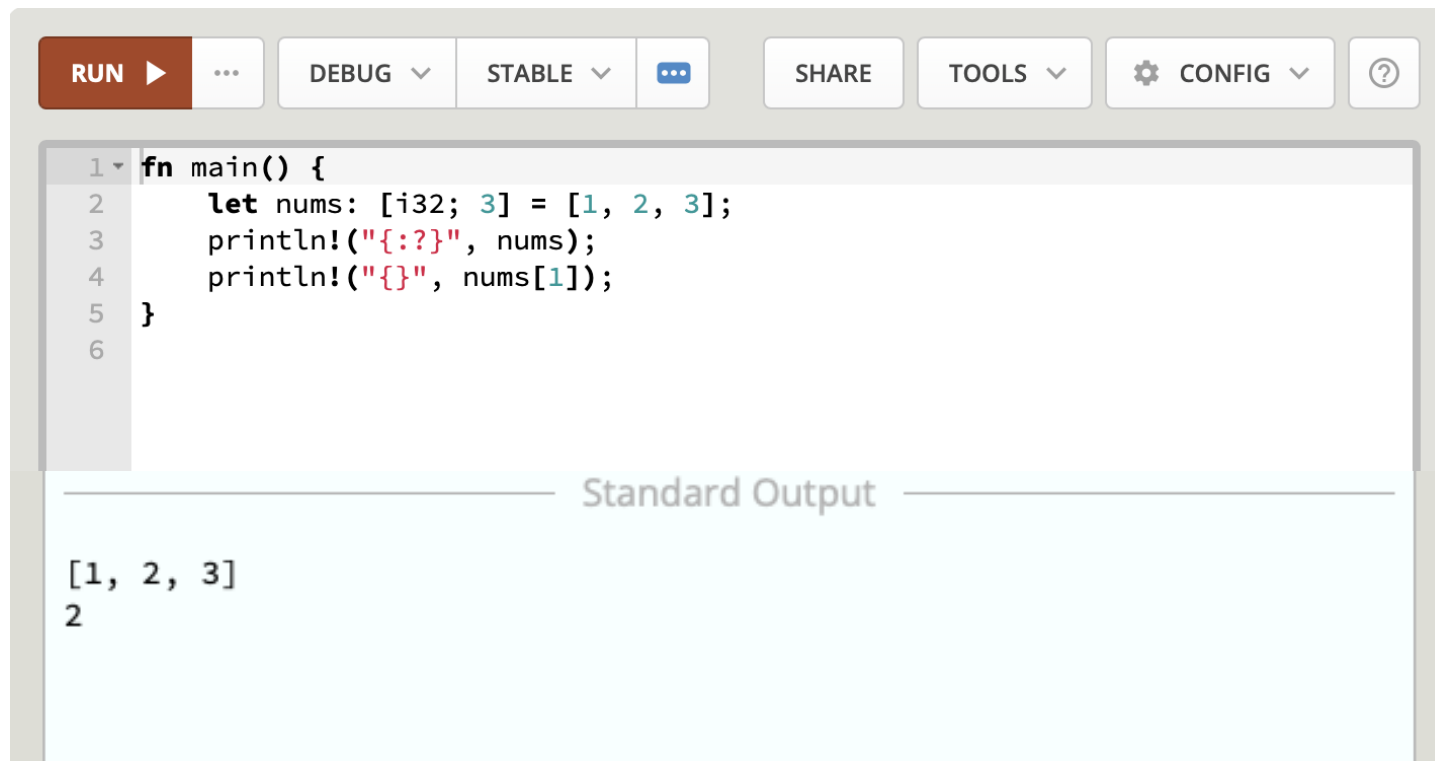
- 동일한 유형의 데이터 요소들의 고정된 길이 집합

- 데이터 유형 – $[T; N]$

- T : 요소의 데이터 타입
 - N : 배열의 길이

- 개별 요소

- $[x]$ 연산자를 사용하여 가져올 수 있음



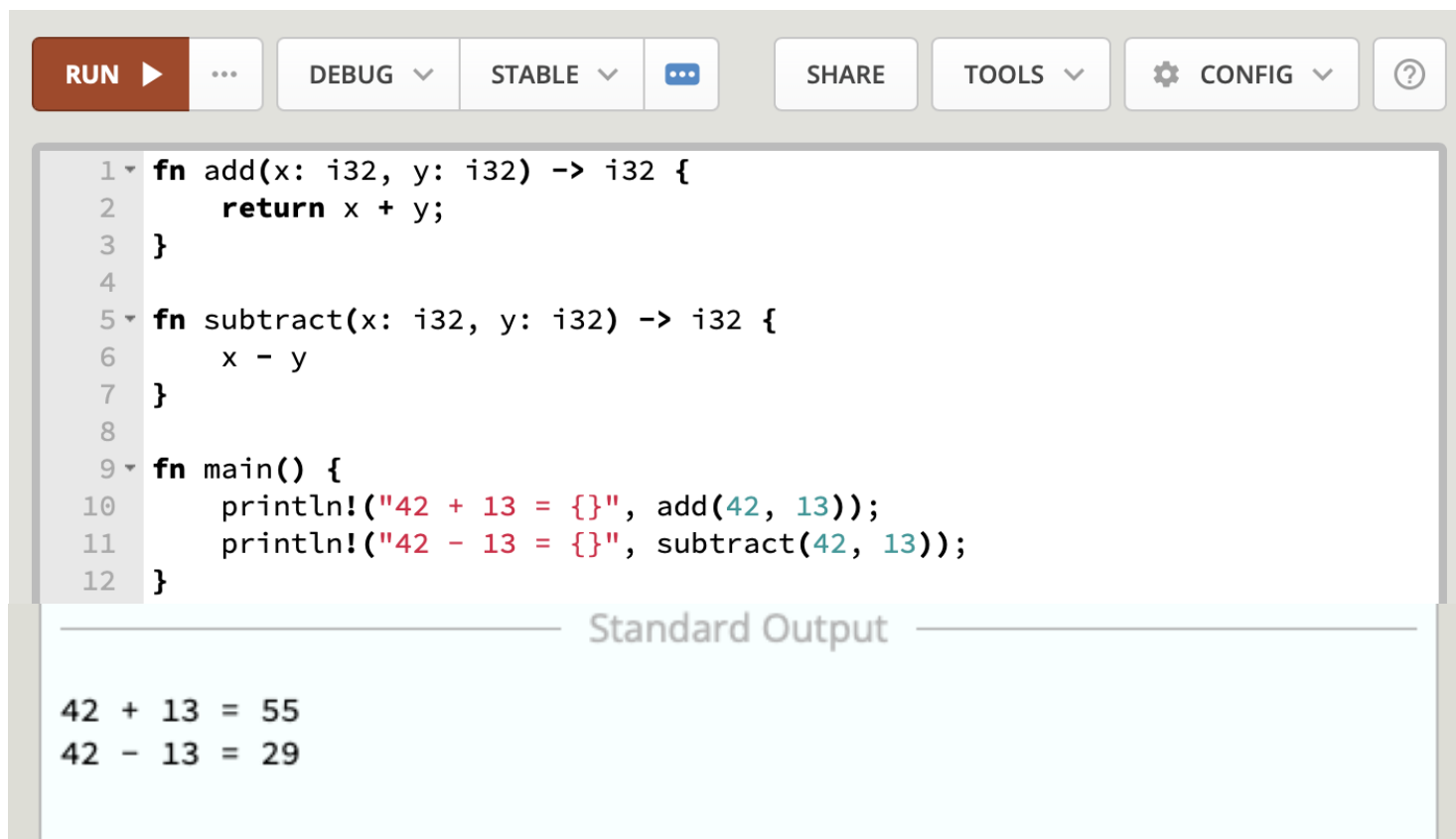
```
1 fn main() {  
2     let nums: [i32; 3] = [1, 2, 3];  
3     println!("{:?}", nums);  
4     println!("{}", nums[1]);  
5 }  
6
```

Standard Output

```
[1, 2, 3]  
2
```

함수

- 표현식만 반환 가능
 - return 키워드 삭제
 - 세미콜론 삭제
- 함수 크기 지정
 - ->를 통해 지정 가능
- 함수 이름
 - `snake_case`



The screenshot shows a code editor with the following Rust code:

```
1 fn add(x: i32, y: i32) -> i32 {  
2     return x + y;  
3 }  
4  
5 fn subtract(x: i32, y: i32) -> i32 {  
6     x - y  
7 }  
8  
9 fn main() {  
10     println!("42 + 13 = {}", add(42, 13));  
11     println!("42 - 13 = {}", subtract(42, 13));  
12 }
```

Below the code editor, the "Standard Output" window displays the results of the program execution:

```
42 + 13 = 55  
42 - 13 = 29
```


조건문과 반복문

- If문

- 주어진 조건이 참인 경우 실행

```
if 조건 {  
    // 조건이 참일 때 실행할 코드  
}
```

- For문

- 반복 가능한 객체를 순회하며 실행

```
// Range를 사용한 for 문  
for 변수 in 시작..끝_미포함 { // range(시작, 끝)  
    // 코드 블록  
}  
  
for 변수 in 시작..=끝_포함 { // range(시작, 끝 + 1)  
    // 코드 블록  
}
```

- While문

- 주어진 조건이 참인 경우 반복 실행

```
while 조건 {  
    // 조건이 참인 동안 반복할 코드  
}
```

- Loop문

- 무한 반복 실행

```
// Loop 키워드를 사용한 무한 반복  
loop {  
    // 무한 반복할 코드  
    // 'break'를 사용해 반복을 종료할 수 있음  
}
```

조건문과 반복문

- 레이블

- while, for, loop문에서 이름을 지정하고 break 가능
- 이름:** 형식으로 작성
- Break할 때 **break 이름;**을 사용

```
1 #[allow(unused_assignments)]
2
3 fn main() {
4     // 레이블을 사용한 while 문 예제
5     let mut x = 0;
6     let mut y = 0;
7
8     'outer_while: while x < 5 {
9         y = 0;
10
11         while y < 5 {
12             if x * y > 10 {
13                 println!("조건 만족: x={}, y={}", x, y);
14                 break 'outer_while;
15             }
16             y += 1;
17         }
18         x += 1;
19     }
20
21
22     // 레이블을 사용한 for 문 예제
23     'outer_for: for x in 0..5 {
24         for y in 0..5 {
25             if x * y > 10 {
26                 println!("조건 만족: x={}, y={}", x, y);
27                 break 'outer_for;
28             }
29         }
30     }
```

```
32 // 레이블을 사용한 loop 문 예제
33 let mut x = 0;
34 let mut y = 0;
35
36 'outer_loop: loop {
37     y = 0;
38
39     loop {
40         if x * y > 10 {
41             println!("조건 만족: x={}, y={}", x, y);
42             break 'outer_loop;
43         }
44
45         y += 1;
46         if y >= 5 {
47             break;
48         }
49     }
50
51     x += 1;
52     if x >= 5 {
53         break;
54     }
55 }
56 }
```

Standard Output

조건 만족: x=3, y=4
조건 만족: x=3, y=4
조건 만족: x=3, y=4

소유권

- 소유권
 - 각 값은 하나의 소유자를 가짐
 - 소유자가 범위를 벗어나면 값이 할당 해제
- 소유권 이전
- 값을 새 변수에 할당할 때
 - s1이 가리키는 값의 소유권이 s2로 이전되어 s1은 사용 불가능
- 값을 새 변수에 할당할 때
 - s가 함수에 전달되면서 소유권이 이전되고 함수 내에서 값의 메모리가 해제

```
let s1 = String::from("hello");  
let s2 = s1;
```

메모리 관리를 단순화하고 안전하게 할 수 있음

```
fn takes_ownership(s: String) {  
    println!("{}", s);  
}  
  
fn main() {  
    let s = String::from("hello");  
    takes_ownership(s);  
    // 여기에서 s는 사용할 수 없습니다.  
}
```

빌림

- 빌림
 - 소유권을 이전하지 않고 값을 참조하거나 사용할 수 있게 해주는 기능
- 불변 빌림, 가변 빌림
 - 불변: 여러 개의 불변 참조를 가져올 수 있음
 - 가변: 한 개의 가변 참조, 해당 변수의 다른 참조가 없어야함

```
fn main() {  
    let mut s = String::from("hello");  
    change(&mut s);  
    println!("The new string is '{}'.", s);  
}  
  
fn change(s: &mut String) {  
    s.push_str(", world!");  
}
```

소유권을 이전하지 않고도 값을
안전하게 사용할 수 있음

데이터 경쟁 문제를 방지하고,
메모리 관리에 대한 안전성을 높
일 수 있음

HIGHT 구현

- HIGHT

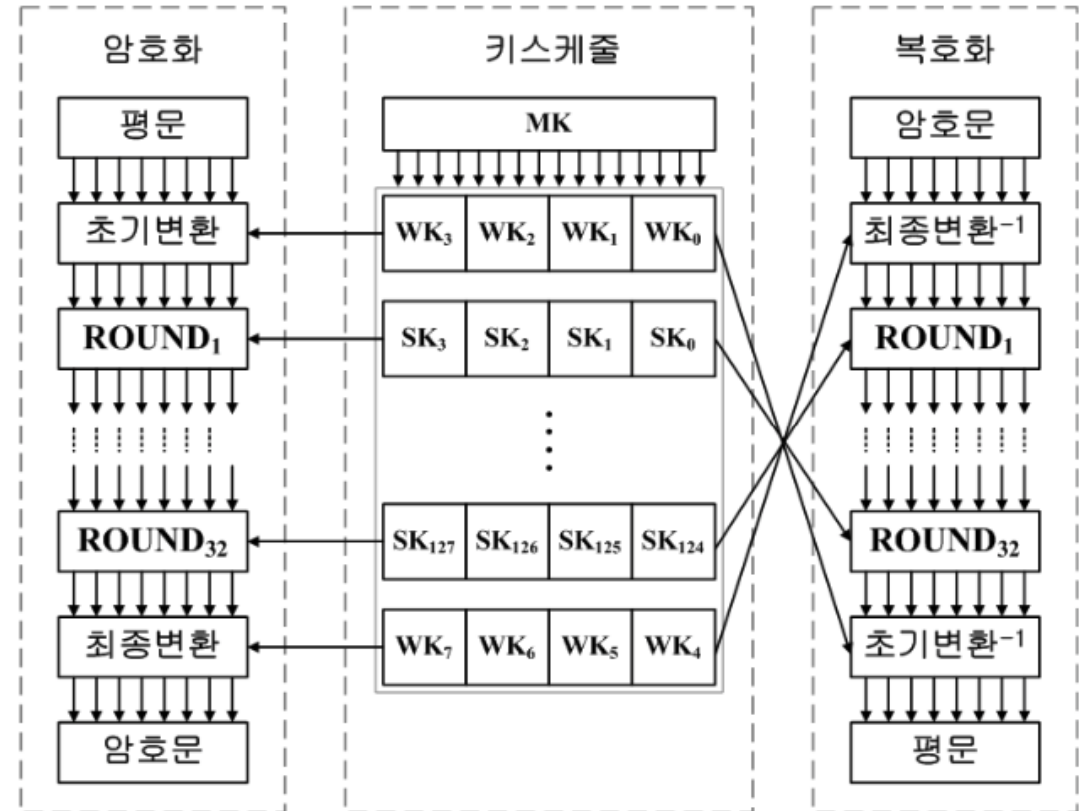
- Feistel 구조의 64비트 암호문

- 화이트닝 키, 서브 키 사용

- 128비트 마스터 키로부터 생성
 - 8개의 8비트 화이트닝 키
 - 128개의 8비트 서브 키

- 32라운드 진행

- 라운드 앞뒤로 초기변환, 최종변환



HIGHT 구현

- 키 스케줄
 - 화이트닝 키와 LFSR을 사용하여 생성한 서브 키로 이루어짐

- 화이트닝 키 생성

$$WK_i = \begin{cases} MK_{i+12} & , 0 \leq i \leq 3 \\ MK_{i-4} & , 4 \leq i \leq 7 \end{cases}$$

- 서브 키 생성

```
For i=0 to 7
  For j=0 to 7
     $SK_{16 \cdot i + j} \leftarrow MK_{j-i \bmod 8} \boxplus \delta_{16 \cdot i + j};$ 
  For j=0 to 7
     $SK_{16 \cdot i + j + 8} \leftarrow MK_{(j-i \bmod 8) + 8} \boxplus \delta_{16 \cdot i + j + 8};$ 
```

```
fn keyschedule(rk:&mut [u8; 136], mk: &[u8; 16]) {  
    // WK 생성  
    for i in 0..4 {  
        rk[i] = mk[i+12];  
        rk[i+4] = mk[i];  
    }  
  
    // SK 생성  
    for i in 0..8 {  
        for j in 0..8 {  
            rk[8+16*i+j] = mk[(j.wrapping_sub(i))&7].wrapping_add(DELTA[16*i+j]);  
            rk[8+16*i+j+8] = mk[((j.wrapping_sub(i))&7)+8].wrapping_add(DELTA[16*i+j+8]);  
        }  
    }  
}
```

RUST에서는 초과하는 비트를 오버플로우로
잡아서 wrapping을 통해 연산 진행

HIGHT 구현

- 변환

- 초기변환, 최종변환으로 이루어짐
- 화이트닝 키를 통해 변환 진행

- 초기변환

- WK_0, WK_1, WK_2, WK_3 를 이용
- 평문 P 를 첫 번째 라운드 함수의 입력인 $X_0 = X_{0,7} \parallel X_{0,6} \parallel \cdots \parallel X_{0,0}$ 로 변환

$$\begin{aligned} X_{0,i} &= P_i, \quad i = 1, 3, 5, 7 \\ X_{0,0} &= P_0 \boxplus WK_0 \\ X_{0,2} &= P_2 \oplus WK_1 \\ X_{0,4} &= P_4 \boxplus WK_2 \\ X_{0,6} &= P_6 \oplus WK_3 \end{aligned}$$

- 최종변환

- WK_4, WK_5, WK_6, WK_7 를 이용
- $Round_{32}$ 의 출력인 $X_{32} = X_{32,7} \parallel \cdots \parallel X_{32,0}$ 을 암호문 C 로 변환

$$\begin{aligned} C_i &= X_{32,i}, \quad i = 1, 3, 5, 7 \\ C_0 &= X_{32,0} \boxplus WK_4 \\ C_2 &= X_{32,2} \oplus WK_5 \\ C_4 &= X_{32,4} \boxplus WK_6 \\ C_6 &= X_{32,6} \oplus WK_7 \end{aligned}$$

HIGHT 구현

- 변환

- 초기 변환

```
fn initial_transformation(pt:&mut[u8; 8], rk:&mut [u8; 136]) {  
  
    pt[0] = pt[0].wrapping_add(rk[0]) & 0xff;  
    pt[2] = pt[2] ^ rk[1];  
    pt[4] = pt[4].wrapping_add(rk[2]) & 0xff;  
    pt[6] = pt[6] ^ rk[3];  
  
}
```

- 최종 변환

```
fn final_transformation(pt:&mut[u8; 8], rk:&mut [u8; 136]) {  
  
    let mut state: [u8; 8] = [0; 8];  
  
    state[1] = pt[2];  
    state[3] = pt[4];  
    state[5] = pt[6];  
    state[7] = pt[0];  
  
    state[0] = pt[1].wrapping_add(rk[4]);  
    state[2] = pt[3] ^ rk[5];  
    state[4] = pt[5].wrapping_add(rk[6]);  
    state[6] = pt[7] ^ rk[7];  
  
    for i in 0..8 {  
        pt[i] = state[i];  
    }  
  
}
```


HIGHT 구현

- 암호화 라운드 함수

두 개의 보조 함수를 가짐

$$F_0(X) = X^{\ll 1} \oplus X^{\ll 2} \oplus X^{\ll 7}$$

$$F_1(X) = X^{\ll 3} \oplus X^{\ll 4} \oplus X^{\ll 6}$$

Round₁부터 Round₃₁까지

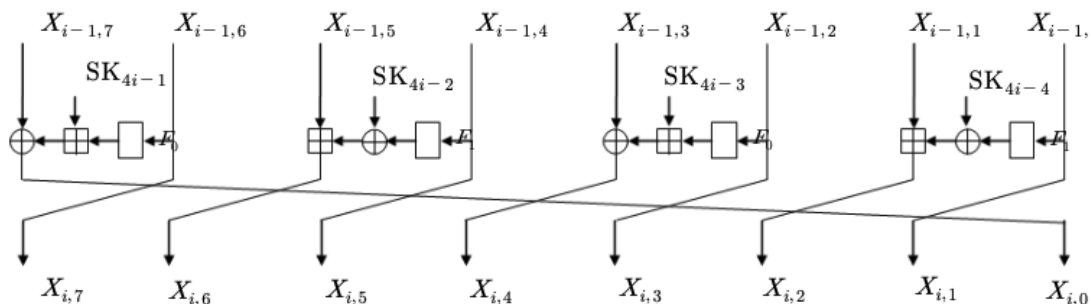
$$X_{i,j} = X_{i-1,j-1}, \quad j = 1, 3, 5, 7$$

$$X_{i,0} = X_{i-1,7} \oplus (F_0(X_{i-1,6}) \boxplus SK_{4i-1})$$

$$X_{i,2} = X_{i-1,1} \boxplus (F_0(X_{i-1,0}) \oplus SK_{4i-4})$$

$$X_{i,4} = X_{i-1,3} \oplus (F_0(X_{i-1,2}) \boxplus SK_{4i-3})$$

$$X_{i,6} = X_{i-1,5} \boxplus (F_0(X_{i-1,4}) \oplus SK_{4i-2})$$



HIGHT의 i 번째 라운드 함수 $Round_i$, $i = 1, \dots, 31$

Round₃₂

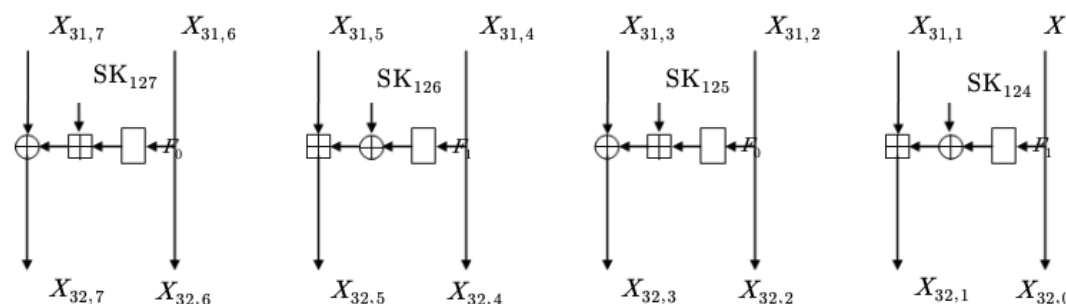
$$X_{32,i} = X_{31,i}, \quad i = 0, 2, 4, 6$$

$$X_{32,1} = X_{31,1} \boxplus (F_1(X_{31,0}) \oplus SK_{124})$$

$$X_{32,3} = X_{31,3} \oplus (F_0(X_{31,2}) \boxplus SK_{125})$$

$$X_{32,5} = X_{31,5} \boxplus (F_1(X_{31,4}) \oplus SK_{126})$$

$$X_{32,7} = X_{31,7} \oplus (F_0(X_{31,6}) \boxplus SK_{127})$$



HIGHT의 32번째 라운드 함수

HIGHT 구현

- 암호화 라운드 함수

```
fn round_function(pt:&mut[u8; 8], rk:&mut [u8; 136], r: usize, i0: usize, i1: usize, i2: usize, i3: usize, i4: usize, i5: usize, i6: usize, i7: usize,) {  
    pt[i0] = (pt[i0] ^ (HIGHT_F0[pt[i1] as usize].wrapping_add(rk[4*r+3]))) & 0xFF;  
    pt[i2] = (pt[i2].wrapping_add(HIGHT_F1[pt[i3] as usize] ^ (rk[4*r+2]))) & 0xFF;  
    pt[i4] = (pt[i4] ^ (HIGHT_F0[pt[i5] as usize].wrapping_add(rk[4*r+1]))) & 0xFF;  
    pt[i6] = (pt[i6].wrapping_add(HIGHT_F1[pt[i7] as usize] ^ (rk[4*r]))) & 0xFF;  
}
```

```
round_function(&mut pt, &mut rk, 2, 7,6,5,4,3,2,1,0);  
round_function(&mut pt, &mut rk, 3, 6,5,4,3,2,1,0,7);  
round_function(&mut pt, &mut rk, 4, 5,4,3,2,1,0,7,6);  
round_function(&mut pt, &mut rk, 5, 4,3,2,1,0,7,6,5);  
round_function(&mut pt, &mut rk, 6, 3,2,1,0,7,6,5,4);  
round_function(&mut pt, &mut rk, 7, 2,1,0,7,6,5,4,3);  
round_function(&mut pt, &mut rk, 8, 1,0,7,6,5,4,3,2);
```

Q & A