# 01. 팩토리얼을 계산하는 순환호출 함수 factorial에서 매개 변수로 5를 주었다면 최대 몇 개의 factorial 함수의 활성 레코드가 동시에 존재할 수 있는가?

5개  
factorial(5)이 factorial(4)를 호출하고, factorial(4)이 factorial(3)를, …. 이런 식으로 factorial(1)까지 호출이 되면서 동시에 factorial 함수 5개가 동시에 존재한다.

# 02. 순환호출을 하였을 경우에 활성 레코드들이 저장되는 위치는 어디인가? (1) 순환호출 함수내부 (2) 변수 (3) 배열 (4) 스택

(4)  
순환호출을 하였을 스택에서 공간을 할당 받아 활성 레코드들을 저장한다.

# 03. 다음 중 활성 레코드에 저장되지 않는 것은 무엇인가? (1) 매개변수의 값 (2) 함수호출이 끝나고 복귀할 주소 (3) 지역변수 (4) 순환호출의 순차번호

(4)  
함수호출이 끝나고 복귀할 주소와 지역변수가 있기에 순환함수에 순차적으로 복귀한다.

# 04. 하나의 함수가 호출할 수 있는 순환호출의 개수는? (1) 1번 (2) 2번 (3) 스택이 허용하는 한도 (4) 무제한

(3)  
스택이 허용하는 한도를 넘어서는 만큼 순환호출을 하게 될 시, 에러코드가 뜨게 된다.

# 05. 다음의 순환호출 함수에서 잘못된 점은 무엇인가? int recursive(int n) { if (n == 1) return 0; return n \* recursive(n); }

마지막 줄은 프로그램을 무한 루프에 빠지게 하므로 다음 순환호출을 할 때 n값을 변경해서 최종적으로 n == 1이 되도록 순환호출 함수를 변경해야 된다.

# 06. 다음의 순환호출 함수에서 잘못된 점은 무엇인가? int recursive(int n) { printf(“recursive(%d)\n”, n); return n \* recursive(n-1); }

매개변수 값의 최소값이 설정되어 있지 않아 n값이 무한대로 작아지면서 순환호출이 무한대로 된다. printf() 앞 혹은 뒤에 스택의 한도 내에 n값이 얼마까지 줄어들 수 있는지 설정해야 한다.

# 07. 다음 함수를 sum(5)로 호출하였을 때, 화면에 출력되는 내용과 함수의 반환값을 구하라. int sum(int n) { printf(“%d\n”, n); if (n < 1) return 1; else return (n + sum(n-1)); }

5  
4  
3  
2  
1  
0

반환값: 16  
sum(5) = 5 + sum(4) = 5 + (4 + sum(3)) = 5 + (4 + (3 + sum(2))) = 5 + (4 + (3 + (2 + sum(1)))) = 5 + (4 + (3 + (2 + (1 + sum(0))))) = 5 + (4 + (3 + (2 + (1 + 1)))) = 16

# 08. 다음 함수를 recursive(5)로 호출하였을 때, 화면에 출력되는 내용과 함수의 반환값을 구하라. int recursive(int n) { printf(“%d\n”, n); if (n < 1) return 2; else return (2 \* recursive(n-1) + 1); }

5  
4  
3  
2  
1  
0

반환값: 95  
recursive(5) = 2 \* recursive(4) + 1 = 2 \* (2 \* recursive(3) + 1) + 1 = 2 \* (2 \* (2 \* recursive(2) + 1) + 1) + 1 = 2 \* (2 \* (2 \* (2 \* recursive(1) + 1) + 1) + 1) + 1 = 2 \* (2 \* (2 \* (2 \* (2 \* recursive(0) + 1) + 1) + 1) + 1) + 1 = 2 \* (2 \* (2 \* (2 \* (2 \* 2 + 1) + 1) + 1) + 1) + 1 = 95

# 09. 다음 함수를 recursive(10)로 호출하였을 때, 화면에 출력되는 내용과 함수의 반환값을 구하라. int recursive(int n) { printf(“%d\n”, n); if (n < 1) return -1; else return (recursive(n-3) + 1); }

10  
7  
4  
1  
-2

반환값: 3  
recursive(10) = recursive(7) + 1 = (recursive(4) + 1) + 1 = ((recursive(1) + 1) + 1) + 1 =  
(((recursive(-2) + 1) + 1) + 1) + 1 = (-1 + 1) + 1) + 1) + 1 = 3

# 10. 다음 함수를 recursive(5)로 호출하였을 때, 화면에 출력되는 내용을 쓰시요. int recursive(int n) { if (n != 1) recursive(n-1); printf(“%d\n”, n); }

1  
2  
3  
4  
5

# 11. 다음 함수에서 asterisk(5)를 호출할 때 출력되는 \*의 개수는? void asterisk(int i) { if (i > 1) { asterisk(i/2); asterisk(i/2); } printf(“\*”); }

7개  
i == 5: 1번  
 == 2: 2번  
 == 1: 4번

# 12. 다음과 같은 함수를 호출하고 “recursive” 문자열을 입력한 다음, 엔터키를 눌렀다면 화면에 출력되는 것은? unknown() { int ch; if ((ch = getchar()) != ‘\n’) { unknown(); } putchar(ch); }

evisrucer  
getchar()은 입력된 char을 int로 변형시켜주고 putchar()은 역으로 int형을 char로 바꾼다. 마지막 순환 때 ch에 ‘\n’이 저장되어 있기 때문에 한 줄 띄워야 할 것 같다..?

# 13. 다음을 계산하는 순환적인 프로그램을 작성하시오. 1 + 2 + 3 + … + n

int recursive(int n) {  
 if (n == 1) return 1;  
 return (n + recursive(n-1));

# 14. 다음을 계산하는 순환적인 프로그램을 작성하시오. 1 + 1/2 + 1/3 + … + 1/n

double recursive(int n) {  
 if (n ==1) return 1;  
 return ((double) 1 / n + recursive(n-1));

# 15. 순환호출되는 것을 이해하기 위하여 fib 함수를 다음과 같이 바꾸어서 실행하여 보라. fib(6)을 호출할 때 화면에 출력되는 내용을 쓰시오. int fib(int n) { printf(“fib(%d) is called\n”, n); if (n == 0) return 0; if (n == 1) return 1; return (fib(n-1) + fib(n-2)); }

fib(6) is called  
fib(5) is called // fib(6-1) (1)  
fib(4) is called // fib(5-1) (1\_1)  
fib(3) is called // fib(4-1) (1\_1\_1)  
fib(2) is called // fib(3-1) (1\_1\_1\_1)  
fib(1) is called // fib(2-1) (1\_1\_1\_1\_1)  
fib(0) is called // fib(2-2) (1\_1\_1\_1\_2)  
fib(1) is called // fib(3-2) (1\_1\_1\_2)  
fib(2) is called // fib(4-2) (1\_1\_2)  
fib(1) is called // fib(2-1) (1\_1\_2\_1)  
fib(0) is called // fib(2-2) (1\_1\_2\_2)  
fib(3) is called // fib(5-2) (1\_2)  
fib(2) is called // fib(3-1) (1\_2\_1)  
fib(1) is called // fib(2-1) (1\_2\_1\_1)  
fib(0) is called // fib(2-2) (1\_2\_1\_2)  
fib(1) is called // fib(3-2) (1\_2\_2)  
fib(4) is called // fib(6-2) (2)  
fib(3) is called // fib(4-1) (2\_1)  
fib(2) is called // fib(3-1) (2\_1\_1)  
fib(1) is called // fib(2-1) (2\_1\_1\_1)  
fib(0) is called // fib(2-2) (2\_1\_1\_2)  
fib(1) is called // fib(3-2) (2\_1\_2)  
fib(2) is called // fib(4-2) (2\_2)  
fib(1) is called // fib(2-1) (2\_2\_1)  
fib(0) is called // fib(1-2) (2\_2\_2)

# 16. 다음의 순환적인 프로그램을 반복 구조를 사용한 비순환적 프로그램으로 바꾸시오. int sum(int n) { if (n == 1) return 1; else return (n + sum(n-1)); }

int sum(int n) {  
 int sum = 0;  
 for (int i = 1; i ≤ n; i++) {  
 sum += i;  
 }  
 return sum;  
}

# 17. 이항계수 (binomial coefficient)를 계산하는 순환함수를 작성하라. 이항계수는 다음과 같이 순환적으로 정의된다. 반복함수로도 구현해보라.

int recursive(int n, int k) {  
 if ((k == 0) || (k == n)) return 1;  
 else if ((k > 0) && (k < n)) return (recursive(n-1, k-1) + recursive(n-1, k));  
 else return 0;

int iterative(int n, int k) {  
 int sum = 1;  
 for (int i == 1; i ≤ n; i++) {   
 sum \*= i;  
 }  
 for (int i == 1; i ≤ (n-k); i++) {  
 sum /= i;  
 }  
 for (int i == 1; I ≤ k; i++) {  
 sum /= i;  
 }  
 return sum;  
}

# 18. Ackermann 함수는 다음과 같이 순환적으로 정의된다. A(0, n) = n + 1; A(m, 0) = A(m-1, 1) A(m, n) = A(m-1, A(m, n-1)) m, n ≥ 1 (a) A(3, 2)와 A(2, 3)의 값을 구하시요. (b) Ackermann 함수를 구하는 순환적인 프로그램을 작성하시요. (c) 위의 순환적인 프로그램을 for, while, do와 같은 반복구조를 사용한 비순환적 프로그램으로 바꾸시요.

(a) 29, 9  
계산이 복잡한 관계로 (b) 코드에 바로 대입해서 풀었다.

(b)  
int recursive(int m, int n) {  
 if (m == 0) {  
 return (n+1);  
 } else if (n == 0) {  
 return recursive(m-1, 1);  
 } else {  
 return recursive(m-1, recursive(m, n-1));  
 }  
}

(c) memory allocation 내용이 나오므로 이 단원에 맞지 않다.

# 19. 본문의 순환적인 피보나치 수열 프로그램과 반복적인 피보나치 수열 프로그램의 수행 시간을 측정하여 비교하라. 어떤 결론을 내릴 수 있는가?

반복적인 피보나치 수열 프로그램의 시간 복잡도가 순환적인 피보나치 수열 프로그램의 것보다 낮기 때문에 반복적인 프로그램의 수행 시간이 순환적인 프로그램의 것보다 짧다.

# 20. 순환호출에서는 순환호출을 할 때마다 문제의 크기가 작아져야 한다. (1) 팩토리얼 계산 문제에서 순환호출이 일어날 때마다 문제가 어떻게 작아지는가? (2) 하노이의 탑에서 순환호출이 일어날 때마다 문제가 어떻게 작아지는가?

(1) 팩토리얼 값이 순환호출 될 때마다 작아지면서 1!까지 도달하게 된다. 도달하는 과정에서 팩토리얼을 곱셈으로 변환시키기 때문에 순환함수에서 한번에 해결해야 되는 문제가 작아진다.

(2) 고려해야 하는 고리의 수가 순환호출이 될 때마다 줄어들기 때문에 (한번 호출될 때마다 가장 큰 고리가 옮기려는 막대에 옮겨짐으로써 고려대상에서 제외된다) 문제가 작아진다.

# 21. 컴퓨터 그래픽에서의 영역 채우기 알고리즘은 순환 기법을 사용한다. 영역 채우기란 다음과 같은 흰색 영역이 있을 때 이 영역을 특정한 색으로 채우는 것이다. 여기서는 이 영역 안쪽을 검정색으로 채운다고 가정해보라. 이런 경우에는 순환 호출을 어떻게 사용할 수 있을까? 2차원 배열이 다음과 같이 되어 있다고 가정하고 영역안의 한 점의 좌표가 주어졌을 경우에 안쪽을 채우는 순환 호출 함수를 작성하여 보라. [그림 2-16]의 ×로 표시된 픽셀이 시작 픽셀이다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 2 | 2 | 2 | 0 | 0 | 0 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 2 | 2 | 2 | 0 | 0 | 0 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 2 | 2 | 2 |  | 0 | 0 | 0 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 2 | 2 | 2 | 0 | 2 | 2 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 2 | 2 | 2 | 0 | 2 | 2 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 2 | 2 | 2 | 0 | 2 | 2 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

**(a) 영역 채우기 대상 도형 (b) 2차원 배열을 이용하여 (a)를 표현**

**[그림 2-16] 영역 채우기**  
#define WHITE 0  
#define BLACK 1  
#define YELLOW 2  
  
int screen[WIDTH][HEIGHT];  
//  
char read\_pixel(int x, int y) {  
 return screen[x][y];  
}  
//  
void write\_pixel(int x, int y, int color) {  
 screen[x][y] = color;  
}  
// 영역 채우기 알고리즘  
void flood\_fill(int x, int y) {  
 if (read\_pixel(x, y) == WHITE) {  
 write\_pixel(x, y, BLACK);  
 flood\_fill(x+1, y); // 순환호출  
 flood\_fill(x-1, y); // 순환호출  
 flood\_fill(x, y+1); // 순환호출  
 flood\_fill(x, y-1); // 순환호출  
 }  
}