//프로그래머는 괄호에 집착해야 해. 악플 쓰다가도 괄호 안 닫고 올렸으면, 새로운 리플 달아서라도 괄호를 닫는다고.

18.10.22

/설치 순서

-JDK. Docs : http://oracle.com

-Edit Plus : http://editplus.co.kr/kr/

JDK 설치 후 경로 복사 Java Developments Kit

C:\Program Files\Java\jdk1.8.0\_181\bin

내 PC 우클릭 - 속성 - 고급 설정 - 환경 변수 - 아래에 path 더블 클릭

- 새로 만들기 - 복사해놓은 설치 경로 붙여 넣기 - 추가된 해당 목록 가장 위로

/Editplus 깔면 처음 언어에 맞게 등록해줘야

도구 - 사용자 도구 구성 - 추가 - 프로그램 - 제목은 니 맘대로, 명령에서는

C:\Program Files\Java\jdk1.8.0\_181\bin 에서 "javac" 얘가 컴파일러야

인수는 파일 이름, 디렉토리는 파일 디렉토리

한 번 더해

이번엔

C:\Program Files\Java\jdk1.8.0\_181\bin 에서 "java.exe"

이건 뭐냐면 컴파일 된 파일을 실행시켜줌

인수 - 확장자 뺀 파일 이름, 디렉토리는 파일 디렉토리

/Class 뒤에 Test 띄고 1 하면 안 돼. 공백 없이 ex) Test 1 안 돼, 에러

Test\_1 이렇게 언더바를 쓰기도 하지만, 거의 안 써. 뭐 특별한 경우 아니면

Class 뒤에 첫 글자는 대문자로 시작함. 에러는 나지 않지만, 가독성을 위해 관례

test1 하지 말고 Test1

Class 뒤에 있는 제목이랑, 파일 저장 이름이랑 같이 하는 게 편해. 관례

18.10.23

/주석 종류와 방법

//은 한 줄 주석, 다음 줄 가면 주석 아님

/\*

\*/

이 사이에 있는 건 여러 줄 주석

class 라든가 괄호라든가, 무엇에 대한 class이고 괄호인지

주석 달아 놓으면 다음에 수정하거나 찾기 편하지 실수도 줄고

/어차피 최종적으로 연살할 때는 int 로 변환해서 써. 걍 첨부터 int 로 쓰기

정수 기본 자료형 – int , 실수 기본 자료형 – double

/프로그래밍이 그냥 영어식 사고방식이랑 똑같네. 어순이나.

class {　클래스에 대해 설명할게. 클래스란 뭐냐면~~}

main { 메인에 대해 설명할게. 뭘 메인이라 할 거냐하면~}

a = b + 1 = c \* 2 = d – 9 //a는 뭐냐하면 b + 1 인데, 그럼 b + 1 은 뭐냐하면,

c \* 2 인데, c \* 2는 뭐냐하면, d – 9 야.

이런 식이야.

I got some present for you that makes you happy

나는 선물을 널 위한 선물을 준비했어. 그리고 그 선물은 ‘뭐냐하면’ 너를 행복하게 해주는 거야.

이런 식인 거지.

/‘++’는 ‘증감연산자’라고 하는데, 앞에 붙을 때와 뒤에 붙을 때, 연산 우선순위가 달라.

b = ++a; 이러면 a에 +1 한 후에, b와 같아짐. but

b = a++; 이러면 a와 b가 같아진 후에, a에 +1이 됨.

늘어나기는 하는데, b에는 변화가 없어 보여. 그러고 난 다음에

c = a 하면 +1 돼 있겠지

/System.out.print(); 는 줄바꿈 없이 그냥 출력하는 거고, 그래서 띄어쓰기도 앞이나 뒤에

따로 일부러 넣어 줘야 해

System.out.println(); 이거는 print line 인 듯. 줄 바꿔주는 기능도 포함돼 있어

/자 봐 신기한 거 보여줄게

int a = 3; 해놓고

System.out.println(a\*8+7-1\*5);

이렇게 해놓으면 a는 함수 안에서만 연산한 것뿐이지, 함수 밖에서는 값이 변함이 없어.

System.out.println("a= " + a);

이렇게 다시 출력하면, 원래 a 값인 3으로 나오지.

/사용자한테 입력 받고 싶을 때. scanner를 사용함.

import java.util.Scanner; //스캔 사용 준비해라

Scanner scan(이거는 식별자) = new Scanner(System.in); //스캐너 생성

int x, y, z; //정수인 x랑 y랑 z가 있다고 하자.

x = scan. nextInt(); //정수 입력 받음

y = scan. nextInt(); //정수 입력 하나 더 받음

z = x + y; //입력 받은 정수 x와 y를 더한 게 z다.

System.out.println("둘이 더하면 " + z);

또는

System.out.println(scan. nextLine() + "는 바보 똥개 멍청이");

이런 식으로 하면 문장 단위로 가능. nextInt / nextLine / 등등만 바꾸면

입력 받을 정보의 형태만 바꿔서 가능

18.10.24

/boolean 은 참/거짓 자료형. true/false--이 이름은 boole 이라는 사람이 발견한 형식이라 그런 용어가 된 거.

/ boolean m; //m이라는 녀석은 true/false 로 나타나는 녀석이다

/ m = x < y; //m은, ‘x < y’가 true/false 인지 나타내는 값이다.

/ m = x = y; // m의 인스턴스는, x 이고, x의 인스턴스는 y이다? 뭐여. x랑 y는 Int라며. 반면에 m은 boolean이라며. true/false의 값을 갖는 m이 어떻게 정수가 돼?

“m의 값은 true/false를 나타내는 값인데, 뭐의 참/거짓이냐면, ‘x=y’에 대한 참/거짓이다.“

라고 말하고 싶으면, “m = x == y” 즉, =을 두 개 써 줘야 해. 참고로, = 은 단독으로 쓰이지 않아. 관계연산자로 쓰이는 등호( = )는 각각 다음가 같이 쓰여. ‘>= (왼쪽이 크거나 같다)’ , ‘<= (왼쪽이 작거나 같다)’ , ‘== (왼쪽과 오른쪽이 같다)’ , ‘!= (왼쪽과 오른쪽이 같지 않다)’ 이래서, 등호는 혼자 쓰이지 않아.

/연산자 종류 – p. 65

연산 우선 순위는

-산술 연산자 -> 관계 연산자 - > 논리 연산자 - > 삼항 연산자 -> 대립 연산자

1) 대입 연산자 - ( = )

2) 산술 연산자 - ( + / - / \* / / / % / ++ / --)

3) 관계 연산자 - ( > / >= / < / <= / == / != )

4) 논리 연산자 - ( && / || / ^ / ! ) 각각 ( and / or / Xor / not )

5) 비트 연산자 - ( & / | / ^ / ~ )

6) 비트 이동 연산자 - ( >> / >>> / << )

7) 혼합 연산자 - ( += / -= / \*= / /= / %= )

8) 삼항 연산자 - ( {조건식} ? {참일 때 취할 값} : {거짓일 때 취할 값} )

/논리 연산자의 경우, ~~~~ && ~~~~ 할 때, 전자가 거짓임이 판명 나면 후자는 연산을 시도하지도 않아. 이미 나왔으니까. 근데, 비트 연산자의 경우, 각 자리의 값을 모두 비교해야 하므로, 일단 연산은 끝까지 진행하기는 함. 끝까지 연산하는 게 필요한 경우, 논리 연산자 대신 비트 연산자를 사용하기도 함.

/자료형 변환 할 때

int tot, avg ;

avg = double (tot / 3) ; 그러면, 이미 tot/3 해서 (int이므로) 소수점 이하 날라가고, 그 후에 double로 바꿔 봐야, 끝에 ‘ .0 ’ 만 붙겠지.

/Scanner hi = new Scanner(System.in);

int a, b, c, d;

double m;

a = hi. nextInt();

b = hi. nextInt();

c = hi. nextInt();

d = hi. nextInt();

m = (double)(a + b + c +d) / 4 ;

/\*이렇게 하면, m의 값은, 지도 double로 표현하고 싶지만, 우변에서 모든 값이 int야. a,b,c,d 다 int로 받았고, 나누는 수도 자동으로 기본 설정 int이고. 그래서 소수점 이하가 나올 수 있는 수라도, 소수점 이하가 절사돼서 나와. (나오는 값은 마치 double 처럼 나올 거야. x.0 이런 식으로. 근데 그게 소수점 이하가 잘려 있는 상태로, ' .0 ' 만 붙는 거야.) 따라서 우변도 double로 바꿔 줘야 해. 방법이 두 가지야. 첫째, 나누는 수를 4.0 이런 식으로 써서, 기본 설정 double로 되도록 만드는 거. 그럼 연산하는 과정에서 자료형을 double 로 맞춰서 계산. 둘째, 나눠지는 수를 double 로 강제 변환. (double)(a + b + c + d)/4 이런 식으로. \*/

/int a,b,c,d;

double f;

a=10; b=5; c=4; d= 7;

f=(a+b+c+d)/(double)4;

System.out.println(f);

//분자나 분모 중 하나라도 double로 바꿔주면 된다는 거잖아

분모 double로 바꿔주는 방법 두 가지. 첫째는 4.0으로 써서 자동변환

둘째는 (double)4로 써서 강제 변환

\*/

181025

/a=127; /\*1byte 는 2^7까지. 127까지는 딱 되지만 128되는 순간 범위 넘어.

이진수로 나타내면 01111111(2) 야. (맨 첫 자릿수는 부호임, 0은 양수 1은 음수)

/d = 10000000000L; long형은 2^63 까지 가능하다며. 왜 이게 안 돼.

d는 내가 long으로 선언했어도, 정의하는 과정에서 상수 100억을 넣으면,

그 상수 100억을 기본 자료형인 int 형식으로 처리해서 d에다가 박아. 그니까

처리 과정에서 걸려버리는 거야. 그래서, 이거 상수이기는 하지만, 처리 과정에서부터,

기본값인 int말고 long으로 처리해달라고 일부러 말 해줘야 해. 그래서 끝에

' L '을 붙여. 10000000000L 이러면 long으로 처리.\*/

/g를 char로 선언하고

g = '가';

System.out.println( (int)g ); 이러면

g는 캐릭터이지만, 잠깐 int로 보겠다, ‘가’ 가 유니코드에 따라 등록돼 있는 그 숫자로 보여줘라.

/Scanner testin = new Scanner(System.in);

int test;

test = testin. nextInt();

System.out.println( (char) test + ", " + (char) ((int)test + 1) );

- 정수 하나를 입력 받아서, 해당 정수로 등록된 유니코드 상의 문자와 그 다음 문자를 출력.

println 안에서, ‘ (char) test + 1 ’이러면,

/-양쪽에 int면 그냥 진짜 더하기야

-양쪽에 char면, ' + ' 연산을 수행하기 위해, 둘 다 int로 변환해서 처리.

-String 타입이나 " " 안에 들어가 있는 문자열이 온다면 그냥 그대로 붙여서 읽어줌

-' + ' 기준으로 양쪽에 char 와 " " 가 있어. 그러면 " " 로 인해서, 덧셈 연산을

진행하는 게 아니므로, char는 그대로 문자로 나오고, 뒤에다가 문자열을 따라서 붙여 줌.

-' + ' 기준으로 양쪽에 int 와 " " 가 있어. 그러면 " " 로 인해서, 덧셈 연산을

진행하는 게 아니므로, int는 그대로 숫자로 나오고, 뒤에다가 문자열을 따라서 붙여 줌.

-int + char 라면, 연산하려면 두 자료형 통일 시켜줘야. 둘 중 큰 자료형 쪽으로

자동 변환. 즉, char에 해당하는 유니코드 숫자와 int를 ' + '연산 해서 int로 출력.

-따라서 int였던 a를 char로 강제 변환해서 (char) a + b 이러면,

a를 char로 강제 변환한 후에, b와 ' + ' 연산 수행하는 과정에서 둘이 다시

int로 자동 변환됨. 즉, 처리 우선 순위가 다음과 같음

a를 char로 강제 변환, a와 b를 +.

181026

//stack이 포인터, heap이 메모리, heap 에 저장된 정보가 '객체'

//char + char 는 int로 나오더니, string + string은 그대로 이어붙여줌

//buffer에서 정수만 쏙 가져다 쓰는 게 nextInt 근데 정수 90 쓰고 엔터

누르면, 그 엔터까지 버퍼에 있어. 근데 숫자만 가져다 쓰니까 버퍼에 엔터가

남아 있어. 그래서 다음 스캔 nextLine에서 버퍼 안이 비어 있지 않고 엔터가 있길래

그 엔터를 바로 가져다가 씀. 즉, 결론적으로, nextInt 이후에 나온 nextLine은

바로 엔터키를 가져오게 됨.

반면에, nextLine은 문자열 " "를 가져오고, 그 뒤의 엔터를 지워주고 나와.

그 이유는, (nextInt, nextDouble, nextByte 등)은 자료형이 숫자이므로, 숫자밖에

못 가져와. 즉

따라서 nextInt 이후에 nextLine으로 스캔 받고 싶으면, nextLine을 중간에

한번 더 넣어줘서 버퍼를 비워주는 과정을 추가해야 함.

그럼 어떤 변수를 내보내서 버퍼를 지우고 오라고 할 것인가? 이 전에 썼던 변수라면

그 변수가 '엔터키'로 초기화 돼서, 사라져 버림. 그니까 이번에 버퍼를 지우는

데에 사용하는 녀석은 이제 앞으로 변수를 또 다시 받게 될 애를 가져와야 함.

181029

/if 구문과 else if 구문, 그리고 else 구문

if 쓰고 소괄호 안에 조건 넣기. 그 조건을 만족할 경우 진행할 연산은, 이어서 중괄호 안에 넣기. 만족하지 않는 경우는 그냥 버림. 아무 일도 안 일어남.

ex) if (a < 100) { println(“잘가”);}

근데 그 조건을 만족하지 않을 때 어떤 일이 일어나도록 만들 수도 있음. else를 쓸 경우 (쓴다면 if와 동급으로 씀), if 에 달아 놓은 소괄호 안의 조건을 만족하지 않을 때, 특정 연산을 진행함. 진행할 특정 연산은 if와 마찬가지로 중괄호 안에 넣기.

ex) else (a < 70) {println(“감사”);

근데 else 안에서도 다시 또 경우를 나누고 싶어. if 구문 절에서 조건을 만족하면 A 행동을 하고, 만족하지 않으면 else를 통해서 B행동을 하는데, 다시 B 안에서도 B1인 경우와 B2인 경우로 세분화해서, B1에 해당하는 경우와 B2에 해당하는 경우로 나누고 싶어. 그러면 else에다가 if를 더 추가해서, else if를 쓴 후, 이어서 소괄호 안에 if와 마찬가지로 세분화 하고 싶은 조건을 넣어줘.

ex) else if (a > 50 ) { println(“지성”); }

else if (a < 50 ) { pirntln(“감사”); }

/\* 주의할 점 : if (조건) 하고 나서 ; 세미콜론 쓰지 않기!!! 세미콜론 쓰면 거기서 문장 끝내겠다는 뜻! 그러면, 말을 하다 마는 거여. ~~조건을 만족한다면. 하고 말을 끝내부러. ; 아무 연산도 진행하지 않고 다음 연산 진행해. 컴파일은 또 되는데, 그 이유는, 중괄호는 쓰든가 말든가 있어도 되고 없어도 되는 거라서. 그냥 소괄호처럼. 그럼 조건에 상관없이, 바로 중괄호 안의 연산을 진행하게 되겠지.

\*/

/switch 구문

switch 넣고, 이어서 소괄호 안에 내가 알고 싶은 녀석의 식별자를 넣어 줌.

switch (str) 이제부터 str에 대해서 swtich 구문 쓸게! 하는 뜻임.

그 식별자의 값이 이런 경우일 때, 저런 경우일 때, 그런 경우일 때, 경우를 나눠서 진행할 연산을 각각 따로 서술해 줌. 한 경우에 대해 서술이 끝나면, 서술 끝나고 다음 경우 설명 시작한다는 뜻으로 break;를 붙여 줌. 그리고 각 경우들 간의 우선순위는 동급임. 위로 올려서 먼저 나온다고 우선순위가 높아지지는 않는다능. 그리고 모든 경우들은 중 괄호로 묶어서 switch에 달아 줌. 예시는 다음과 같음.

switch (str)

{

case “메롱” :

System.out.println(“약오르지 까꿍”);

break;

case “두껍아” :

System.out.println(“헌집 줄게 새집 다오”);

break;

case "피카츄“ :

System.out.println(“라이츄 파이리 꼬부기”);

break;

}

이런 식으로 나열. 마지막에 중괄호 닫아주고.

//참고로 swtich 안에 수식으로 넣어도 됨. switch (yy % 12) 이런 식으로.

181031

/for 구문에서 사용할 수 있는 명령어

break; - 반복문 밖으로 나가버리기

continue; - 해당 회에 한해서, continue 이후의 연산은 생략하고, 다음 반복 횟수부터 재개

이 'continue' 명령어는 응용 방식이 매우 다양해.

for 구문 진행 중에, 이런 조건일 때 이렇게 하고, 저런 조건일 땐 저렇게 하고,

그런 식으로 경우를 나눠서 실행하고 싶을 때, 중요.

continue 는, for 구문을 끝내지 않으면서, 반복문을 수행하는 중에 해당 횟수에 한해서는 continue 이후의 연산은 무시하고(건너 뛰고), 다음 반복 횟수부터 그대로 다시 진행. 그렇기 때문에, if와 함께 응용해서 하자면 응용 범위가 많음. 쓸모가 많음.

continue 응용 예시)

//정수 입력을 다섯 번 받을 건데, 받는 중에 음수가 나오면 해당 횟수는 무효로 치고

다시 받아서, 어쨌든 다섯 개의 정수는 모두 받아내기. 그리고 그 총합을 구하기.

for ( ; i <= 5;)

{

System.out.print(i + "번 째 정수 입력 : " );

n = sc. nextInt();

if(n < 0)

{

continue;

}

i = i + 1;

s = s + n;

}

/\*

사고 방식이, ‘n1, n2, n3.... 모두 받아서 다 더한다!’

이런 것에만 익숙해져 있는데, 우리가 적어도 for 구문 쓸 때는,

‘매 회마다 반복하는 형태로는 어떻게 나타낼 수 있을까’

하는 사고 방식으로 생각해 봐야 함.

\*/

/\*

증감식을 비워놓고, for 구문의 실행문 안으로 넣어도 됨.

조건문을 만족할 때 실행되는 연산 안에 포함해 놓고, 증감식 자리는 비워 놔도 됨.

그럴 경우엔, for 구문 실행 중에 특정 조건을 만났을 때만 횟수를 인정하고

특정 조건일 때는 횟수를 인정해주지 않는(횟수에 포함해주지 않는) 코딩을 짜기가 쉽지

\*/

/null 문장이라고, 그냥

; 이렇게 땀 한 번 흘려주면 공백으로 인식.