variable (std::cout - cout은 std 내의 하위 객체다)
/ 연산자의 int 연산은 몫을 의미한다. (나머지 = 나뉜 수 - 몫 x 나누는 수 -> 음수 나누기에서 몫은 아는데 나머지가 헷갈릴 때 다음으로 판단)
식별어 - 변수 이름. 알파벳 대소문자, _ 숫자만 사용 가능. 예약어 사용 불가. 처음부터 숫자도 불가.
1byte = 8bit, int (32bit signed), short int (16s), long int (32s), long long int (64s),
float (32bit signed), double (64s), long double (96s or 128s),
unsigned - 범위가 0~, const - 불변, float-point_type - 가수e지수 = 가수 * 10 ^ 지수
0x~~(~~은 16진수임을 표현.), 0~~(8진수), 0b~~(2진수)
std::hex, std::dec, std::oct, std::bitset 출력하기 전에 써줘서 출력되는 진수 형태를 원하는대로 바꾸자.
진수 연산해보기 - 0x101.01 = 1*16^2+0*16^1+1*16^0+0*16^(-1)+1*16^(-2)
char - 8bit unsigned, int + char = int, '₩n'(줄바꿈), '₩b'(백스페이스), '₩a'(알람), '₩t'(탭), '₩0'(null) ,'₩₩',
enumerated (class) definition - enum (class) enum_name { elements_list&assignment };
enumerated calss object - enum-class_name object_name = enum-class_name::element_name;
expression
연산자(operator)와 피연산자(operand), 우선순위(precedence)와 결합성(associativity)
% 연산자는 int 연산만 가능
casting - static_cast<new_type>(var) , 주석 - //, /*~*/
overflow/underflow - 유카리 스키마
signed 시스템 - 2진수의 1의 보수(2진수 기준 0과 1이 뒤바뀜), 2의 보수(2진수 기준 1의 보수 + 1)(10진수 기준 부호가 바뀜), 2진수 기준 최상위 자릿수가 0이면 10진수 기준 이 수가 양수임을 확인 가능.(최상위 자릿수는 부호 표현.), 따라서 n bit 기준 (-2^(n-1)~2^(n-1)-1) 범위 구현 가능
부동 소수점 형태 - 부호 * 0 or 1 * 10 ^ 지수 (2진수 기준) = 부호 * 가수 * 2 ^ 지수 (10진수 기준) (암튼 계산하면 오차가 존나 난다는 것만 기억하자.)
increment/decrement ++x(prefix), x++(postfix)
assignmet operator - +=, -=, ~~, &=, |=, ~~
bitwise operator - &, |, ^(xor), ~a(0,1 뒤바꿈), a>>n(2진수 기준 오른쪽으로 n만큼 자릿수 이동), <<
conditional expression
if (condition(activate_if_true)) { contents } else if (condition(activate_if_true)) {contents} else { contents }
switch (정수형 수식) {
case 수식의 결과로 가능한 값:
    contents
    break; (밑에 다 씹고 탈주. 만약 안쓰면 나머지 case 조건도 전부 확인하겠지.)(fallthrough))
default: (모든 case에 부합하지 않으면 실행. 어짜피 밑에 코드도 없으니 break 안씀.)
    contents
type bool - T면 1(0 제외 정수), F면 0을 할당. (정상적인 코드 통과(ex 할당)은 T)
트루펄스로 출력하고 싶으면 std::cout << std::boolalpha << bool_vari;
relational operator - <, >, <=, >= // ==, != (우선순위로 자름)
소수점 계산 결과 == 소수점 계산 결과 -> 미세한 오차로 인해 실제 계산이 같아도 여기선 F. 따라서 다음과 같이 쓰는 것이 좋다; 결과1 - 결과2 < 1E-6
정밀도 함수 - #include <iomanip> std::cout << std::setprecition(유효숫자 개수) << d1; (참고로 디폴트 유효숫자 개수는 6개다.)
logical operator - !, &&, ||, (우선순위대로)
dangle else - else는 중괄호 구분 없이 쓰면 가장 가까운 if와 if else 관계를 맺는다.
conditional operator - (condition) ? (return_if_true) : (return_if_false)
nested & multi-way
repetition
while (condition(loop_if_true)) { contents }
do { contents } while (condition(loop_if_true))
for (initialization;condition(loop_if_true);modification) { contents } (위 세 조건은 null_space를 통해 생략 가능하며, contents에서 따로 써줄 수도 있다.) (modification은 a += 2 같은걸로 표현)
TF_of_cin - F가 되는 경우는 입력받을 변수의 타입과 맞지 않는 입력시와 eof 커맨드(ctrl + z) 입력시.
std::cin.eof() (eof 상태인가?), std::cin.fail() (잘못 입력받은 상태인가?) std::cin.good() (정상 상태인가?)
std::cin.clear() (false -> true) #include <limits>
std::cin.ignore(std::numeri_limits<std::streamsize>::max(), '₩n') (잘못된 입력값 삭제)
출력 필드 너비 설정 함수 - #include <iomanip> std::cout << std::setfill(char) << std::setw(width) << std::left << var << std::endl;
abnormal_loop_termination - break (바로 탈출), label: contents & goto label; (label 영역의 코드로),
contiune (밑에 남은 반복문 내용 전부 생략하고 다음 반복으로) (남은 밑부분을 else로 묶어서 대체 가능),
, 연산자 - 뒤의 피연산자를 반환함. =보다도 우선순위 낮음.
최대 최소 - #include <limits> std::numeric_limits<double>::max()

function
definition - return_type function_name(parameter_declaration) { contexts }
call - function_name(argument_name);
prototype_declaration - return_type function_name(parameter_declaration);
표준함수 - c++에서 기본적으로 제공하는 함수
종류(<cmath>) - sqrt(double-type_argument) (제곱근), pow(x,y) (x^y), fabs(x) (절댓값), sin(x) (라디안 기준), asin(x), atan(x) (플마반파이), acos(x) (0~파이), atan2(x,y) (x,y 좌표 벡터의 각도), log(x) (lnx),
종류(<algorithm>) - std::max(x,y), std::min(x,y),
종류(<cctype>) - toupper(char/int-type_argument) (대문자로) (return을 int로 바꿔주고 싶으면 prototype_declaration으로 return_type을 설정해주자.), tolower(a) (소문자로), isupper(a), islower(a), isalpha(a) (알파벳인가?), isdigit(a) ((문자형일때) 숫자인가?)
무작위수 관련 함수 - srand(static_cast<unsigned>(time(0))); random_num = rand() % range;
pass_by_value - local_var vs global_var (로컬과 글로벌의 이름이 x로 같으면 ::x가 글로벌을 의미한다.)
static_vari - 함수 안에서 사용하면, 함수가 호출되고 나서 반환 이후 버려지는 변수들이 다음 호출 때에도 유지된다. (초기화 문장도 무시한다.)
overload - 같은 이름, 다른 파라미터
default_argument - 파라미터는 뒤에서부터 차례대로 값을 미리 할당할 수 있다. (물론 인수 박을때 새로 넣어줘도 된다.) 이것은 입력에 다양한 가능성을 주므로 오버로드 상태로 함수를 만들어두었으면 방해가 될 수가 있다.
recursion - 함수의 내용이 함수 자신을 호출하는걸 포함하고 있는 상황.
pointer_definition - object_type* pointer_name = nullptr; (nullptr은 포인터 object 無し를 의미)
object_of_pointer == *pointer
reference_definition - object_type& reference_name;
pointer_of_object == &object
reinterpret_cast<new_type>(var) (포인터를 정수형이나 다른 타입을 가리키는 포인터로 바꿀때 사용)
인수에 영향 안주는 함수 - 실체 있는 파라미터에 인수 할당하고 파라미터로 붕가붕가하는 구조
인수에 영향 주는 함수 - 1. 파라미터가 레퍼런스라서 실체를 인수로 할때 (pass by reference) 2. 파라미터가 포인터라서 인수의 주소를 공유할 때 (pass by address)
higher-order_function - 함수를 파라미터로 받거나 리턴하는 함수
function_pointer - 인수로 쓰인 (함수의 return) 자체에도 타격을 주고 싶으면 쓰면됨. 하지만 진짜 주소가 있는건 아니라서 정의 및 호출할 때만 별 붙이고 나머지는 일반 함수처럼 쓰면됨. ((f*) (x, y))
vector
std::vector<element_type> vector_name (vector_size, elements' value);
std::vector<element_type> vector_name {element's value, element's value, ...};
index는 정수형이라 소수 들어가면 짜르고 정수만. 값이 있는 index 사이 빈 index에 대해서는 안배움.
vector method - vector_name.push_back(element_value);
종류 - pop_back(element_value), operator[](index), at(index), size(), clear(), empty()
생성자 초기화 리스트 - for (ele_var : vec_var) { contents } //벡터의 엘리먼트를 차례대로 하나씩 받는다.
다차원 벡터 - std::vector<std::vector<int>> matrix(size, std::vector<int>(size)); (matrix[행][열])

- [unary] (post)++, (post)--, static_cast
- [unary] (pre)++, (pre)--, !, +, -
- [binary, left associativity] *, /, %
- [binary, left associativity] +, -
- [binary, left associativity] <<, >>
- [binary, left associativity] >, <, >=, <=
- [binary, left associativity] ==, !=
- [binary, left associativity] &&
- [binary, left associativity] ||
- [binary, right associativity] =, +=, -=, *=, /=, %=

```
        }
        if (a < b) {
            return -1;
        }
        if (a == b) {
            return 0;
        }
    }
    void Ex2(int a, int b, int& c, int* d) {
        for (int i = std::min(a, b); i <= std::max(a, b); ++i) {
            if (i % 2 == 0) {
                c += i;
            }
            if (i % 2 == 1) {
                *d += i;
            }
        }
    }
    double Ex3(double a) {//테일러 급수로 코사인 구현
        double sign = 1, upper = 1, lower = 1, plus = 0, cosine = 1;
        for (int a = 1;; a++) {
            if (fabs(plus) <= 1.E-15) {
                break;
            }
            upper *= (a * a);
            for (int b = 1; b <= 2 * a; b++) {
                lower *= b;
            }
            sign *= (-1);
            plus = sign * upper / lower;
            cosine += plus;
        }
        return cosine;
    }
    int Ex4(int a, int b, int c = 0) {//비트와이즈 연산자로 16진수 자리 옮기기
        double d = 0;
        if (c == 0) {
            d = (a << 8) + b;
        }
        if (c == 1) {
            d = a + (b << 8);
        }
        return d;
    }
    std::vector<int> Ex5(std::vector<int> v) { //벡터 한자리수만 뽑기
        std::vector<int> r;
        for (int i = 0; i < v.size(); i++) {
            if (v[i] > 0) {
                r.push_back(v[i] % 10);
            }
        }
        return r;
    }
    std::vector<std::vector<int>> Ex6(std::vector<std::vector<int>>& v) { //벡터에 -1 곱
        for (int i = 0; i < v.size(); i++) {
            for (int j = 0; j < v[i].size(); j++) {
                v[i][j] *= (-1);
            }
        }
        return v;
    }
    std::vector<int> Ex7(std::vector<std::vector<int>> v) { //벡터 뒤집기
        std::vector<int> r;
        for (int i = 0; i < v.size(); i++) {
            int max, min = std::numeric_limits<int>::min();
            for (int j = 0; j < v[i].size(); j++) {
                if (v[i][j] > min) {
                    max = v[i][j];
                }
            }
            r.push_back(max);
        }
        return r;
    }
    void Ex8(int a) {//16진수 표현 (test recursion)
        int b = a;
        int num = 1;
        while (b > 16) {
            b /= 16;
            num *= 16;
        }
        if (b <= 9) {
            std::cout << b;
            if (a >= 16) {
                Ex8(a - b * num);
            }
        }
        if (b > 9) {
            std::cout << static_cast<char>('A' + (b - 10));
            if (a >= 16) {
                Ex8(a - b * num);
            }
        }
    }
```

(Ctrl)