빨간색 문제들은 배점사항이 큰것들.

3. 함수의 return value data type의 종류와 parameter의 종류를 prototype형태로 기술하라. ex) void func(void)

```
int add(int a, int b);

double divide(double numerator, double denominator);

char getGrade(int score);

char* concatenateStrings(const char* str1, const char* str2);

struct Person* createPerson(const char* name, int age);

int* createIntArray(int size);

int* findValueInArray(int* array, int size, const char* target);

void printSum(int a, int b);

void initializeData(void);

int* addPointers(int* ptr1, int* ptr2);

struct Point* findClosestPoint(struct Point* points, int numPoints);

int (*getMaxInMatrix(int matrix[], int rows, int cols))[cols];

struct Complex complexAdd(struct Complex a, struct Complex b);

int** createMatrix(int rows, int cols);
```

4. 재귀함수에 대해 기술하시오.

stack필요하니 그런 예제 그림 있으면 좋을 듯.

재귀 함수란 함수 내에서 자기 자신을 호출하는 함수를 뜻한다.

그럼 재귀함수는 어떠한 작업을 하나?

재귀함수는 자기자신을 호출하면서 반복을 진행하는 작업을 한다. 재귀함수에는 이러한 반복을 끝내기 위한 종결문과 자신을 호출하는 셀프 호출 이 두 가지는 꼭 필요하다.

그럼 반복을 하는 것이라면 반복문을 쓰면 되는 것이 아닌가? 맞는말이다. 재귀함수는 for문과 while문 등 반복문으로 대체 가능하다. 하지만 반복문보다 재귀함수가 더 효율적인 종류의 문제들이 존재한다. 예를 들면 여러 단계를 포함할 수 있는 데이터를 다루는 문제, 정렬 알고리즘, 피보나치, 타워 오브 하노이 등이 있다.

왜 위 문제들은 재귀함수가 적합할까? 위 문제들을 해결하기 위해선 하나의 큰 복잡한 문제를 비슷한 작은 여러 문제로 나누어 해결하는 것이 더 효율적이기에 재귀함수가 더 효율적이고 간결한 코드가 된다.

그럼 다른 문제에서는 재귀함수를 쓰는건 어떠한가? 재귀함수는 경우에 따라서는 아주 간결하고 직관적인 코드로 문제를 해결할 수 있게 해주지만 그렇지 않은 상황에서는 심각한 비효율성을 낳을 수 있기 때문에 알고리즘 작성시 유의해야한다.

또한 재귀함수는 이해하기 어렵다. 보통 우리가 접하는 함수들은 시작과 끝이 한 세트로 존재한다. 하지만 재귀함수는 시작 시작 시작 시작 종료 종료 종료 종료 이런식의 실행을 하게되어 맨 처음 시작했던 것이 제일 마지막에 종료가 되어 해석하는 도중 복잡하다고 느껴질 수 있다. 그리고 재귀함수는 호출될 때마다 메모리에 스택이 쌓이게 된다. 이를 한계치 이상으로 호출돼서 스택이 넘쳐버리면 메모리 부족으로 에러가 발생한다. 단점은 또 있다 속도면에서 이다. 재귀함수는 jump가 잦기 때문에 반복문에 비해 시간을 더 소모한다.

하지만 위 같은 문제점을 해결하는 꼬리 재귀 최적화라는 것이 존재한다. 꼬리 재귀는 재귀함수를 컴퓨터가 재해석해서 선형 알고리즘으로 만들어 실행하는 것이다. 이렇게 되면 일이 아무리 많이 발생해도 스택이 넘치는 일은 일어나지 않는다.

하지만 꼬리재귀는 조건이 있다 재귀함수의 return값에 자기 자신함수만 호출되어야한다.

이러한 단점이 있는 재귀함수는 어떻게 동작할까?

쉽게 말하면 스택 내에 차곡 차곡 쌓아두었다가 일괄로 처리한다. 아래의 예제를 이용해서 그림으로 설명해보겠다.

5. 함수 포인터에 대해 기술하시오. 프로토타입이 모든 동일한 함수는 하나의 이름으로 처리 가능

먼저 포인터란 변수의 주소를 저장하는 변수이다. 함수 포인터는 변수의 주소가 아닌 함수의 주소를 저장하는 변수이다. 함수 포인터는 리턴타입 (*포인터명)(인수타입); 으로 선언 후
포인터명 = 함수명;으로 주소를 저장시켜 주면 된다. 포인터명은 반드시 ( )로 감싸 주어야하며 위처럼 주소를 포인터에 저장할 때 함수명만 쓰고 인자는 쓰지않았다. 이유는 함수명에 있다. 배열 포인터는 배열의 이름이 시작 주소값이 된다. 이처럼 함수도 함수명자체에 함수의 주소값이 저장되어있다. 이렇게 함수 포인터를 선언하고 나면 똑같은 주소를 가진 함수처럼 인자를 전달 받을수 있고 함수 호출도 가능해진다.
예제 1번 적고
함수포인터의 활용을 한번 보겠다. 예제2 적은다음에
showif(a,10,isodd)에서 isodd는 함수를 호출하는 것이 아닌 함수의 주소를 전달하는 것이다.
이 코드는 조건에 맞게 출력하게 만들어주는 코드이다
위처럼 코드를 짜면 무엇이 좋을까?
개발자들 마다 원하는 조건이 다를수도 있다. 그래서 조건을 함수내에 만드는 것이 아닌
조건에 해당하는 조건부 함수를 미리 만들어 놓고 해당되는 함수의 주소를 가르킬수 있게 만들어 준다면 어떤 조건이든지 형태만 맞게 인자를 전달해주면 조건에 맞는 출력을 하게 해줌.

6. C언어의 메모리 구조에 대해 기술하시오.
Ram이라는 메모리에 각각의 프로그램들이 실행될 때 2진수로 적재가 된다.
RAM은 코드영역, 데이터영역, 힙영역, 스택영역 총 네가지로 나뉜다.
코드 영역에는 c언어가 컴파일된 정보가 저장된다. CPU는 코드 영역에 저장된 명령어를 하나씩 가져가서 처리하게 됩니다.
데이터 영역에는 전역변수, static변수, 문자열 등 저장된다.
컴파일시 함께 할당되며 프로그램이 종료되면 소멸한다.
힙 영역은 동적 할당영역이다. 프로그램을 작성하다가 정해진 메모리 말고 일시적으로 썼다가 반납하는 메모리를 뜻한다.
스택 영역은 지역변수, 함수의 파라미터 등을 저장한다.
함수의 호출과 함께 할당되며 호출이 완료되면 소멸한다. 스택은 메모리의 높은 주소에서 낮은 주소의 방향으로 할당되며 늦게 들어온 순서부터 먼저 출력되는 후입선출 방식이며 데이터 저장은 push, 데이터 인출은 pop으로 진행된다.

7. 메모리 동적 할당 관련 함수를 간략히 설명하라. 뭔지만 간단히 설명
동적 메모리 할당이란 프로그램이 실행 도중에 동적으로 메모리를 할당 받는 것을 말한다.
프로그램에서는 필요한 만큼의 메모리를 시스템으로부터 할당을 받아서 사용하고, 사용이 끝나면 시스템에 메모리를 반납한다.
필요한 만큼만 할당을 받고 또 필요한 때에 사용하고 반납해 메모리를 효율적으로 사용할 수 있는 것임
동적 메모리는 malloc( ) 계열의 라이브러리 함수를 사용하여 할당받아 사용 가능.
동적 메모리의 사용이 끝나면 반드시 해당 메모리 영역을 명시적으로 반납해주어야함.

9.(20점) 인의예지신과 중도에 대해 기술하라.
인간이지켜야할 다섯가지 기본 덕목 인, 의, 예, 지, 신
인간은 어질고 의롭고 예의 있고 지혜로우며 믿음이 있어야 한다 라는것이지만 너무 어느 한쪽으로 치우쳐져있으면 안된다라는 뜻.