**Flagger v1.0 설명서**

**개요**

Flagger은 단순한 프로그래밍 언어이다.

튜링 완전성이 보장되어 있다. 즉, 컴퓨터로 풀 수 있는 모든 문제를 풀 수 있다.

**특징**

변수의 개념이 없다.

여러 코드의 동작이 동시에 일어난다.

논리 연산자를 지원하지 않는다. (And, or 등…)

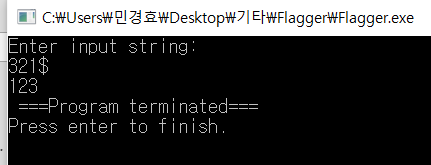
모든 명령어는 소문자이다.

**사용 방법**



위와 같이 소스 파일을 “Source.txt” 라고 짓고(**대소문자 주의**) Flagger.exe를 누른다.

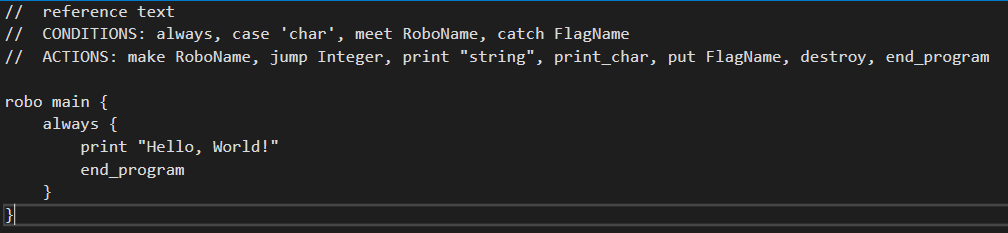
이후 프로그램에 입력할 값을 콘솔창에 치고 엔터를 누른다.



**문법**

예시를 보자.

**예시 1: Hello, World!**



맨 위: **주석. //로 줄을 시작하면 주석을 넣을 수 있다.** 저 주석은 이 언어에서 지원하는 **모든** 명령어가 적혀 있으므로, 프로그램을 짤 때 위에 두고 참고하면 좋다.

복사를 위한 내용:

// reference text

// CONDITIONS: always, case 'char', meet RoboName, catch FlagName

// ACTIONS: make RoboName, jump Integer, print "String", print\_char, put FlagName, destroy, end\_program

robo main { main이라는 이름의 로봇을 정의

always { 조건: always =매 턴마다 실행

print "Hello, World!" 출력: 뒤의 내용 출력

end\_program 프로그램 종료

}

}

모든 행동은 로봇 (robo)에 의해 일어난다. 처음에는 “main”이라는 이름의 로봇만 한 마리 존재하며, **main이라는 이름의 로봇은 반드시 필요하다**.

로봇이 하는 모든 행동(ACTION : end\_program 등)은 조건(CONDITION : always 등) 안에 적혀 있어야 한다.

**이때, 로봇이 한 턴 안에 하는 행동은 순서의 영향을 받지 않는다. 즉, 동시에 일어난다.**

지금 우리는 매 턴마다 (always) 할 일 두 가지를 적어 두었다.

이 두 가지 일은 반드시 ‘모두’, ‘동시에’ 일어난다.

예를 들어,

always {

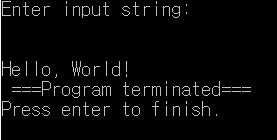
end\_program

print "Hello, World!"

}

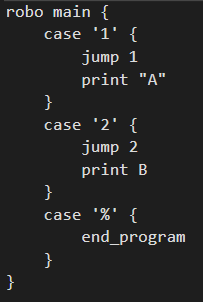
아까 코드에서 end\_program을 먼저 써도 원래 코드와 동일한 코드이다. end\_program을 print보다 먼저 수행하는 것이 아니기 때문에 Hello, World! 는 이 경우에도 정상적으로 출력된다.

결과:

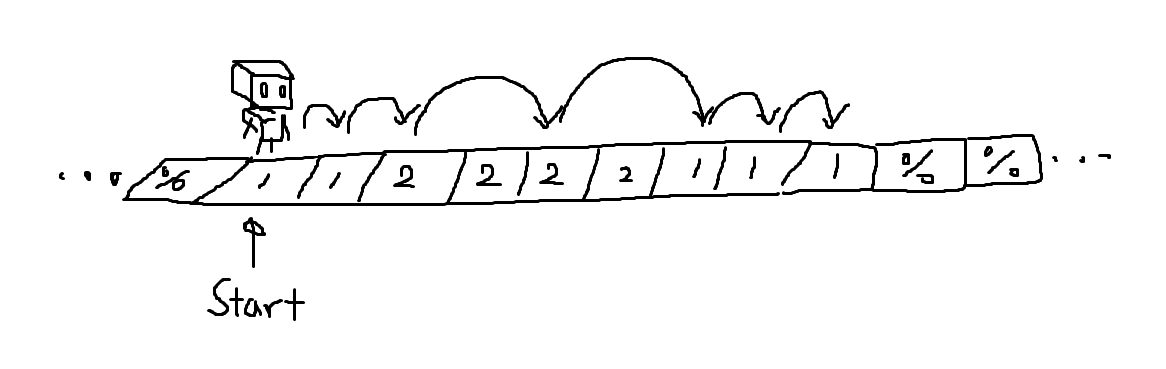


**예시 2: 걸어다니는 로봇들**

**로봇들은 입력 문자 위를 돌아다닌다고 생각할 수 있다.**



**처음엔 main 로봇 하나가 첫번째 글자 위에 놓여 있다.**



따라서 만일 “112222111” 이 입력으로 들어왔다면,

Main은 바닥에서 1을 읽고(case ‘1’) 오른쪽으로 한 칸(jump 1) 이동한다. 그리고 A를 출력한다.

(왼쪽으로 이동하고 싶다면 음수를 넣어두면 된다)

그러다가 2를 읽으면(case ‘2’) 오른쪽으로 두 칸 이동하며(jump 2) B를 출력한다.

**입력 String을 벗어나면 바닥은 ‘%’기호로 가득 차 있다.** %기호를 읽으면 프로그램을 종료한다.

따라서 **출력 결과는 AABBAAA 이다.**

참고: 위 프로그램은 1,2 말고 다른 글자가 있으면 무한루프에 걸린다.

**예시 3. 상호작용하는 로봇들**

Condition 명령어

**meet** RoboName {

}

이름(타입)이 RoboName인 로봇과 같은 칸에 놓여 있을 시 작동한다.

Action 명령어

**destroy :** 자기 자신을 파괴한다.

**make** RoboName: 이름(타입)이 RoboName인 로봇을 **현재 자신이 있는 위치에** 새로 만든다

.

\*이름은 Type이므로 같은 이름의 로봇은 여럿 있을 수 있다.

**문제:**

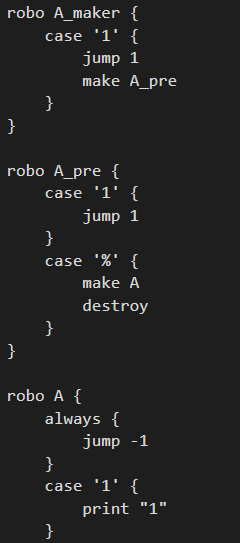
입력으로 1이 n개 들어온다. 이때, 출력으로는 n^2개의 1을 출력하시오.

Ex) 111 -> 111111111111

입력으로 1이 n개 있을 시 1을 n개 출력하는 로봇(A라 하자)은 ‘예시 2’를 참고하면 만들 수 있다.

따라서 그런 로봇(A)을 n개 만든다면 문제가 해결된다.

그리고, 오른쪽으로 걸어가면서 1을 볼 때마다 (A)를 만들라고 하면 A를 n개 만들 수 있다.

A를 만드는 제작자. 오른쪽으로 가면서 1이 있을 때마다 A를 만든다.

\*이때, A는 문자열의 끝에서 시작해야 제대로 작동할 것이므로 A를 바로 만드는 대신 문자열의 끝에 도달하면 A를 만드는 A\_pre를 만든다. (make A\_pre)

A\_pre는 오른쪽으로 가다가 문자열의 끝에 도달하면 A를 만든다.

그러고 나면 자폭한다. (destroy)

A는 왼쪽으로 가면서 1을 n개 출력한다.

이때 유의해야 하는 점은 모든 턴의 ACTION은 동시에 일어난다는 점이다.

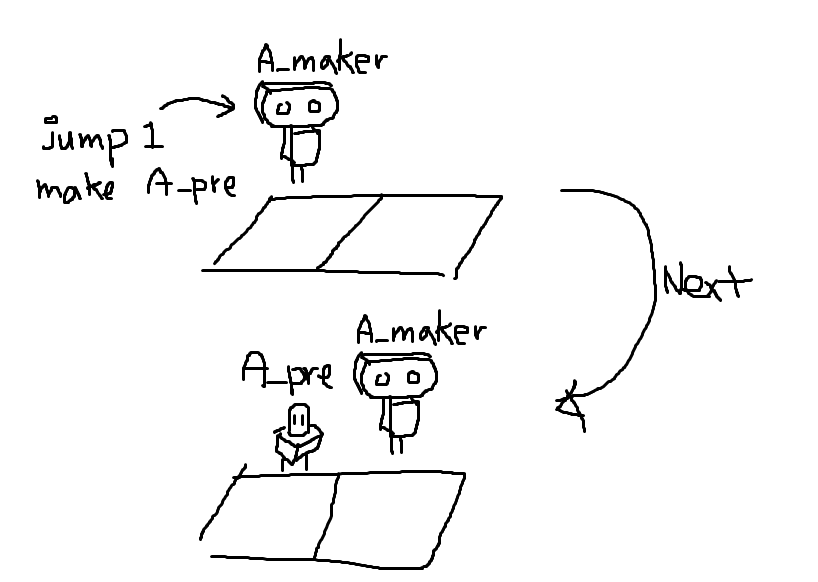
즉 맨 위 줄의

jump 1

make A\_pre

는 ‘오른쪽으로 한 칸 가고 나서 A\_pre를 만드는 것’이 아니다.

즉, A\_pre는 무조건 원래 자신이 있던 위치에 생성된다. (jump와 무관)



또한, 모든 ACTION은 턴의 끝에 실행되기 때문에, 이 턴에 A\_pre의 코드는 하나도 작동하지 않는다.

마지막으로, 모든 로봇의 CONDITION 검사와 ACTION은 동시에 일어난다.

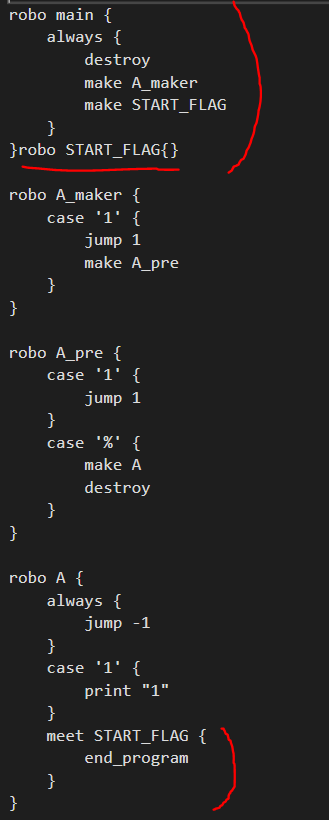
즉, 여러 로봇이 있을 시 어떤 로봇이 먼저 작동할지를 걱정할 필요는 없다. 예를 들어 위의 A\_pre와 A\_maker에 모두 ‘jump 1’ 이 있지만 A\_pre의 것이 먼저 작동해서 둘이 만난다거나 할 일은 없다. (동시에 움직인다)

이때, 앞의 코드는 잘 작동하기는 하지만 프로그램 종료를 언제 해야 하는지를 알기 힘들다.

예전처럼 A가 %를 만나면 종료한다고 하면 편하겠지만, 이번엔 A의 시작지점이 오른쪽 %이기 때문에 그러면 시작하자 마자 종료가 되어 버린다.

따라서 프로그램이 시작할 때에 ‘시작 지점’에 시작이라는 것을 알리는 로봇을 하나 세워 놓고,

A가 그 로봇을 만나면 종료하도록 하자.

빨간 표시가 된 곳은 새롭게 추가된 내용이다.

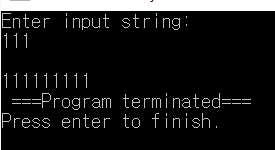
Main은 처음에 A\_maker과 시작지점 로봇(START\_FLAG)를 만든다. 그 후 자폭한다.

START\_FLAG와 만날 시 프로그램 종료.

이때, A가 여러 개니까 다른 A가 작동하는 도중 프로그램이 종료되는 것을 걱정할 수도 있는데,

A의 위치가 모두 같으며 (겹쳐서 돌아다닌다),

이 턴의 모든 로봇의 액션을 시행하고 나서야 프로그램이 종료되므로 (A의 end\_program은 다른 모든 로봇의 코드가 시행되고 나서야 작동) 문제가 되지 않는다.



**예시 4. 깃발꼽는 로봇들**

\*\*이 기능은 최적화 외에는 거의 쓸 일이 없으므로 굳이 알 필요는 없다.

Action

**put** FlagName : 깃발 이름 FlagName을 **현재 자신의 위치로** 옮긴다.

**print\_char:** 자신이 놓여 있는 위치의 Char을 출력한다. 예를 들어 로봇이 A위에 있다면 A출력.

Input의 범위를 벗어나 있다면 %출력.

Condition

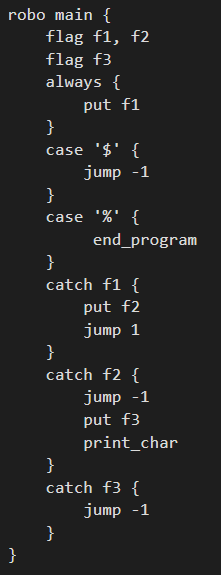
**catch** FlagName : FlagName 깃발과 만날 시(같은 위치가 될 시) 작동한다.

이때 flag는 마치 로컬 변수와 같아서, 다른 로봇들과 위치가 공유되지 않는다. 심지어 이름(타입)이 같은 로봇이라고 해도, 각기 다른 flag위치를 가진다.

또한, flag는 이름별로 하나이기 때문에, 같은 이름의 flag를 새로운 곳에 꼽으라고 명령하면 예전의 flag는 없어진다.

\*\*flag를 사용하기 위해서는 flag a 와 같이 선언을 해 주어야 한다.

예시:



푸는 문제: 입력이 “문자열$”꼴로 들어오면 문자열을 거꾸로 출력한다. 로봇은 하나만 사용.

예: 1234$ -> 4321

코드를 설명하기는 난해하다. 이해하고 싶다면 직접 시뮬레이션 하는 것이 빠르다.

**마무리**

위에 소개된 것이 Flagger언어의 모든 기능이다. 잘 개발해 보기 바란다.

(매우 어려운) 연습 문제

1. 입력으로 1이 여러 n개(n >= 2) 주어진다. n이 3의 배수라면 “Yes”, 아니면 “No”를 출력하라.

예: 111111 -> Yes, 1111 -> No

2. 입력되는 char은 1,2 뿐이다. 문자열이 정확히 “12” 인지 판별하라.

예: 12 -> Yes, 122 -> No, 21221111 ->No

3. 입력되는 char은 1\*2\* 꼴이다. 입력되는 문자열이 (1 n개 + 2 n개) 꼴인지 판별하라.

예: 11222 -> No, 1122 -> Yes, 111222 -> Yes, 1111122 -> No

(1과 2는 최소 2개씩은 있다)