

# ECE 220 Computer Systems & Programming

## Lecture 22 – C to LC-3 with Linked Data Structure



# C to LC-3 – Assembly Translation with linked data structure

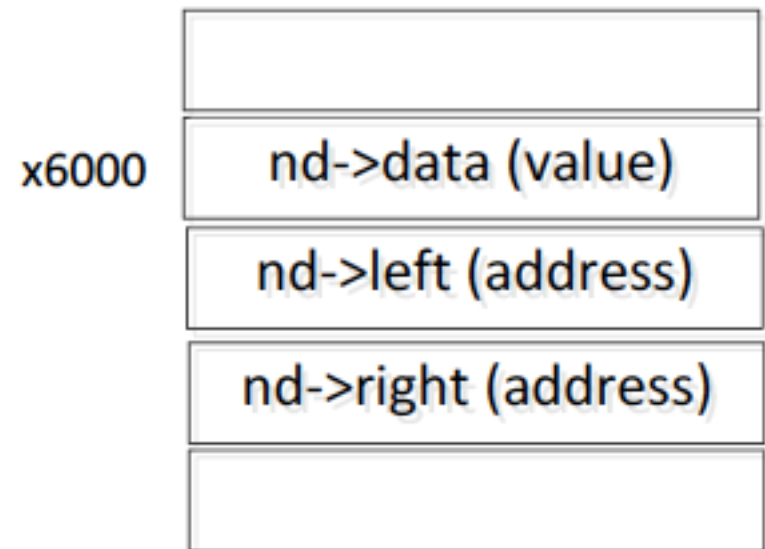
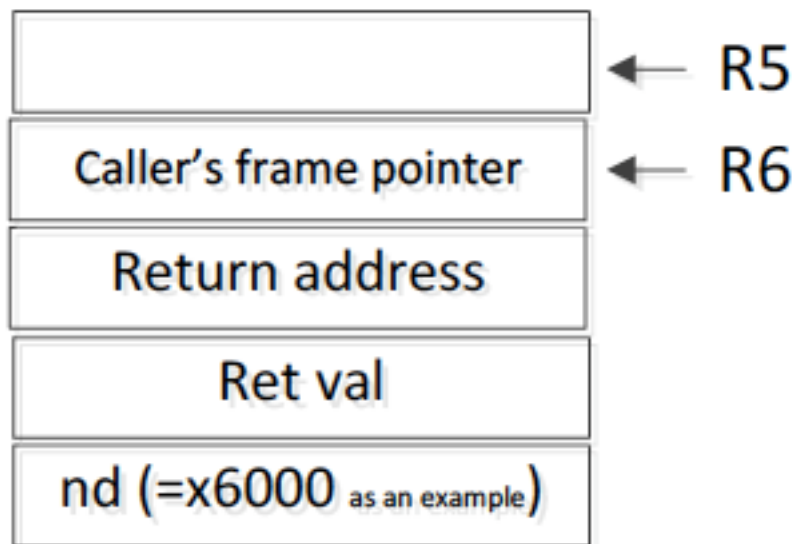
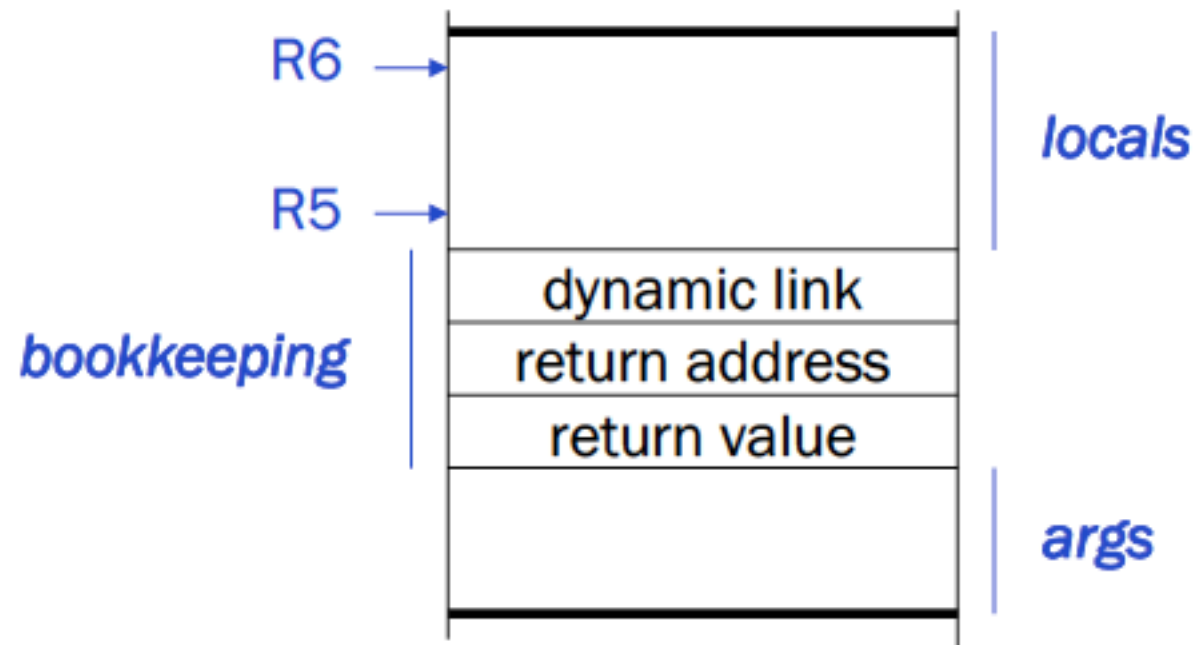
## Recursive tree traversal

**Problem statement:** Convert the following function from C to LC-3. This function recursively traverses a binary tree.

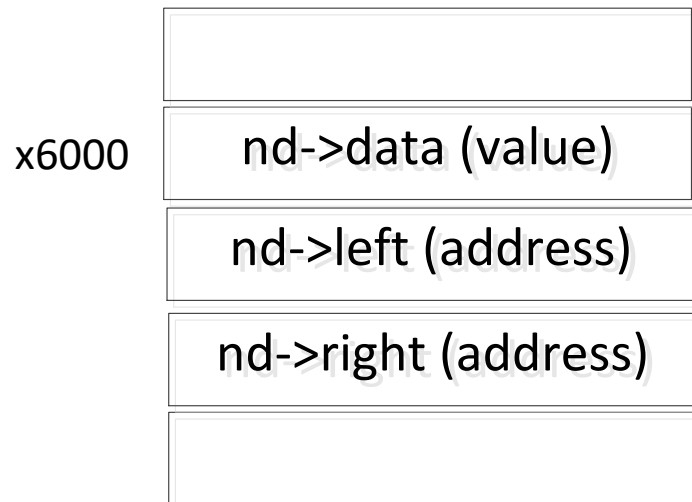
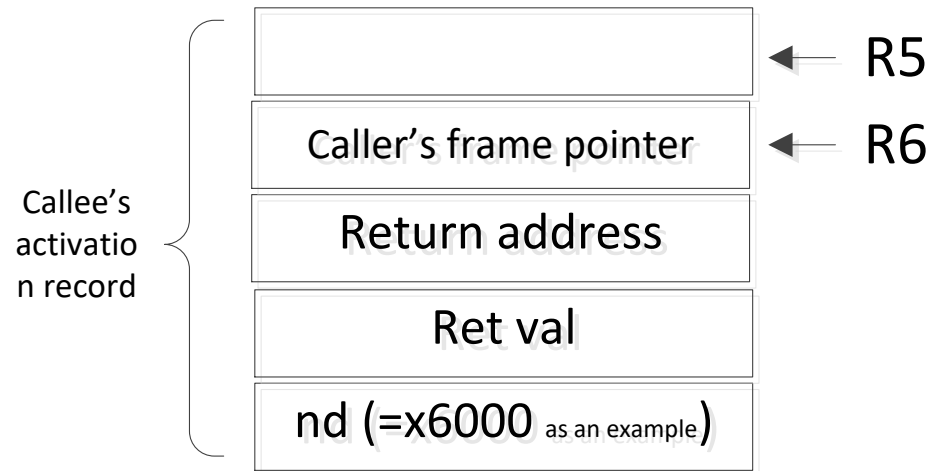
```
void TraverseTree(t_node *nd)
{
    if (nd != NULL)
    {
        TraverseTree(nd->left);
        TraverseTree(nd->right);
    }
}
```

```
typedef struct nodeTag t_node;
struct nodeTag
{
    int data;
    t_node *left;
    t_node *right;
};
```

# Activation Record



# Step#1



## TRAVERSE\_TREE

; Allocate space for return value

**ADD R6, R6, #-1**

; Push return address to stack

**ADD R6, R6, #-1**

**STR R7, R6, #0**

; Store callee's frame pointer

**ADD R6, R6, #-1**

**STR R5, R6, #0**

; Set up new frame pointer

**ADD R5, R6, #-1**

## Step#2: Implement Logic Function

; if (nd == NULL), skip to the end

**LDR R0, R5, #4;**

**BRz DONE**

; TraverseTree(nd->left);

**LDR R1, R0, #1 ; load nd->left to R1**

; push nd->left to stack

**ADD R6, R6, #-1**

**STR R1, R6, #0**

; call subroutine

**JSR TRAVERSE\_TREE**

; tear-down the rest of the stack

**ADD R6, R6, #2**

; TraverseTree(nd->right);

**LDR R0, R5, #4**

**LDR R2, R0, #2 ; load nd->right to R2**

; push nd->right to stack

**ADD R6, R6, #-1;**

**STR R2, R6, #0;**

; call subroutine

**JSR TRAVERSE\_TREE**

; tear-down the rest of the stack

**ADD R6, R6, #2**

**Teardown the activation record, return:**

**DONE**

; Restore frame pointer

**LDR R5, R6, #0**

**ADD R6, R6, #1**

; Restore return address

**LDR R7, R6, #0**

**ADD R6, R6, #1**

**RET**

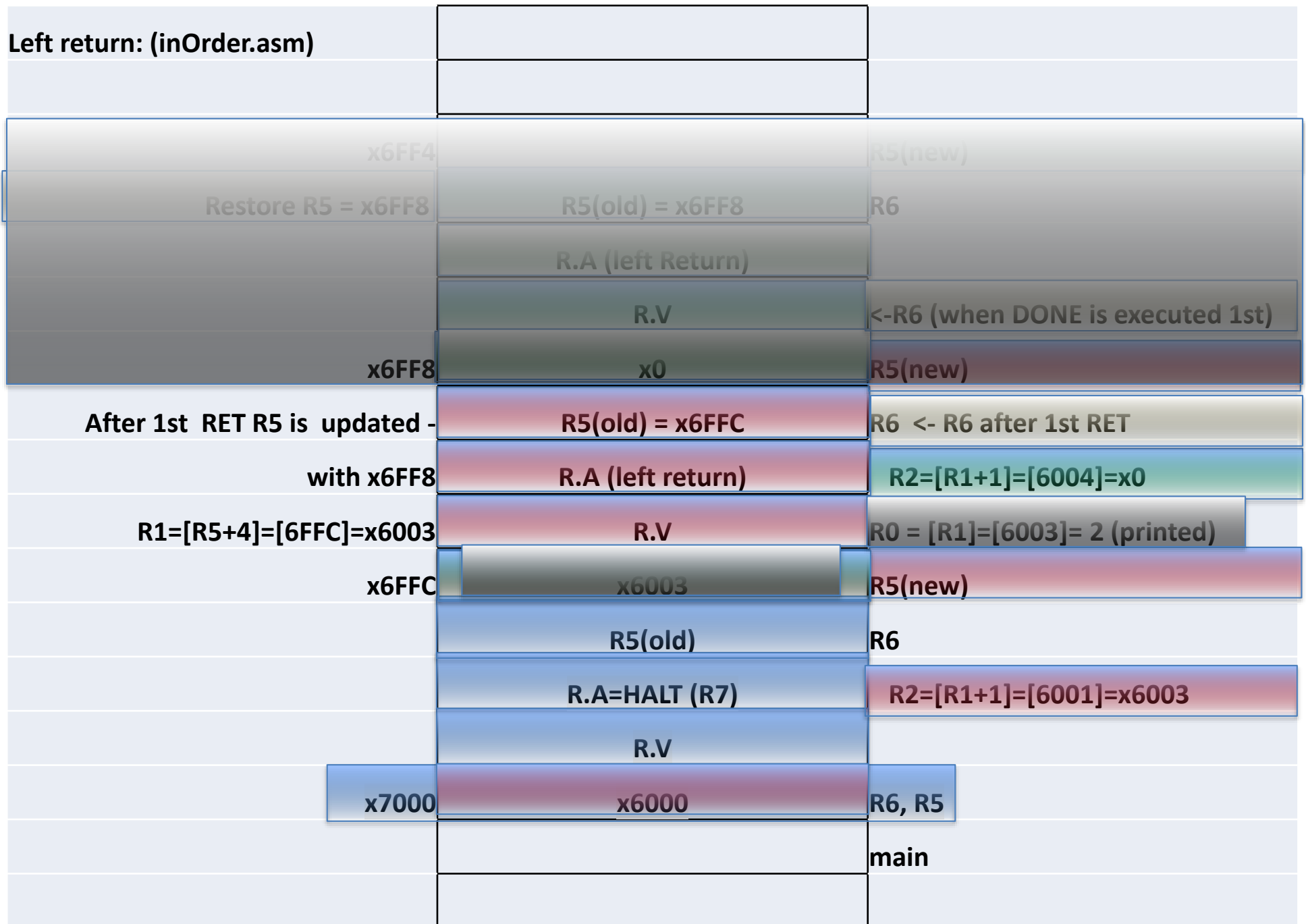
## inOrder LC3 (please see, inOrder.asm in github)

```
void inorder(t_node *node)
{
    // Base case
    if (node == NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```

Left return: (inOrder.asm)		
x6FF4		R5(new)
	R5(old) = x6FF8	R6
	R.A (left Return)	
	R.V	<-R6 (when DONE is executed 1st)
x6FF8	x0	R5(new)
After 1st RET R5 is updated -	R5(old) = x6FFC	R6 <- R6 after 1st RET
with x6FF8	R.A (left return)	R2=[R1+1]=[6004]=x0
R1=[R5+4]=[6FFC]=x6003	R.V	R0 = [R1]=[6003]= 2 (printed)
x6FFC	x6003	R5(new)
	R5(old)	R6
	R.A=HALT (R7)	R2=[R1+1]=[6001]=x6003
	R.V	
x7000	x6000	R6
		main

Right Return (inOrder.asm)		
x6FF4		R5(new)
After 2nd RET R5 is updated -	R5 (old) =x6FF8	R6
with x6FF8	R.A (right return) R7	
(after 2nd DONE, RET)->R6	R.V	R3= [R1+2]=[6004]= 0 (NULL)
x6FF8	x0	R5(new) <-R6
After 2nd RET R5 is updated -	R5(old) = x6FFC	R6 <- R6 after 2nd RET
with x6FFC	R.A (left return)	R2=[R1+1]=[6004]=x0 (NULL)
<b>After 2nd return R7 is left return</b>	R.V	<b>R0 = [R1]=[6003]= 2 (printed)   &lt;-R6</b>
x6FFC	x6003	R5(new)
	R5(old)	R6
	R.A=HALT (R7)	R2=[R1+1]=[6001]=x6003
	R.V	
x7000	x6000	R6
		main





Right Return (inOrder.asm)		
x6FF4		R5(new)
After 2 <sup>nd</sup> DONE R5 is updated -	R5 (old) =x6FF8	R6 ; R5=x6FF8 (after 2 <sup>nd</sup> DONE)
with x6FF8	R.A (right return) R7	
(after 2nd DONE, before RET)->R6	R.V	R3= [R1+2]=[6005]= 0 (NULL)
x6FF8	x0	R5(new) <-R6
After 2nd RET R5 is updated -	R5(old) = x6FFC	R6 <- R6 after 2nd RET
with x6FFC	R.A (left return)	R2=[R1+1]=[6004]=x0 (NULL)
After 2nd return R7 is left return	R.V	R6 (before 2nd RET)
After 2 <sup>nd</sup> RET R5 = x6FFC	x6003	R5(new)
R1 = [R5+4] = [x7000] = x6000	R5(old)	R6
R0=[R1]=[x6000]=x38 (print 8)	R.A=HALT (R7)	R2=[R1+1]=[6001]=x6003
	R.V	
x7000	x6000	R6
		main

# Recursive linked list traversal

**Problem statement:** Convert the following function from C to LC-3. This function recursively traverses a linked list and prints its content.

```
/* typedef struct tag {char data; struct tag *next;} node; */
```

```
int print_list(node *head)
{
    if (!head) return 0;
    printf("%c", head->data);
    return print_list(head->next);
}
```

# Main function: (print\_list.asm)

```
        .ORIG x3000
MAIN
        LD R5, RSTACK
        LD R6, RSTACK

        LD R0, HEAD
        STR R0, R6, #0 ; push list head address to the stack

        JSR PRINT_LIST

        HALT

HEAD
        .FILL x2004

RSTACK
        .FILL x7000
```

## PRINT\_LIST

; Bookkeeping

**ADD** R6, R6, #-3 ; Space for bookkeeping

**STR** R7, R6, #1 ; Save return address

**STR** R5, R6, #0 ; Save prev. frame pointer

**ADD** R5, R6, #-1 ; Move frame pointer

; if (!head) return 0;

**LDR** R1, R5, #4 ; R1 <- head

**BRz** DONE ; if head is NULL

; printf("%c", head->data);

**LDR** R0, R5, #4

**LDR** R0, R0, #0

**OUT**

```
; print_list(head->next)
```

```
LDR R1, R1, #1 ; R1 <- head->next
```

```
ADD R6, R6, #-1 ; Push head->next as parameter
```

```
STR R1, R6, #0
```

```
JSR PRINT_LIST
```

```
; return
```

```
LDR R0, R6, #0 ; Load return value to R0
```

```
STR R0, R5, #3 ; Store return value from R0 to correct location
```

```
ADD R6, R6, #2
```

```
BR TEARDOWN
```

DONE

```
AND R0, R0, #0
```

```
STR R0, R5, #3
```

TEARDOWN

```
LDR R7, R5, #2 ; Restore R7
```

```
LDR R5, R5, #1 ; Restore R5
```

```
ADD R6, R6, #2 ; Pop stack
```

```
RET
```

```
.END
```

## Data file: data.asm

```
; data.asm  
    .ORIG x2000  
  
    .FILL x43  
    .FILL x2006  
  
    .FILL x41  
    .FILL x2000  
  
    .FILL x46  
    .FILL x2002  
  
    .FILL x45  
    .FILL x0  
  
    .END
```