```
/**
 * ListNode - The structure of a list node containing an integer value and a poi
nter to the next node
 *
 * @param value: The integer value in the node
 * @param next:  Pointer to the next ListNode in the list
 */
.data
    .align 8
    ListNode:
        .quad 0            // value
        .quad 0            // next (ptr)

    .align 8
    ListNode_size:
        .quad ListNode_end - ListNode

    .align 8
    ListNode_end:

.text
/**
 * linked_list_create - Creates an empty linked list
 *
 * @return x0: Pointer to the head node (NULL for an empty list)
 *
 * Registers used: none
 */
.global linked_list_create
    linked_list_create:
        mov x0, #0  // Initialize the head node as NULL (empty list)
        ret

/**
 * linked_list_append - Appends an element to the end of the linked list
 *
 * @param x0: Pointer to the current head of the linked list
 * @param w1: The integer value to be appended to the list
 *
 * @return x0: Pointer to the new head of the linked list
 *
 * Registers used: x0, x1, x2, x8, lr
 * Registers saved: lr
 */
.global linked_list_append
linked_list_append:
    // Save lr
    str lr, [sp, #-16]!

    // Allocate memory for the new node and store the value
    ldr x2, =ListNode_size
    bl malloc
    str w1, [x0]
    // Initialize the 'next' field of the new node to NULL
    str xzr, [x0, #8]

    // Debugging information
    ldr x2, =print_node_format
    bl printf                // printf("Append: new node=%p, head=%p\n", x0, x1);

    // Branch to handle special case when the head is NULL (i.e., when list is e
mpty)
```

```
    cbz x0, empty_list

    // Traverse the list to find the last node
    ldr x1, [x0, #8]
    bl printf                // printf("Append: start traversal, node=%p, next=%p\n", x0, x1
);
    cbz x1, insert_node
traversal_loop:
    mov x0, x1
    ldr x1, [x0, #8]
    bl printf                // printf("Append: next node=%p, next=%p\n", x0, x1);
    cbz x1, insert_node
    b traversal_loop

insert_node:
    // Insert the new node and update the 'next' pointer of the last node
    bl printf                // printf("Append: inserting node=%p, next=%p\n", x0, x1);
    str x0, [x0, #8]
    b finished

empty_list:
    // For an empty list, the new node becomes the head
    bl printf                // printf("Append: empty list, setting head=%p\n", x0);
    str xzr, [x0, #8]

finished:
    // Restore lr and return
    ldr lr, [sp], #16
    ret

/**
 * linked_list_print - Prints the elements of a linked list
 *
 * @param x0: Pointer to the head node of the linked list
 *
 * Registers used: x0, w1, lr
 * Registers saved: lr
 */
.extern printf
.global linked_list_print
    linked_list_print:
        // Save lr
        str lr, [sp, #-16]!

        // Traverse the list and print each element
        ldr x2, =print_node_format
    print_loop:
        cbz x0, done_print   // Exit loop if the end of the list is reached
        ldr w1, [x0]
        bl printf            // printf("%d\n", w1);;
        ldr x0, [x0, #8]
        b print_loop

    done_print:
        ldr lr, [sp], #16
        ret

data:
    print_node_format: .asciz "%d\n"

/**
 * linked_list_free - Frees the memory allocated for the linked list
 *
 * @param x0: Pointer to the head node of the linked list
 *
 * Registers used: x0, x1
 * Registers saved: none
 */
.global linked_list_free
    linked_list_free:
        cbz x0, done_free    // Exit if the end of the list is reached
        ldr x1, [x0, #8]
        bl free
        mov x0, x1
        b linked_list_free

    done_free:
        ret
```