자료구조와실습
연습문제

학번 : 2016110056
학과 : 불교학부
이름 : 박승원
날짜 : 2016년 9월 29일

1. 팩토리얼을 계산하는 순환 호출 함수 factorial에서 매개 변수로 5를 주었다면 최대 몇 개의 factorial함수의 활성 레코드가 동시에 존재할 수 있는가? 5개

2. 순환 호출을 하였을 경우에 활성 레코드들이 저장되는 위치는 어디인가?

   (a) 순환호출 함수 내부

   (b) 변수

   (c) 배열

   (d) 스택

3. 다음 중 활성 레코드에 저장되지 않는 것은 무엇인가?

   (a) 파라미터의 값

   (b) 함수 호출이 끝나고 복귀할 주소

   (c) 지역변수

   (d) 순환호출의 순차번호

4. 다음 중 순환호출이 불가능한 언어는?

   (a) C언어

   (b) Pascal 언어

   (c) Basic 언어

   (d) Java 언어

5. 하나의 함수가 호출할 수 있는 순환 호출의 개수는?

   (a) 1번

   (b) 2번

   (c) 스택이 허용하는 한도

   (d) 무제한

6. 다음의 순환 호출 함수에서 잘못된 점은 무엇인가?

```
int recursive(int n)
{
   if(n==1) return 0;
   return n*recursive(n);
}
```

   n을 n-1로 고쳐야 한다.

7. 다음의 순환 호출 함수에서 잘못된 점은 무엇인가?

```
int recursive(int n)
{
   printf("recursive(%d)\n", n);
   return n*recursive(n-1);
}
```

초항이 없어서 무한루프에 빠진다.

8. 다음 함수를 sum(5)로 호출하였을 때, 화면에 출력되는 내용과 함수의 반환 값을 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

9. 다음 함수를 recursive(5)로 호출하였을 때, 화면에 출력되는 내용과 함수의 반환 값을 구하라.

```
int recursive(int n)
{
   printf("%d\n", n);
   if(n<1) return -1;
   else return (2*recursive(n-1)+1);
}
```

5
4
3
2
1
0
반환값:255

10. 다음 함수를 recursive(10)로 호출하였을 때, 화면에 출력되는 내용과 함수의 반환 값을 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

11. 다음 함수를 recursive(5)로 호출하였을 때, 화면에 출력되는 내용을 쓰시오.

```
int recursive(int n)
{

   if(n != 1) recursive(n-1);
   printf("%d\nLeftarrow", n);
}
```

1
2
3
4
5

12. 다음 함수에서 asterisk(5)를 호출할 때 출력되는 *의 개수는?

```
int asterisk(int i)
{
   if(i>1) {
      asterisk(i/2);
      asterisk(i/2);
   }
   printf("*");
}
```

7개

13. 다음과 같은 함수를 호출하고 "recursive"문자열을 입력한 다음, 엔터키를 눌렀다면 화면에 출력되는 것은?

```
unknown()
{
   int ch;
   if (ch = getchar()) != '\n') unknown();
   putchar(ch);
}
```

evisrucer

14. 다음을 계산하는 순환적인 프로그램을 작성하시오.

1+2+3+...+n

```
int sum(int n)
{
   return n == 1 ? 1 : n + sum(n-1);
}
```

15. 다음을 계산하는 순환적인 프로그램을 작성하시오.

1 + 1/2 + 1/3 + ... + 1/n

```
float sum(float n)
{
   return n == 1 ? 1 : 1/n + sum(n-1);
}
```

16. 순환 호출되는 것을 이해하기 위하여 fib함수를 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(4) is called
fib(3) is called
fib(2) is called
fib(1) is called
fib(0) is called
fib(1) is called
fib(2) is called
fib(1) is called
fib(0) is called
fib(3) is called
fib(2) is called
fib(1) is called
fib(0) is called
fib(1) is called
fib(4) is called
fib(3) is called
fib(2) is called
fib(1) is called
fib(0) is called
fib(1) is called
fib(2) is called
fib(1) is called
fib(0) is called

17. 다음의 순환적인 프로그램을 반복구조를 사용한 비순환적 프로그램으로 바꾸시오.

```
int sum(int n)
```

```
{
   if(n==1) return 1;
   else return n+sum(n-1);
}

// 반복구조로
int sum(int n)
{
   int sum = 0;
   for(int i=1; i<=n; i++) sum += i;
   return sum;
}
```

18. 이항계수(binomial coefficient)를 계산하는 순환 함수를 작성하라. 이항계수는 다음과 같이 순환적으로 정의된다. 반복 함수로도 구현해보라.

$$ {}_nC_k = \begin{cases} {}_{n-1}C_{k-1} +_{n-1} C_k & if\ 0 < k < n \\ 1 & if\ k = 0\ or\ k = n \end{cases} $$

```
int nCr(int n, int r)
{
   if(r==0 || r==n) return 1;
   return nCr(n-1, r-1) + nCr(n-1, r);
}

// 반복함수
int factorial(int n) {
   int r = 1;
   for(int i=1; i<=n; i++) r *= i;
   return r;
}
int nPr(int n, int r) {
   return factorial(n)/factorial(r);
}
int nCr(int n, int r)
{
   return nPr(n, r)/factorial(r);
}
```

19. Ackermann 함수는 다음과 같이 순환적으로 정의된다.

A(0, n) = n+1
A(m, 0) = A(m-1, 1)
$A(m,n) = A(m-1, A(m,n-1)) m,n \geqq 1$

(a) A(3,2)와 A(2,3)의 값을 구하시오. 29, 9

(b) Ackermann 함수를 구하는 순환적인 프로그램을 작성하시오.

```
int A(int m, int n) {
   if(m == 0) return n+1;
```

```
    if(n == 0) return A(m-1, 1);
    return A(m-1, A(m, n-1));
}
```

(c) 위의 순환적인 프로그램을 for, while, do와 같은 반복구조를 사용한 비순환적 프로그램으로 바꾸시오.

```
int Stack[MAX];
int top = -1;
int Push(int value);
int Pop();
int Ackermann(int m, int n);
int main()
{
    int m, n;
    printf("Enter two values : ");
    scanf("%d %d", &m, &n);
    printf("The result is %d.\n", Ackermann(m, n));
    return 0;
}

int Push(int value)
{
    if ( top >= MAX - 1 ) {
        printf("Stack overflow\n");
        return -1;
    }
    Stack[++top] = value;
    return value;
}

int Pop()
{
    if ( top < 0 ) {
        return -1;
    }
    return Stack[top--];
}

int Ackermann(int m, int n)
{
    while ( 1 ) {
        if ( 0 == m ) {
            ++n;
            m = Pop();
            if ( m == -1 ) return n;
        } else if ( 0 == n ) {
            --m;
            ++n;
        } else {
```

```
            if ( Push(m-1) == -1 ) exit(1);
            --n;
        }
    }
    return -1;
}
```

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

    반복적인 프로그램이 더 빠르다. 재귀적인 프로그램에서는 메모리의 할당, 함수의 호출 등의 작업이 일어나기 때문으로 보인다.

21. 순환 호출에서는 순환 호출을 할때마다 문제의 크기가 작아져야 한다.

    (a) 팩토리얼 계산 문제에서 순환 호출이 일어날 때마다 문제가 어떻게 작아지는가?
    계산하고자 하는 입력값이 1씩 작아진다.

    (b) 하노이의 탑에서 순환 호출이 일어날 때마다 문제가 어떻게 작아지는가?
    옮기는 더미가 하나씩 줄어든다.

소감

순환구조를 반복구조로 바꾸는 것이 함수만 주고 구하라니 굉장히 어려웠다. 수열의 진행을 하나하나 조사해야 되었다. 만약 재귀함수가 없었다면 얼마나 불편했을까하고 생각해보게 되었다. 에커만 함수는 결국 인터넷을 참조하는 수밖에 없었다. 그런데, 이상한 것이 인터넷의 프로그램도 스택을 이용하여 계산을 하는 것이라, 반복문이라고 보기에는 이상했다. 더 이상 에커만 함수를 연구하는 것이 생산적이지는 않을 것 같아 더 연구하지는 않았다. 프로그래머의 입장에서 프로그래밍에서 가장 중요한 것은 반복되는 가장 작은 단위를 찾는 것인데, 재귀함수는 이런 점에 가장 부합한다. 재귀함수를 잘 활용하는 것이 매우 중요하다고 생각한다.

그리고, 이책에서는 recursive를 순환적인 프로그램으로 해석했다. 다른 책에서는 대부분 재귀적인 프로그램으로 해석하는데, 여기서는 용어가 달라 헷갈린다. 순환과 반복이란 용어가 거의 비슷한 뜻이니 재귀라는 단어를 쓰면 좋겠다.