8. ptr을 루트 노드로 하는 이진검색 트리에서 key 값을 갖는 노드의 포인터를 반환하는 함수 struct btnode *search(struct btnode *ptr, int key)를 작성하라. 단, 함수는 recursion으로 구현된다고 가정한다. [15]

```
struct btnode {
    int  data;
    struct btnode *lchild;
    struct btnode *rchild;
};
struct btnode *search(struct btnode *ptr, int key)
{
    if (!root)
        return NULL;
    if (key == root->data)
        return root;
    if (key < root->data)
        return search(root->lchild, key);
    return search(root->rchild, key);
}
```

9. Threaded binary tree에서 ptr이 가리키는 노드의 inorder predecessor를 반환하는 함수 struct tbt *inpred(tbt *ptr)를 작성하라. [15]

```
struct tbt {
    int  data;
    short int left_thread;    short int right_thread;
    struct tbt *lchild;       struct tbt *rchild;
};
struct tbt *inpred(struct tbt *ptr)
{
    struct *temp = ptr->lchild;
    if (ptr->left_thread == 0){
        while(!temp->right_thread)
            temp = temp->rchild;
    }
    return temp;
}
```

10. 다음은 original을 루트로 하는 이진트리를 복사하되, 왼쪽 서브트리와 오른쪽 서브트리의 위치를 변경하는 함수 "struct btnode *swap(struct btnode *original)" 이다. ㉮, ㉯, ㉰에 들어갈 코드는 무엇인가? [10]

```
struct btnode { int  data;    struct btnode *lchild;    struct btnode *rchild; };
struct btnode *swap(struct btnode *original)
{
    struct btnode *temp;
    if (original) {
        temp = (struct btnode *) malloc(sizeof(struct btnode));
        temp->lchild = ㉮;    temp->rchild = ㉯;    temp->data = ㉰;
        return temp;
    }
    return NULL;
}
```

swap(original->rchild) swap(original->lchild) original->data

3. 다음은 first와 second가 가리키는 두 개의 이진트리가 동일한지를 검사하는 알고리즘이다. ㉮와 ㉯, ㉰에 들어갈 내용은 무엇인가? [10]

```
typedef struct node *tree_ptr;
struct node {
    int data;
    tree_ptr left_child;
    tree_ptr right_child;
};
```

```
int equal(tree_ptr first, tree_ptr second)
{
    return ((㉮) || (first && second &&
    (first->data == second->data) &&
    ( ㉯ ) && equal(first->left_child, second->left_child)
    ( ㉰ ))     equal(first->right_child, second->right_child)
}
```

(!first && !second)

12. 쓰레드 이진 트리(threaded binary tree)에서 parent 노드의 왼쪽에 child 노드를 추가하는 함수 "void insert_left(struct tbt *parent, struct tbt *child)" 를 작성하라. [20]

```
struct tbt {
    int data;
    short int left_thread;    short int right_thread;
    struct tbt *lchild;       struct tbt *rchild;
};

struct tbt *inpred(struct tbt *ptr)
{
    struct tbt *temp = ptr->lchild;

    if (!ptr->left_thread)
        while (!temp->right_thread)
            temp = temp->rchild;
    return temp;
}

void insert_left(struct tbt *parent, struct tbt *child)
{

}
```

```
struct tnode *temp;

child->lchild = parent->lchild;
child->left_thread = parent->left_thread;

child->rchild = parent;
child->right_thread = 1;

parent->lchild = child;
parent->left_thread = 0;

if(child->left_thread == 0){
    temp = inpred(child);
    temp->rchild = child;
}
```

13. 다음은 이진트리에 포함된 노드의 수를 반환하는 함수이다. 가와 나에 포함될 내용은 무엇인가? [10]

```
struct node {
    int data;
    struct node *lchild;
    struct node *rchild;
};
```

```
int count(struct node *ptr)
{
    if ( 가 ) ptr == NULL
        return 0;
    else
        return 나; 1 + count(ptr->lchild) + count(ptr->rchild)
}
```

17. 다음은 이진검색 트리(binary search tree)에서 data 값이 가장 큰 노드에 대한 포인터를 반환하는 함수이다. 가와 나에 들어갈 내용은 무엇인가? [10]

```
struct node {   int data;   struct node *rchild;   struct node *lchild;   };
struct node *FindMax(struct node *ptr) {
    if (ptr == NULL)   return NULL;
    else if (ptr->rchild == NULL)   가;   return ptr
    else 나;   return FindMax(ptr->rchild)
}
```

19. 다음은 이진트리(binary tree)의 깊이를 구하는 알고리즘이다. 가와 나에 들어갈 내용은 무엇인가? [10]

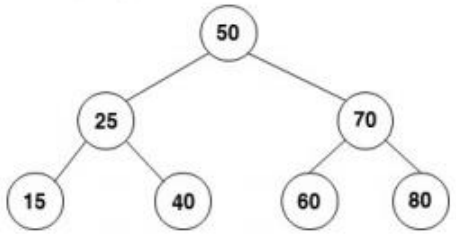
```
struct node {   int data;   struct node *rchild;   struct node *lchild;   };
int tree_depth(struct node *ptr) {
    int n1, n2;
    if (ptr == NULL)   return 0;
    n1 = 가;   tree_depth(ptr->lchild);
    n2 = 나;   tree_depth(ptr->rchild);
    return (n1 > n2) ? n1 + 1 : n2 + 1;
}
```

25. 다음은 이진트리에서 자식 노드가 2개인 노드의 수를 반환하는 함수이다. 가와 나에 들어갈 내용은 무엇인가? [20]

```
struct btnode {     int data;    struct btnode *lchild;    struct btnode *rchild; };

int count_node2(struct btnode *ptr)
{
    if (ptr == NULL)    return 0;
    if ( 가 )  ptr->rchild && ptr->lchild
        return 나;  return 1 + count_node2(ptr->lchild) + count_node2(ptr->rchild)
    else
        return count_node2(ptr->lchild) + count_node2(ptr->rchild);
}
```

29. 2. 이진검색 트리에서 임의의 노드 x에 대한 조상(ancestor)들을 루트노드부터 x까지 차례대로 출력하는 아래 함수를 완성하라. 예를 들면, 아래 트리에서 ancestor(root, 40)를 호출할 경우 50, 25, 40이 차례대로 출력된다. [15]




```
struct btnode
{   int data;       struct btnode *lchild;       struct      btnode
*rchild; };

void ancestor(struct btnode *root, int data) {
    if (root == NULL) return;
    if (root→data == data)      {          가      } printf("%d", root->data);
    else if (root→data < data)  {          나      } printf("%d", root->data);
                                                     ancestor(root->rchild, data);
    else                        {          다      } printf("%d", root->data);
                                                     ancestor(root->lchild, data);
}
```

28. 다음은 AOE 네트워크에서 vertex의 earliest time을 계산하는 프로그램이다. 그래프는 인접 리스트로 구성되어 있으며, struct node의 dur 필드는 에지의 소요 시간을 저장한다. 그리고 push()과 pop() 연산은 이미 구현되어 있다고 가정하자. 가와 나에 들어갈 내용은 무엇인가? [15]

```
struct node {   int vertex;  int dur; struct node *link; };
struct hdnode {     int count;   struct node *link; };

void calculateEarliestTime(hdnode graph[], int n, int earliest[])
{
    int i, j, k;
    struct node *ptr;

    for (i = 0; i < n; i++) {
        if ( graph[i].count == 0 )          graph[i].count=top; top=i
            push( i );                  // stack에 추가
        earliest[i] = 0;        // earliest time을 0으로 초기화
    }

    for (i = 0; i < n; i++) {
        if (StackEmpty() == true)   return;        // 오류
      else {              j=top; top=graph[top].count
            j = pop( );                     // stack에서 제거
            for ( 가 ) {ptr = graph[j].link; ptr != NULL; ptr = ptr->link
                k = ptr->vertex;
                graph[k].count--; ;
                if ( graph[k].count == 0 )
                    push( k );                  // stack에 추가
                                graph[k].count=top; top = k
    나      if (earliest[k] < earliest[j] + ptr->dur)
                earliest[k] = earliest[j] + ptr->dur;
            }
        }
    }
}
```

16. 다음은 Dijkstra의 shortest path를 발견하는 알고리즘이다. 가와 나에 들어갈 내용은 무엇인가? 단, 그래프는 인접 행렬 cost[][]에 저장되어 있다고 가정한다. [10]

```
void shortestpath(int v, int cost[][MAX_VERTICES], int distance[], int n, short int found[])
{       // Dijkstra의 Algorithm을 구현
    int i, u, w;
    for (i = 0; i < n; i++)  {    found[i] = FALSE;   distance[i] = cost[v][i];  }

    found[v] = TRUE;
    distance[v] = 0;
    for (i = 0; i < n - 2; i++) {
        u = choose(distance, n, found);      // w ∉ S 중에서 최단 경로 선택
        found[u] = TRUE;
        for (w = 0; w < n; w++)
            if (found[w] == FALSE)
                if ( 가 ) distance[w] > distance[u] + cost[u][w]
                    나 ; distance[w] = disatnce[u] + cost[u][w]
    }
}
```

32. 다음은 A를 루트로 하는 이진검색 트리에서 key가 입력될 위치를 반환하는 함수이다. 가와 나에 들어갈 내용은 무엇인가?

```
struct node {   int  data; struct node *lchild, *rchild; };

struct node *modified_search(struct node *A, int key)
{
    for (struct node *ptr = A; ptr != NULL; ) {
        if (ptr→data == key)        // 기존에 존재하는 키를 입력: 오류!
            return NULL;
        if (key < ptr→data) {
            if ( 가 ) ptr->lchild == NULL
                return ptr;
            else ptr = ptr→lchild;
        }
        else {
            if ( 나 ) ptr->rchild == NULL
                return ptr;
            else ptr = ptr→rchild;
        }
    }
    return NULL;              // root가 NULL일 경우
}
```

3. 다음은 threaded binary tree에서 parent 노드의 왼쪽에 노드를 추가하는 알고리즘이다. 가부터 다에 들어갈 내용은 무엇인가? [20]

```
struct tnode {  int  data; struct tnode *lchild, *rchild;
                                  short  lthread, rthread; };

struct tnode *inpred(struct tnode *ptr)
{   // ptr이 가리키는 노드의 inorder predecessor를 return
    struct tnode *temp = ptr->lchild;
    if (ptr->lthread == 0) {        // lchild가 thread가 아님
        while (temp->rthread == 0)
            temp = temp->rchild;   가
    }
    return temp;
}

void insert_left(struct tnode *parent, struct tnode *child)
{    // parent의 왼쪽에 child 추가
    struct tnode *temp;
    child->lchild = parent->lchild;
    child->lthread = parnet->lthread;   나
    child->rchild = parent;
    child->rthread = 1;
    parent->lchild = child;
    parent->lthread = 0;
    if (child->lthread == 0) {
        temp = inpred(child);
        tmep->rchild = child;          다
    }
}
```

6. 정수 배열 list[]에 대해 list[i]..list[m]까지는 정렬되어 있고, list[m+1]..list[n]까지도 정렬되어 있다고 하자. 정렬된 두 개의 그룹을 합병하여 정수 배열 sorted[]에 저장하는 함수 merge()에 대해 가, 나, 다에 들어갈 내용은 무엇인가? [10]

```
void merge(int list[], int sorted[], int i, int m, int n)
{
    int j = m + 1, k = i, t;

    while ( 가 ) {  i<=m && j<=n
        if ( 나 )  list[j]>=list[i]   sorted[k++] = list[i++];
        else       sorted[k++] = list[j++];
    }
    if ( 다 ) m < i
        for (t = j; t <= n; t++)       sorted[k++] = list[t];
    else
        for (t = i; t <= m; t++)       sorted[k++] = list[t];
}
```

8. 정수 배열 list[]를 정렬하는 아래 quicksort 알고리즘에서 가와 나에 들어갈 내용은 무엇인가?

```
void quicksort (int list[ ], int left, int right)
{
    int pivot, i, j;
    int temp;
    if (left < right) {
        i = left;    j = right + 1;    pivot = list[left];
        do {
            do
                i++;
            while (list[i] < pivot);
            do
                j--;
            while (list[j] > pivot);
            if ( i < j )
                SWAP( list[i], list[j], temp );
        } while ( i < j );
        SWAP( list[left], list[j], temp );
        가; quicksort(list, left, j-1);
        나; quicksort(list, j+1, right);
    }
}
```

4. n개의 데이터를 저장하는 정수 배열 list[]를 정렬하기 위한 삽입 정렬(insertion sort) 알고리즘이 아래와 같다. 가와 나에 들어가는 내용은? [10]

```
void insertion_sort(int list[], int n)
{
        int i, j, next;
    for (i = 1; i < n; i++) {
        next = list[i];
        for ( 가 )  j=i-1; next<list[j] && j>=0; j--
            나 ; list[j+1] = list[j];
        list[j+1] = next;
    }
}
```