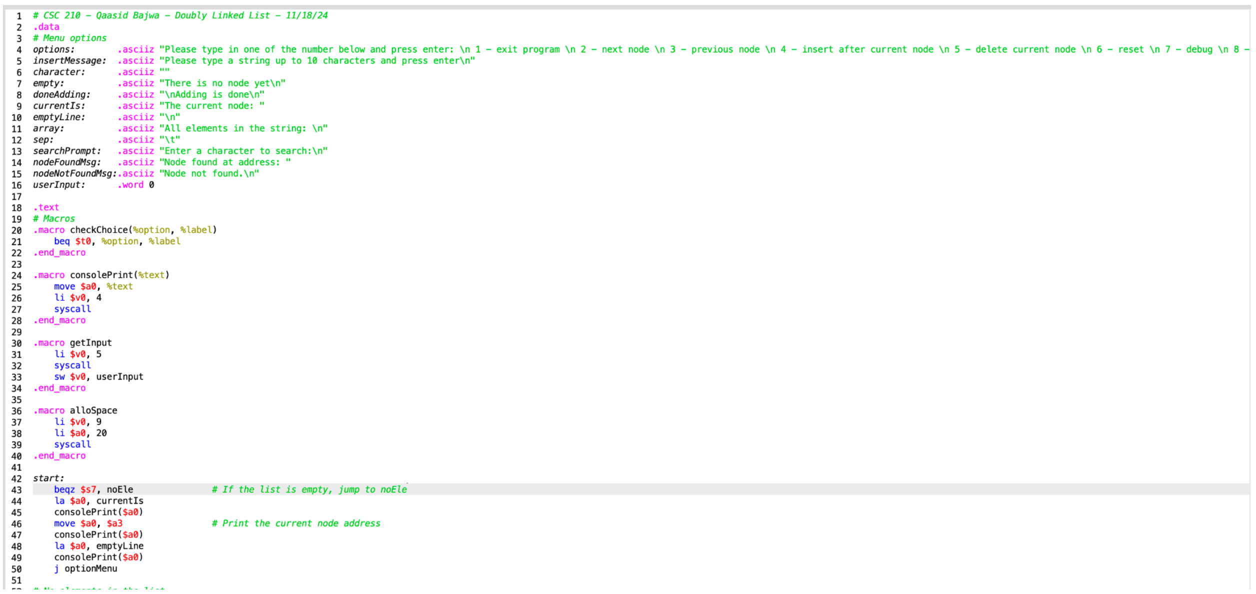
Bajwa, Qaasid 11/18/24

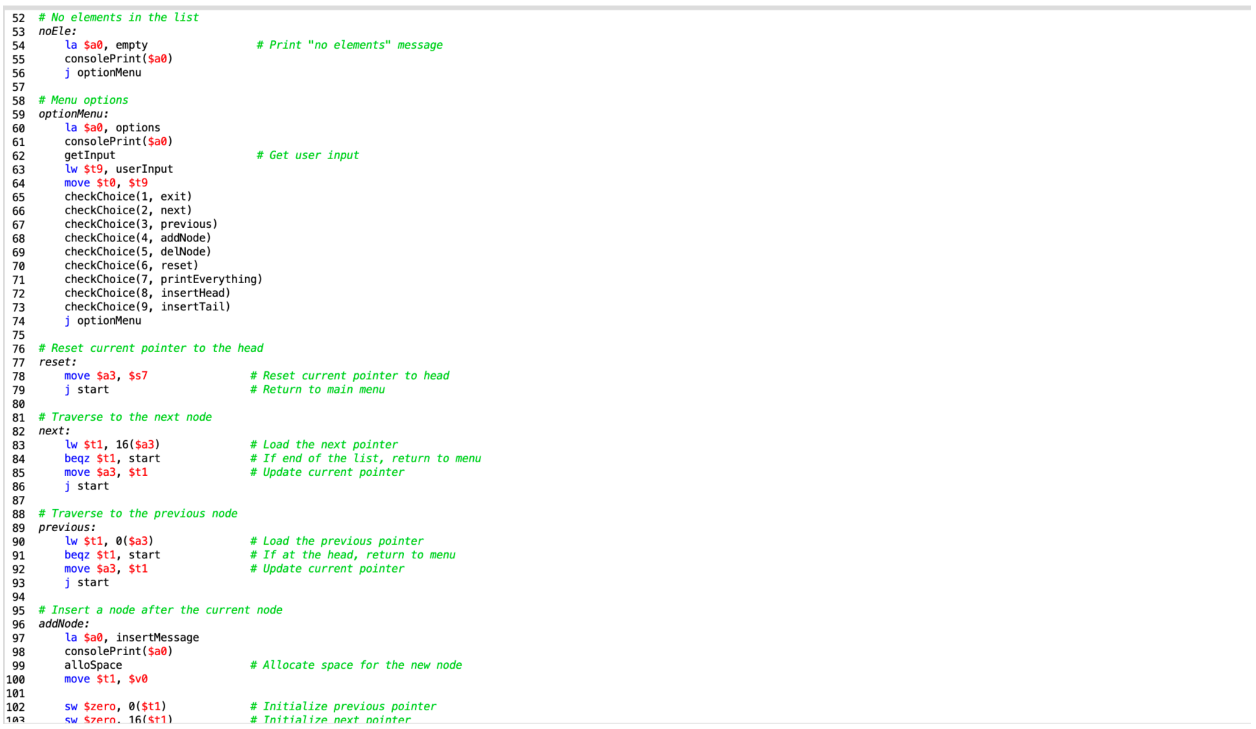
CSC21000 - Converting Single Linked List to Doubly Linked List

This homework assignment entailed transforming an existing single linked list program into a fully functional doubly linked list implementation. A doubly linked list allows traversal in both forward and backward directions, which necessitates maintaining both next and previous pointers for each node. Additional improvements include adding nodes at specific positions (head, tail, or after the current node), deleting nodes (from the head, tail, or a specific position), debugging by printing all elements, and searching for a node containing a specific value.

Section 1: Code

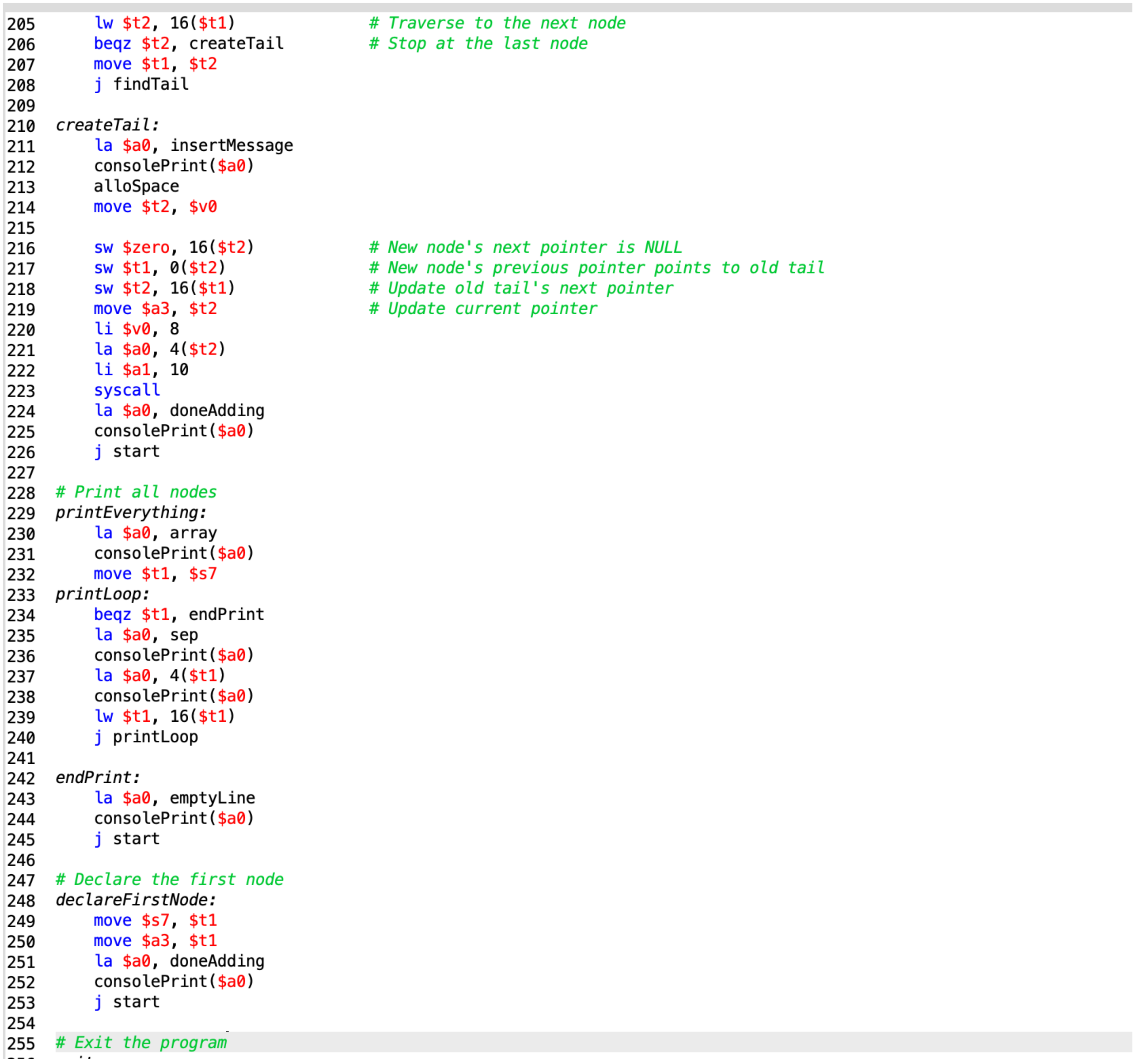


```
# CSC 210 - Qaasid Bajwa - Doubly Linked List - 11/18/24
.data
# Menu options
options:        .asciiz "Please type in one of the number below and press enter: \n 1 - exit program \n 2 - next node \n 3 - previous node \n 4 - insert after current node \n 5 - delete current node \n 6 - reset \n 7 - debug \n 8 -
insertMessage:  .asciiz "Please type a string up to 10 characters and press enter\n"
character:      .asciiz ""
empty:          .asciiz "There is no node yet\n"
doneAdding:     .asciiz "\nAdding is done\n"
currentIs:      .asciiz "The current node: "
emptyLine:      .asciiz "\n"
array:          .asciiz "All elements in the string: \n"
sep:            .asciiz "\t"
searchPrompt:   .asciiz "Enter a character to search:\n"
nodeFoundMsg:   .asciiz "Node found at address: "
nodeNotFoundMsg:.asciiz "Node not found.\n"
userInput:      .word 0

.text
# Macros
.macro checkChoice(%option, %label)
    beq $t0, %option, %label
.end_macro

.macro consolePrint(%text)
    move $a0, %text
    li $v0, 4
    syscall
.end_macro

.macro getInput
    li $v0, 5
    syscall
    sw $v0, userInput
.end_macro

.macro alloSpace
    li $v0, 9
    li $a0, 20
    syscall
.end_macro

start:
    beqz $s7, noEle                 # If the list is empty, jump to noEle
    la $a0, currentIs
    consolePrint($a0)
    move $a0, $a3                   # Print the current node address
    consolePrint($a0)
    la $a0, emptyLine
    consolePrint($a0)
    j optionMenu

```

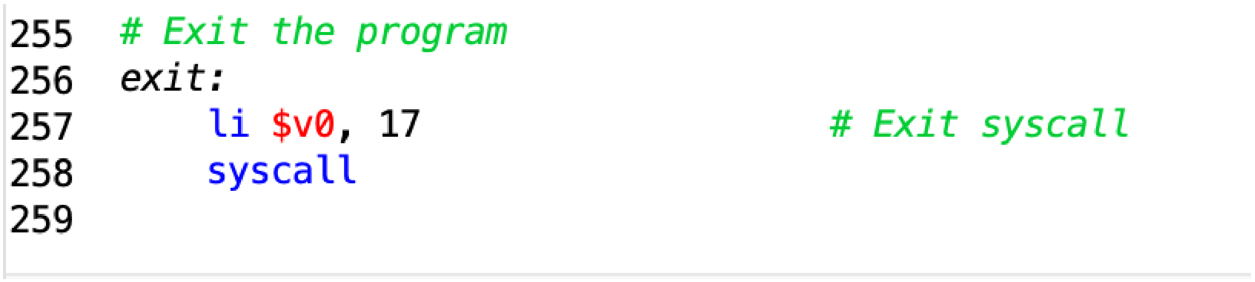


```
# No elements in the list
noEle:
    la $a0, empty                   # Print "no elements" message
    consolePrint($a0)
    j optionMenu

# Menu options
optionMenu:
    la $a0, options
    consolePrint($a0)
    getInput                        # Get user input
    lw $t9, userInput
    move $t0, $t9
    checkChoice(1, exit)
    checkChoice(2, next)
    checkChoice(3, previous)
    checkChoice(4, addNode)
    checkChoice(5, delNode)
    checkChoice(6, reset)
    checkChoice(7, printEverything)
    checkChoice(8, insertHead)
    checkChoice(9, insertTail)
    j optionMenu

# Reset current pointer to the head
reset:
    move $a3, $s7                  # Reset current pointer to head
    j start                        # Return to main menu

# Traverse to the next node
next:
    lw $t1, 16($a3)                # Load the next pointer
    beqz $t1, start                # If end of the list, return to menu
    move $a3, $t1                  # Update current pointer
    j start

# Traverse to the previous node
previous:
    lw $t1, 0($a3)                 # Load the previous pointer
    beqz $t1, start                # If at the head, return to menu
    move $a3, $t1                  # Update current pointer
    j start

# Insert a node after the current node
addNode:
    la $a0, insertMessage
    consolePrint($a0)
    alloSpace                      # Allocate space for the new node
    move $t1, $v0

    sw $zero, 0($t1)               # Initialize previous pointer
    sw $zero, 16($t1)              # Initialize next pointer
    sw $zero, 16($t1)            # Initialize next pointer

    # Read input string for the node
    li $v0, 8
    la $a0, 4($t1)
    li $a1, 10
    syscall

    # If list is empty, make this the first node
    beqz $s7, declareFirstNode

    lw $t2, 16($a3)              # Load the next pointer of the current node
    beqz $t2, noNextNode

    # Insert in the middle
    sw $t1, 16($a3)              # Update current node's next pointer
    sw $a3, 0($t1)               # Update new node's previous pointer
    sw $t2, 16($t1)              # Update new node's next pointer
    sw $t1, 0($t2)               # Update next node's previous pointer
    j start

noNextNode:
    sw $t1, 16($a3)              # Update current node's next pointer
    sw $a3, 0($t1)               # Update new node's previous pointer
    move $a3, $t1                # Update current pointer
    la $a0, doneAdding
    consolePrint($a0)
    j start

# Delete the current node
delNode:
    beqz $s7, start              # If the list is empty, return to the menu

    # Handle deleting the head
    lw $t2, 0($a3)               # Load the previous pointer
    beqz $t2, delHead            # If no previous node, delete the head

    # Handle deleting the tail
    lw $t3, 16($a3)              # Load the next pointer
    beqz $t3, delTail            # If no next node, delete the tail

    # General case: Delete a node in the middle
    lw $t3, 16($a3)              # Load the next node
    sw $t2, 0($t3)               # Update the previous pointer of the next node
    lw $t2, 0($a3)               # Load the previous node
    sw $t3, 16($t2)              # Update the next pointer of the previous node

    move $a3, $t3                # Move the current pointer to the next node
    j start                      # Return to the menu

# Subroutine for deleting the head node
```

```
delHead:
    lw $t2, 16($a3)              # Load the next node
    beqz $t2, resetList          # If no next node, reset the list
    sw $zero, 0($t2)             # Update the new head's previous pointer
    move $s7, $t2                # Update the head pointer
    move $a3, $t2                # Update the current pointer
    j start                      # Return to the menu

# Subroutine for deleting the tail node
delTail:
    lw $t2, 0($a3)               # Load the previous node
    sw $zero, 16($t2)            # Update the new tail's next pointer
    move $a3, $t2                # Update the current pointer
    j start                      # Return to the menu

# Subroutine for resetting the list (when it becomes empty)
resetList:
    move $s7, $zero              # Reset the head pointer
    move $a3, $zero              # Reset the current pointer
    j start                      # Return to the menu

# Insert at the head of the list
insertHead:
    la $a0, insertMessage
    consolePrint($a0)
    alloSpace                   # Allocate space for the new node
    move $t1, $v0

    sw $zero, 0($t1)            # New node has no previous pointer
    move $t2, $s7               # Load the current head
    sw $t2, 16($t1)             # New node points to the old head as its next pointer

    beqz $t2, updateHead         # If the list is empty, skip pointer update
    sw $t1, 0($t2)              # Update old head's previous pointer

updateHead:
    move $s7, $t1               # New node becomes the head
    move $a3, $t1               # Update current pointer
    li $v0, 8                   # Read the input string for the node
    la $a0, 4($t1)
    li $a1, 10
    syscall
    la $a0, doneAdding          # Inform the user that the node was added
    consolePrint($a0)
    j start

# Insert at the tail of the list
insertTail:
    beqz $s7, insertHead         # If the list is empty, insert at the head
    move $t1, $s7
findTail:
```



```
    lw $t2, 16($t1)              # Traverse to the next node
    beqz $t2, createTail         # Stop at the last node
    move $t1, $t2
    j findTail

createTail:
    la $a0, insertMessage
    consolePrint($a0)
    alloSpace
    move $t2, $v0

    sw $zero, 16($t2)            # New node's next pointer is NULL
    sw $t1, 0($t2)               # New node's previous pointer points to old tail
    sw $t2, 16($t1)              # Update old tail's next pointer
    move $a3, $t2                # Update current pointer
    li $v0, 8
    la $a0, 4($t2)
    li $a1, 10
    syscall
    la $a0, doneAdding
    consolePrint($a0)
    j start

# Print all nodes
printEverything:
    la $a0, array
    consolePrint($a0)
    move $t1, $s7
printLoop:
    beqz $t1, endPrint
    la $a0, sep
    consolePrint($a0)
    la $a0, 4($t1)
    consolePrint($a0)
    lw $t1, 16($t1)
    j printLoop

endPrint:
    la $a0, emptyLine
    consolePrint($a0)
    j start

# Declare the first node
declareFirstNode:
    move $s7, $t1
    move $a3, $t1
    la $a0, doneAdding
    consolePrint($a0)
    j start

# Exit the program
```



```
# Exit the program
exit:
    li $v0, 17                   # Exit syscall
    syscall

```

Section 2: Code Explanation

This is my code for the program. The original program implemented a singly linked list with a head pointer ($s7) and a current pointer ($a3). It included functionality for adding nodes, deleting nodes, and printing the list. To upgrade to a doubly linked list, each node now has two pointers (Previous pointer: Points to the previous node in the list, and Next pointer: Points to the next node in the list). This required updating the memory allocation size for each node to include space for the previous pointer. We have some new functionality, such as traversing backward, inserting at the head, inserting at the tail, deleting at the head and tail, and search functionality.

The code starts off with the data section, where it has options, insertMessage, character, empty, doneAdding, currentIs, emptyLine, array, sep, searchPrompt, nodeFoundMsg, nodeNotFoundMsg, and userInput, which are self explanatory for their purpose.

Next, we have the text section, which contains our macros: checkChoice, consolePrint, getInput, and alloSpace. checkChoice checks if the user input matches a specific menu option and jumps to the corresponding label, consolePrint prints a string to the console using syscall, getInput reads an integer input from the user, and alloSpace allocates memory on the heap for a new node.

The main execution now starts. Start checks if the list is empty. If it is, it jumps to the noEle routine, otherwise it prints the current node's address and contents, then proceeds to the menu. noEle displays the "no elements" message if the list is empty and returns to the menu for further action. optionMenu prints the menu options and prompts the user to select an operation. It also uses the checkChoice macro to jump to the appropriate subroutine based on user input.

We then have our linked list operations. Reset resets the current pointer ($a3) to the head ($s7) and then returns to the main menu. Next **m**oves the current pointer to the next node by loading the address from the next pointer (16($a3)).If the current node is the last node (next == 0), it returns to the main menu. Previous moves the current pointer to the previous node by loading the address from the previous pointer (0($a3)).If the current node is the head (previous == 0), it returns to the main menu.

We now have our insertion operations. addNode adds a node after the current node. It allocates memory for a new node and initializes its pointers to 0. It reads the user’s input string and stores it in the node’s memory. It also updates the next and previous pointers of the involved nodes to insert the new node correctly, and handles special cases for adding after the last node. noNextNode handles the case where the current node is the last node, as it updates the next pointer of the current node and the previous pointer of the new node. insertHead inserts a node at the head of the list, as well as allocating memory for the new node and links it to the old head. It updates the head pointer ($s7) and the current pointer ($a3). insertTail inserts a node at the tail of the list, and traverses to the last node of the list using the next pointer. It also allocates memory for the new node and links it to the old tail.

We now move on the the deletion operations. delNode deletes the current node. It adjusts the next pointer of the previous node and the previous pointer of the next node to bypass the current node, and handles special cases for deleting the head or tail. delHead deletes the head node. It updates the head pointer ($s7) and the previous pointer of the new head. delTail deletes the tail node and updates the next pointer of the new tail node. resetList resets the list when all nodes are deleted and sets both the head pointer ($s7) and the current pointer ($a3) to 0.

We now have our printing operation. printEverything traverses the list starting from the head and prints the content of each node. It Uses the next pointer to move to the subsequent nodes until the end of the list.

We declare the first Node with declareFirstNode, which initializes the head and current pointers when the first node is added to an empty list.

We then exit our program with syscall.

This is an explanation of the code in the program.

<u>Section 3: Important Subroutines</u>

The important subroutines in our code are listed below: addNode, insertHead, insertTail, delNode, printEverything, and search.

addNode inserts a new node after the current node. It updates the next pointer of the current node to point to the new node, updates the previous pointer of the new node to point to the current node, and handles the case where the current node is the tail of the list.

insertHead adds a new node at the beginning of the list. It updates the new node's next pointer to point to the current head and updates the old head's previous pointer to point to the new node.

insertTail traverses to the end of the list and adds a new node. It updates the next pointer of the old tail to point to the new node and updates the new node's previous pointer to point to the old tail.

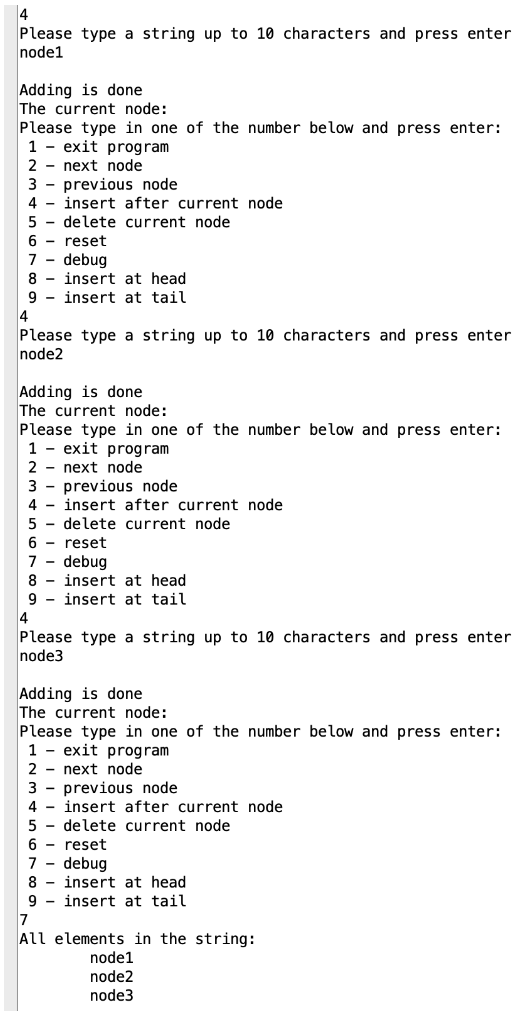
delNode deletes the current node. It updates the next pointer of the previous node and the previous pointer of the next node to bypass the current node and has special handling for deleting the head or tail nodes.

printEverything prints the entire list starting from the head, and iterates using the next pointer until the end of the list.

Search locates a node containing a specific character. It starts from the head and iterates using the next pointer, and prints the address of the found node or a "not found" message.

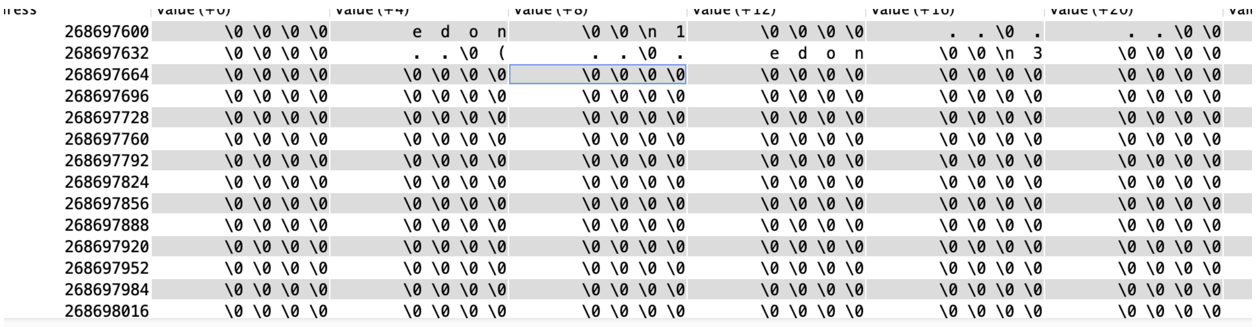
Section 4: Testing

Add Node after Current Node



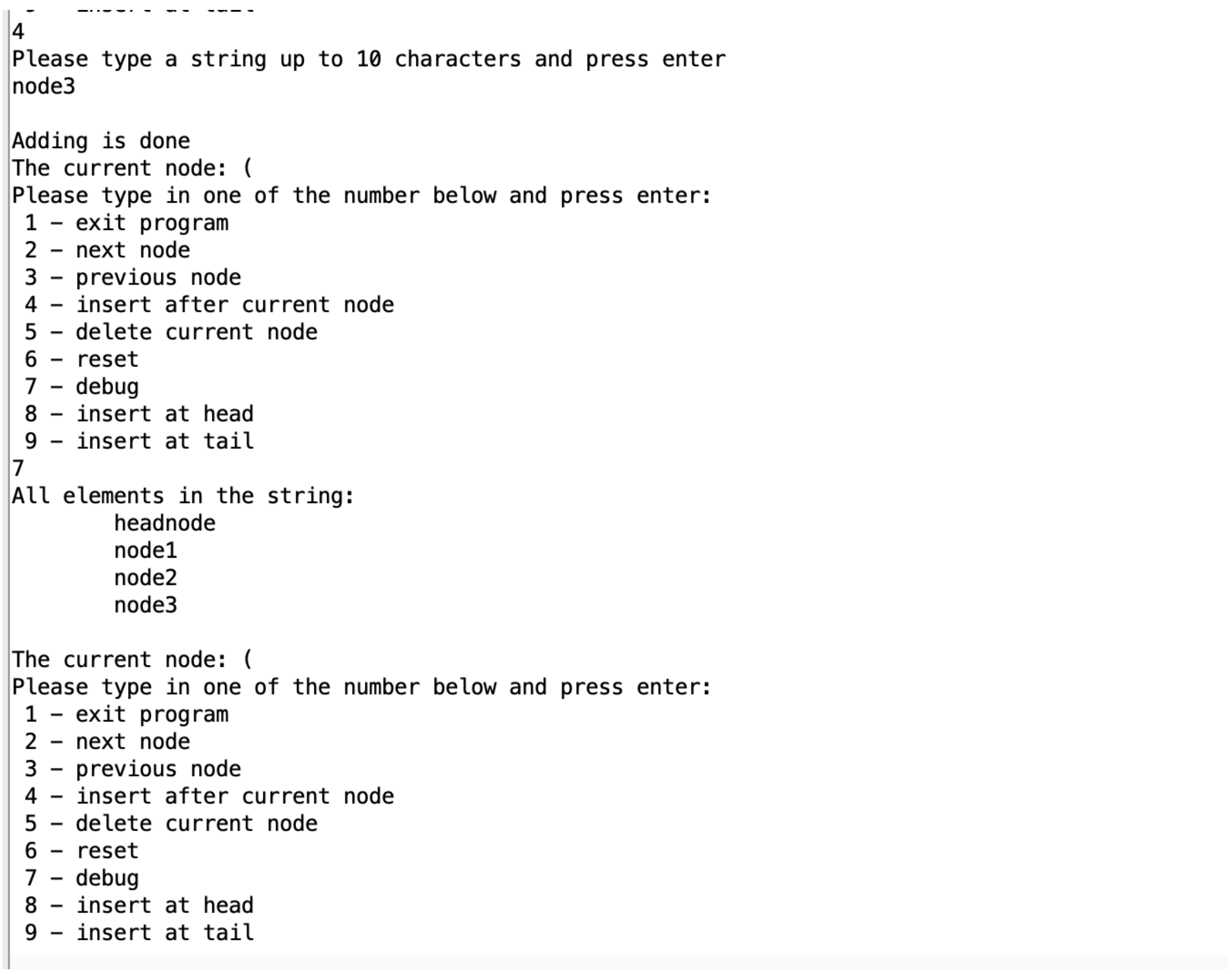
```
4
Please type a string up to 10 characters and press enter
node1

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
4
Please type a string up to 10 characters and press enter
node2

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
4
Please type a string up to 10 characters and press enter
node3

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
7
All elements in the string:
        node1
        node2
        node3
```

Traverse Forward and Backward

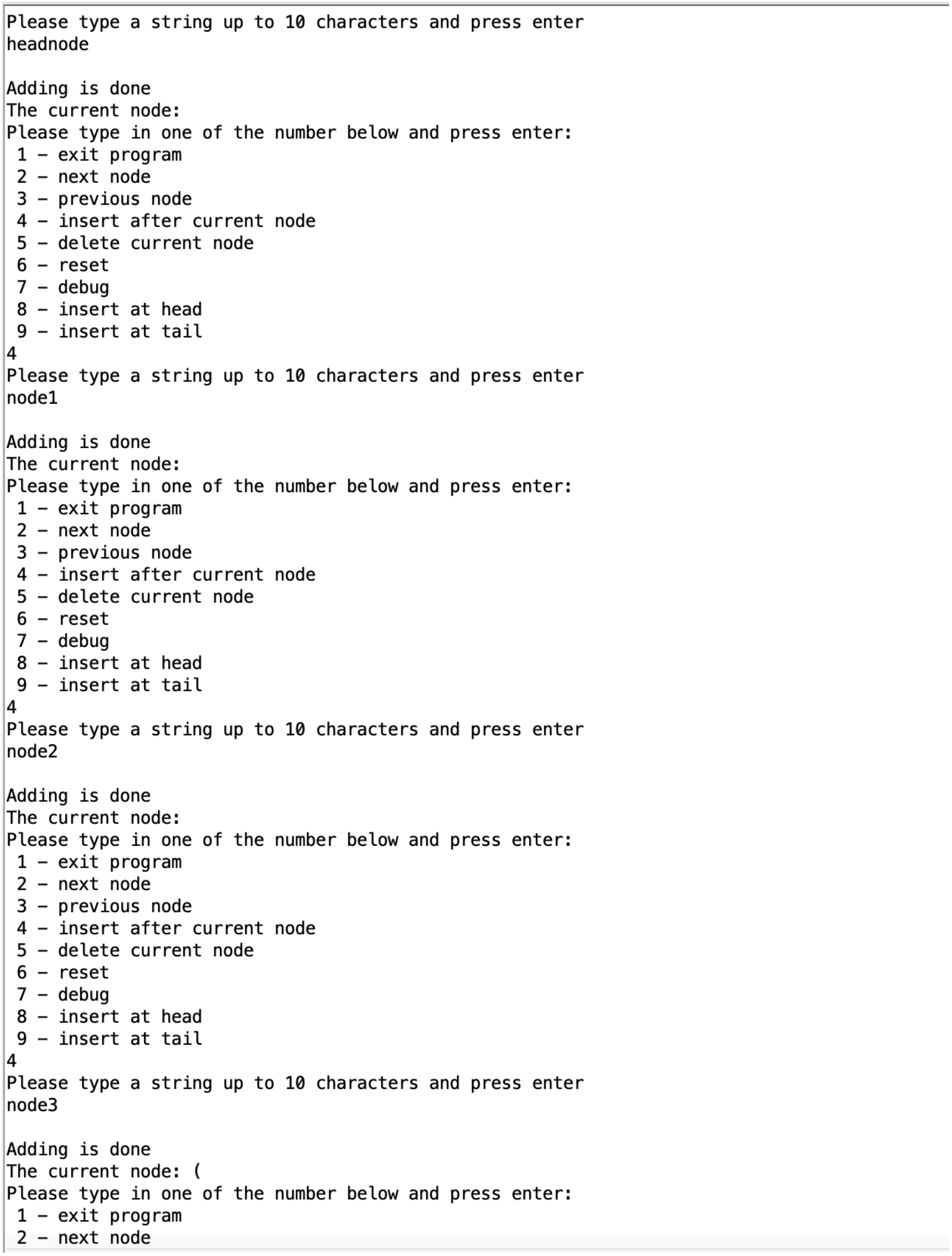


```
ress       Value (+0)    Value (+4)    Value (+8)    Value (+12)   Value (+16)   Value (+20)   Val
 268697600  \0 \0 \0 \0      e  d  o  n  \0 \0 \n  1   \0 \0 \0 \0      .  . \0  .      .  . \0 \0
 268697632  \0 \0 \0 \0      .  . \0  (      .  . \0  .    e  d  o  n  \0 \0 \n  3   \0 \0 \0 \0
 268697664  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268697696  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268697728  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268697760  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268697792  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268697824  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268697856  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268697888  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268697920  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268697952  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268697984  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
 268698016  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0  \0 \0 \0 \0
```

Insert at Head

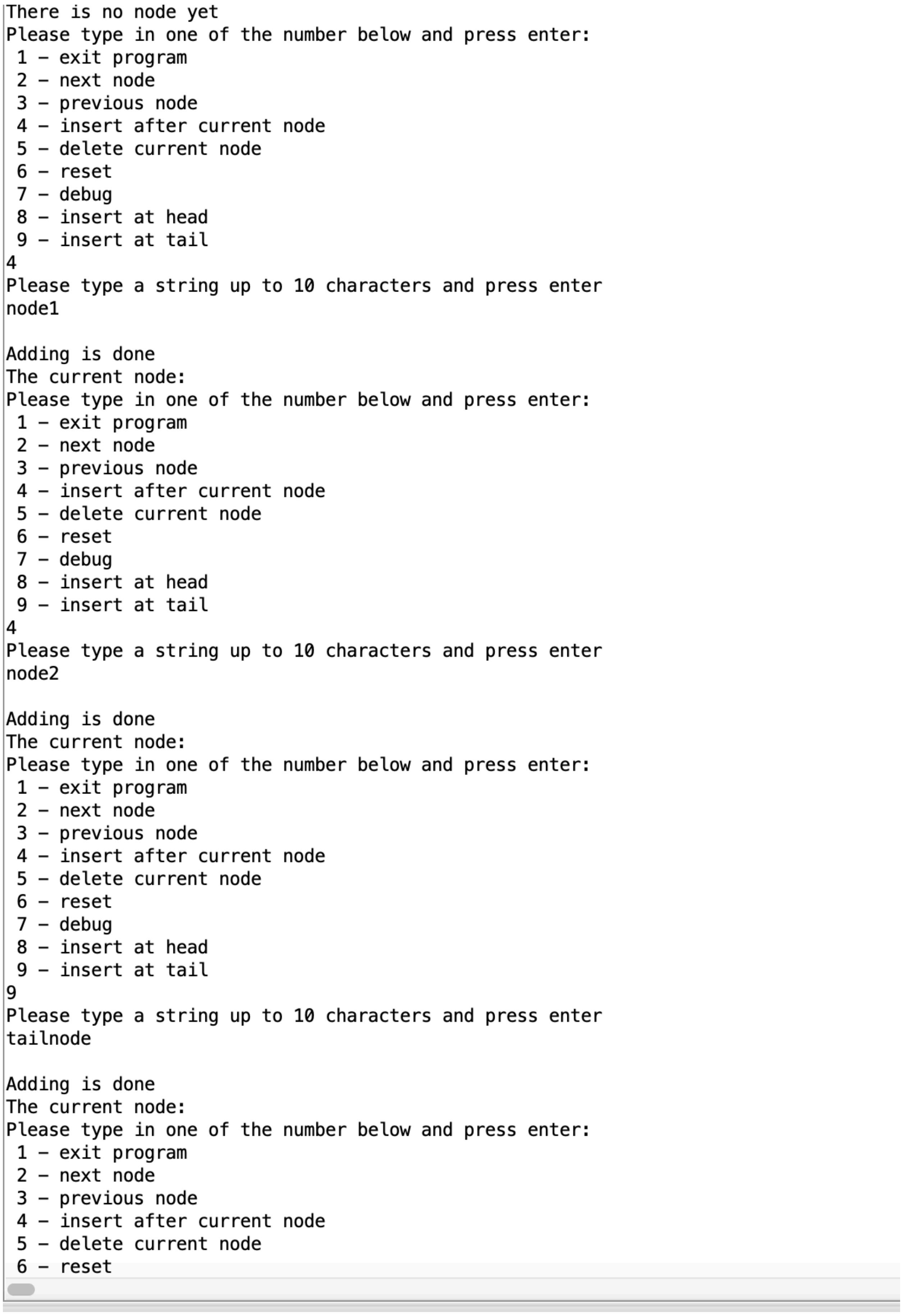


```
4
Please type a string up to 10 characters and press enter
node3

Adding is done
The current node: (
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
7
All elements in the string:
        headnode
        node1
        node2
        node3

The current node: (
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
```



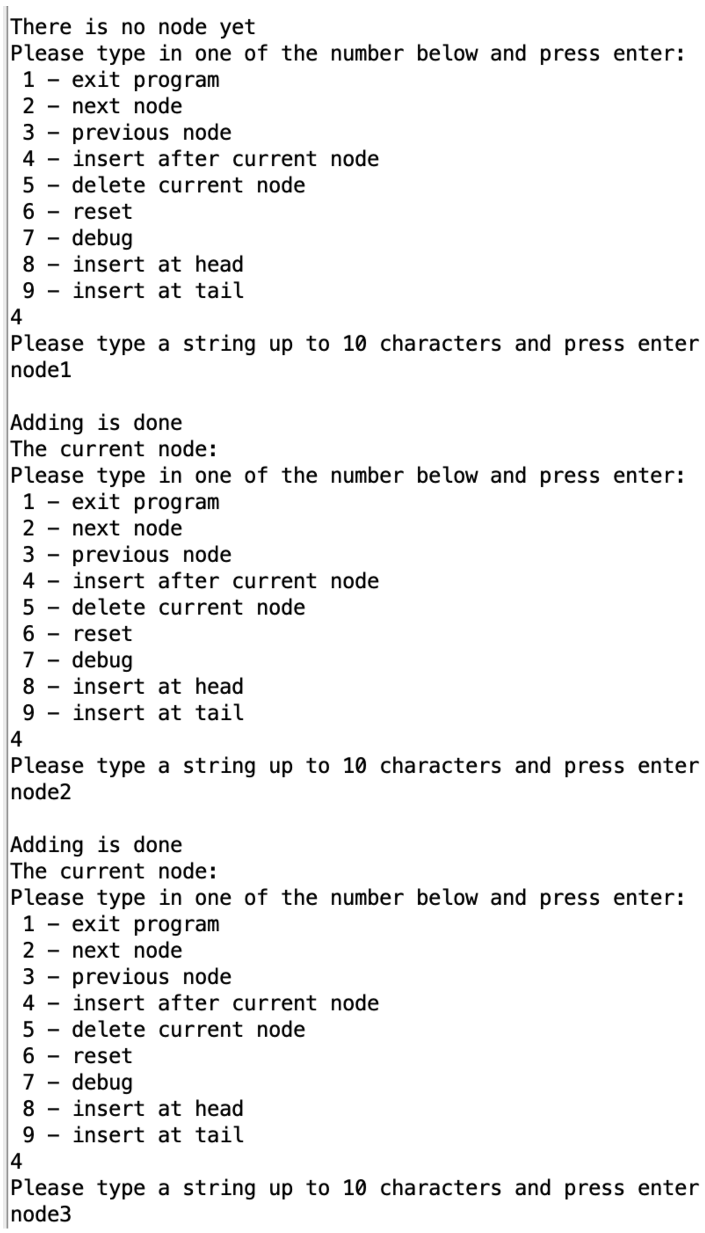
```
Please type a string up to 10 characters and press enter
headnode

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
4
Please type a string up to 10 characters and press enter
node1

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
4
Please type a string up to 10 characters and press enter
node2

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
4
Please type a string up to 10 characters and press enter
node3

Adding is done
The current node: (
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
```

Insert at Tail

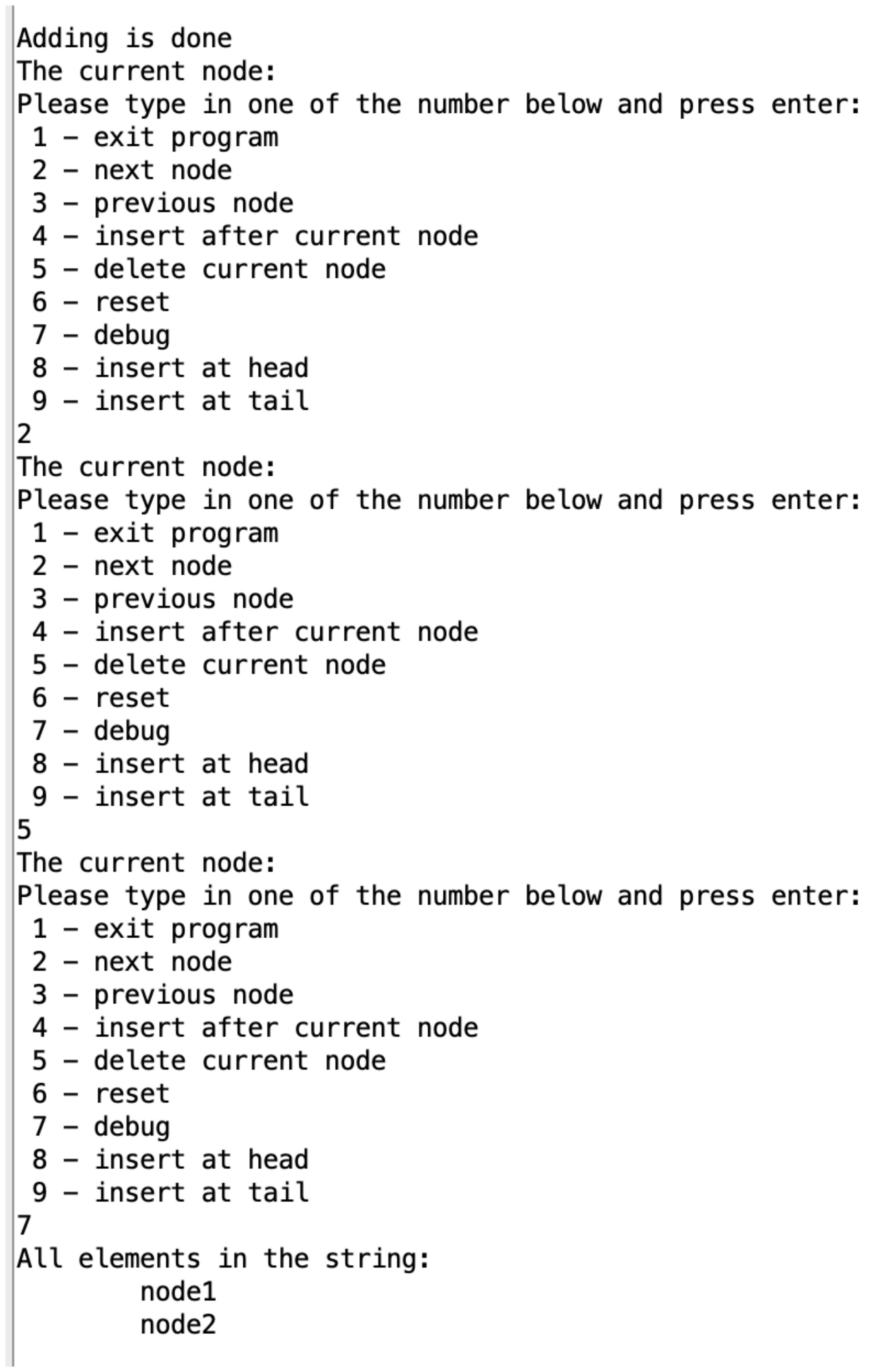


```
There is no node yet
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
4
Please type a string up to 10 characters and press enter
node1

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
4
Please type a string up to 10 characters and press enter
node2

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
9
Please type a string up to 10 characters and press enter
tailnode

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
7
All elements in the string:
        node1
        node2
        tailnode

The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
```

Delete Current Node



```
There is no node yet
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
4
Please type a string up to 10 characters and press enter
node1

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
4
Please type a string up to 10 characters and press enter
node2

Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
4
Please type a string up to 10 characters and press enter
node3
```



```
Adding is done
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
2
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
5
The current node:
Please type in one of the number below and press enter:
 1 - exit program
 2 - next node
 3 - previous node
 4 - insert after current node
 5 - delete current node
 6 - reset
 7 - debug
 8 - insert at head
 9 - insert at tail
7
All elements in the string:
        node1
        node2
```

Section 5: Conclusion

The project successfully transformed a singly linked list into a doubly linked list in MIPS Assembly. The updated implementation supports bidirectional traversal and provides additional operations for insertion, deletion, and debugging. This project demonstrates the efficient use of pointers and memory management in assembly programming.