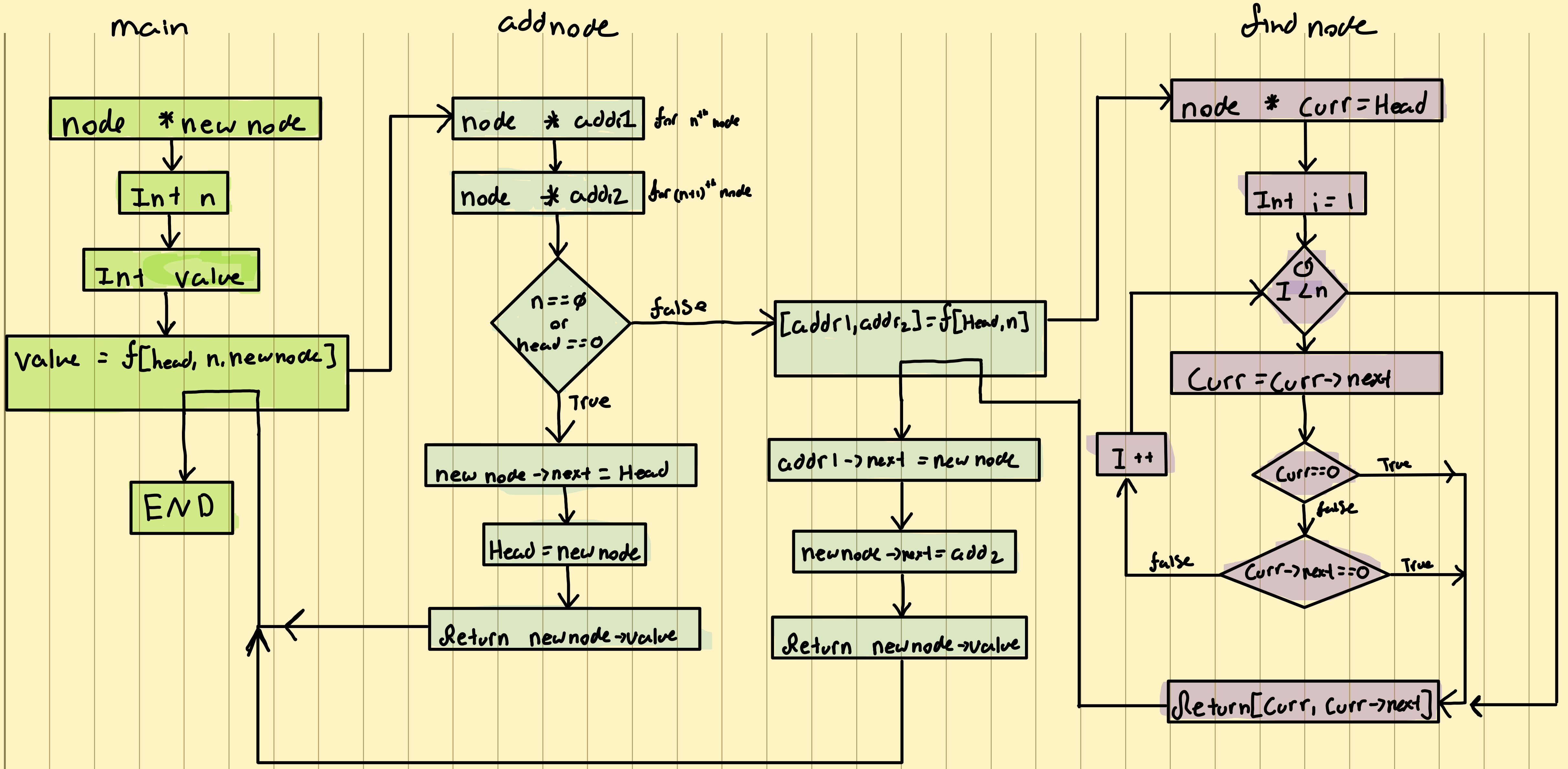

Register	data	variable
$s0	x	*Head
$s1	y	*new node
$s2	[illegible]	n - where to insert
$s3	0	value that will be returned

```
// Parameters of a node in the linked list (need not declare or initialize in MIPS)
typedef struct node {
    int value;    // Value in the node accessed by node->value
    node* next;    // Address of next node accessed by node->next
} node;       // Datatype for each node

node *head;     // address of head (first node) of linked list (global pointer)

int main() {
    // Variable Declaration
    node *newNode;     // address of node to be added
    int n;      // number of the node in the list after which node is to be added
    int value;    // Value of the node to be added

    // Task of main function
    value = addNode(head, n, newNode);
}

int addNode (node* head, int n, node* newNode) {
    node *addr1, *addr2; // addr1 = address of n^th node, addr2 =  address of (n+1)^th node
    if (n == 0 || head == 0) {                    // If node should be added at the beginning of the 
list
        newNode->next = head;   // Next for new node = head of original list
        head = newNode;     // global head updated to the new node
        return(newNode->value);  // value of the node = data at the address of the node, and then 
return to caller
    }
    [addr1, addr2] = findNode (head, n);     // Call findNode function
    addr1->next = newNode;    // Next for n^th node = node to be added
    newNode->next = addr2;    // Next for added node = (n+1)^th node of original list
    return(newNode->value);     // value of the node = data at the address of the node
}

node* findNode (node* head, int n) {
    node* curr = head;    // Start with head of linked list
    for (int i = 1; i < n; i ++) {
        curr = curr->next;    // Update the pointer to next node address
        if (curr == 0)            // Break if end of List
            break;
        if (curr->next == 0)            // Break if end of List
            break;
    }
    return([curr, curr->next]);         // Two return values (need not return as array in MIPS)
}
```

Registers	Variables
$s0	head
$s1	newNode
$s2	n
$s3	val

Addresses	Contents
newNode	newNode->value
head	node1->value
head + 4	node1->next
node1->next	node2->value
node1->next + 4	node2->next
node2->next	node3->value
node2->next + 4	node3->next
...	...