Hw09

소프트웨어융합학과 2019102106 예다정

1.

a. recursive로 구현한 함수가 종료될 조건을 말한다.

b. recursive로 구현한 함수 내부에서 자기 자신 함수를 다시 호출하게 되는 조건을 말한다.

c. 함수가 호출될 때 컴파일러는 함수의 Code영역과 Data영역을 분리하는데, 이 때 Data영역은 runtime에 stack으로 관리된다. (data: 함수의 parameter, local variables, return address, return value)

d. 프로그램 내의 변수 이름과 메모리 상의 주솟값을 binding하는 시간을 말한다. (compile time, run time)

e. 함수 구현부에서 recursive call을 한 번만 포함하고 있고 recursive call이 함수 안의 statement들보다 제일 뒤에 있을 때를 tail recursion이라고 한다.

2.

a. Base-Case Question: list.info[startIndex]가 찾고자 하는 value와 같은 값인 경우, 혹은 startIndex가 list.length – 1인 경우 recursive로 함수를 다시 호출하지 않고 빠져나온다.

b. Smaller-Caller Question: 다음 번에 함수를 호출할 때 list에서 찾기 시작할 startIndex를 1씩 늘려서 호출한다. 따라서 list안에서 value를 찾거나 startIndex가 list의 마지막 인덱스에 도달하는 base case로 수렴하게 된다.

c. General-Case Question: base case에 해당하지 않는 general case의 경우 함수의 startIndex 파라미터에 현재 startIndex에서 1을 더한 startIndex + 1을 넘겨주고 함수를 호출하고 있기 때문에 함수가 모두 정상적으로 작동한다고 할 수 있다.

3.

a. Fun함수의 입력값으로 음수가 전달될 경우

b. base case인 num == 0 에 도달하지 않아 무한루프에 빠지게 된다.

c. 0이 return된다.

d. -15가 return된다.

4.

a. 함수가 최초로 실행되자마자 return문을 만나 종료된다. 따라서 base case를 추가해 base case인 경우 0을 return하도록 해야한다.

b. list->info를 제곱한 값을 sum에 더하고있지 않다. 따라서 sum += list->info \* list->info 로 수정해야 한다.

c. 틀린 곳 x

d. 틀린 곳 x

e. 힘수가 실행되면 list가 null이 될 때까지 SumSquares가 호출됐다가 null인 노드에서 0을 반환하면서 계속해서 return값에 0을 곱하게 된다. 따라서 list != NULL인 경우(즉, general case)의 동작을 수정해야한다.

5.

a.

int NumPaths(int row, int col, int n) {

if (row == n)

return 1;

else if (col == n)

return 1;

else

return NumPaths(row + 1, col, n) + NumPaths(row, col + 1, n);

}

b. 중복되는 계산(입력값이 동일한 함수 호출)을 여러 번 하게 된다.

c.

const int MAX\_ROWS = 10;

const int MAX\_COLS = 10;

int path[MAX\_ROWS][MAX\_COLS];

//path 2차원 행렬의 모든 원소 -1로 초기화

int NumPaths(int row , int col, int n) {

if (path[row][col] == -1) {

if (row == n || col == n) {

path[row][col] = 1;

}

else {

path[row][col] = NumPaths(row + 1, col, n) + NumPaths(row, col + 1, n);

}

}

return path[row][col];

}

d.

int main() {

for (int row = 0; row < MAX\_ROWS; row++) {

for (int col = 0; col < MAX\_COLS; col++)

path[row][col] = -1;

}

cout << NumPaths(1, 1, 4) << endl;

}

e.

두 번째 방법으로 구현했을 때 같은 입력값을 갖는 함수 호출을 중복해서 하지 않기 때문에 그만큼 시간과 메모리를 절약할 수 있다. 따라서 두 번째로 구현 시 time efficiency와 space efficiency가 더 좋다

6.

