Doubly linked list

Bello Melido

CSC 210

September 25, 2023

Table of contents.

Introduction:

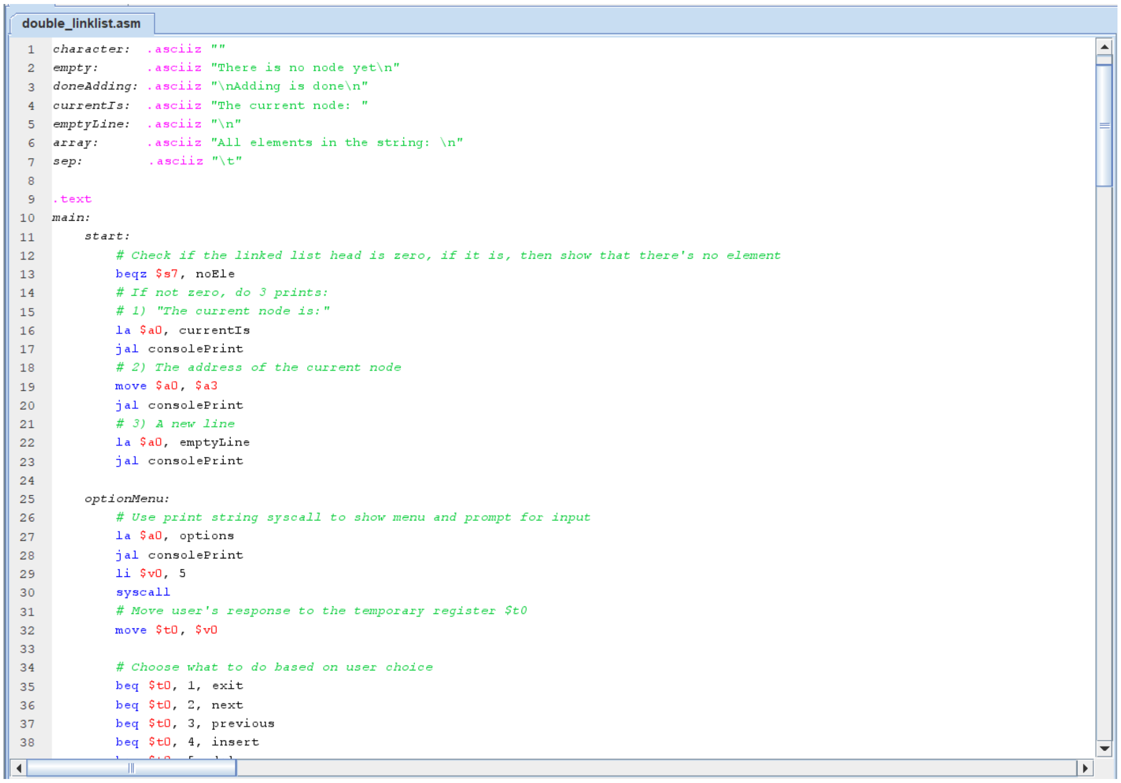
The provided MIPS assembly code is designed to create a user-friendly interactive program for managing a list of text entries, similar to a to-do list. Users can perform actions like adding new entries, deleting existing ones, moving between entries, and more. The program communicates with users through text menus, making it easy to understand and use.

Objective:

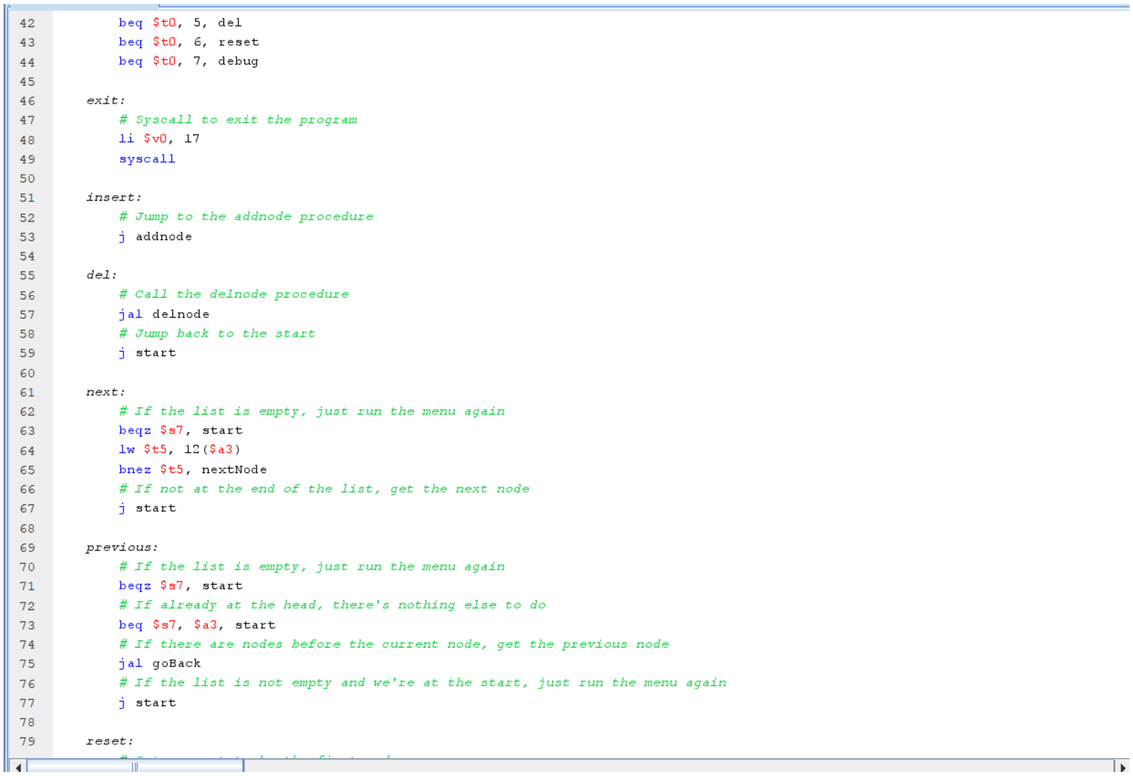
The main goal of this MIPS assembly code is to demonstrate the creation of a simple program for managing a list of text entries. The program aims to achieve the following objectives:

1. Create and manage a list of text entries, somewhat like a to-do list.
2. Offer a menu-based interface that allows users to:
   - Add new text entries to the list.
   - Delete the currently selected entry.
   - Navigate between entries, moving forward or backward.
   - Reset to the beginning of the list.
   - See all the entries in the list for debugging purposes.
   - Exit the program when finished.
3. Handle special cases, such as an empty list or actions on the first or last entry.
4. Allocate memory efficiently for new entries.
5. Provide clear and helpful messages to guide users during interaction.

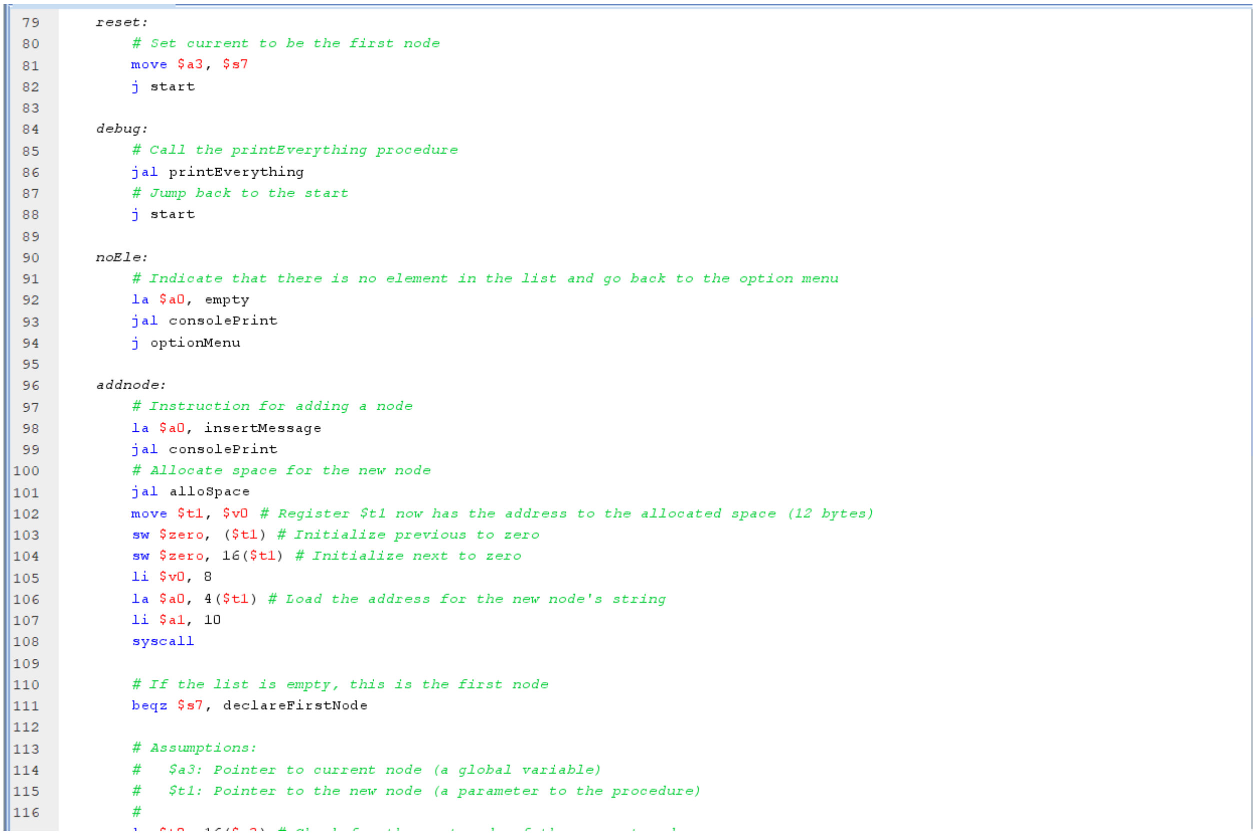
Code screenshot



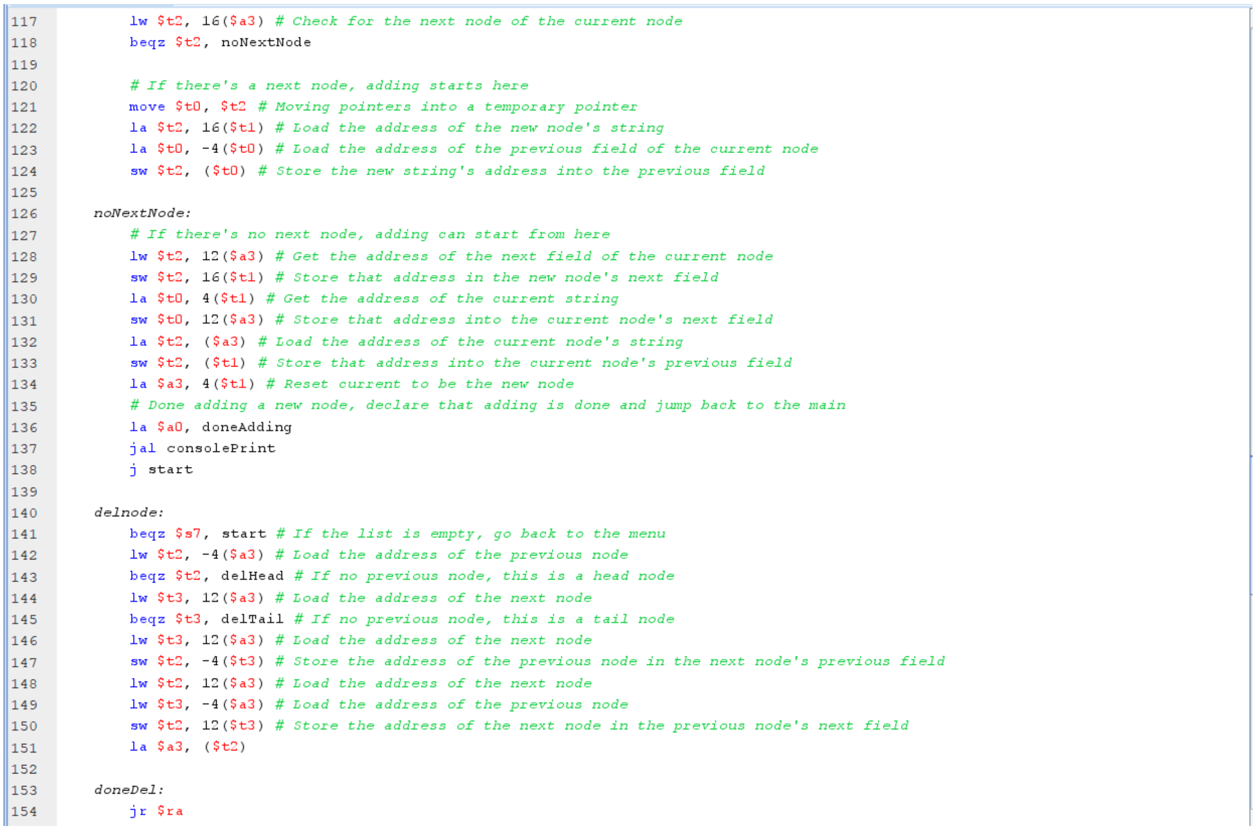
```
character:  .asciiz ""
empty:      .asciiz "There is no node yet\n"
doneAdding: .asciiz "\nAdding is done\n"
currentIs:  .asciiz "The current node: "
emptyLine:  .asciiz "\n"
array:      .asciiz "All elements in the string: \n"
sep:        .asciiz "\t"

.text
main:
    start:
        # Check if the linked list head is zero, if it is, then show that there's no element
        beqz $s7, noEle
        # If not zero, do 3 prints:
        # 1) "The current node is:"
        la $a0, currentIs
        jal consolePrint
        # 2) The address of the current node
        move $a0, $a3
        jal consolePrint
        # 3) A new line
        la $a0, emptyLine
        jal consolePrint

    optionMenu:
        # Use print string syscall to show menu and prompt for input
        la $a0, options
        jal consolePrint
        li $v0, 5
        syscall
        # Move user's response to the temporary register $t0
        move $t0, $v0

        # Choose what to do based on user choice
        beq $t0, 1, exit
        beq $t0, 2, next
        beq $t0, 3, previous
        beq $t0, 4, insert
```



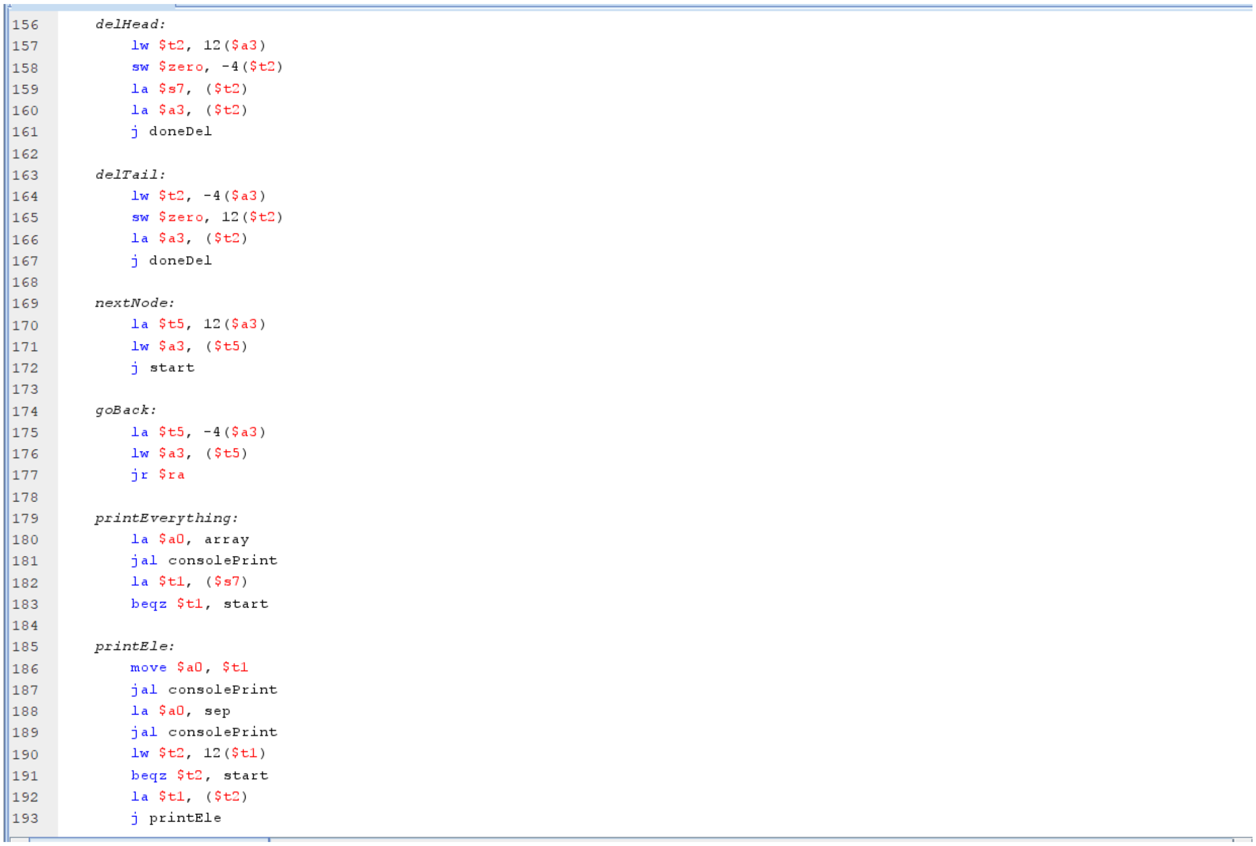
```
        beq $t0, 5, del
        beq $t0, 6, reset
        beq $t0, 7, debug

    exit:
        # Syscall to exit the program
        li $v0, 17
        syscall

    insert:
        # Jump to the addnode procedure
        j addnode

    del:
        # Call the delnode procedure
        jal delnode
        # Jump back to the start
        j start

    next:
        # If the list is empty, just run the menu again
        beqz $s7, start
        lw $t5, 12($a3)
        bnez $t5, nextNode
        # If not at the end of the list, get the next node
        j start

    previous:
        # If the list is empty, just run the menu again
        beqz $s7, start
        # If already at the head, there's nothing else to do
        beq $s7, $a3, start
        # If there are nodes before the current node, get the previous node
        jal goBack
        # If the list is not empty and we're at the start, just run the menu again
        j start

    reset:
```



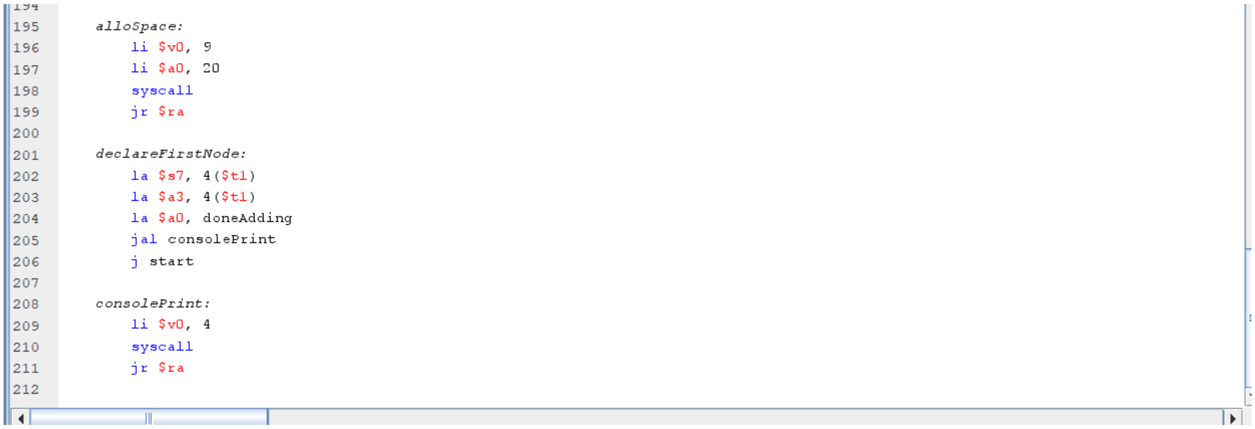
```
    reset:
        # Set current to be the first node
        move $a3, $s7
        j start

    debug:
        # Call the printEverything procedure
        jal printEverything
        # Jump back to the start
        j start

    noEle:
        # Indicate that there is no element in the list and go back to the option menu
        la $a0, empty
        jal consolePrint
        j optionMenu

    addnode:
        # Instruction for adding a node
        la $a0, insertMessage
        jal consolePrint
        # Allocate space for the new node
        jal alloSpace
        move $t1, $v0 # Register $t1 now has the address to the allocated space (12 bytes)
        sw $zero, ($t1) # Initialize previous to zero
        sw $zero, 16($t1) # Initialize next to zero
        li $v0, 8
        la $a0, 4($t1) # Load the address for the new node's string
        li $a1, 10
        syscall

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

        # Assumptions:
        #   $a3: Pointer to current node (a global variable)
        #   $t1: Pointer to the new node (a parameter to the procedure)
        #
```



```
        lw $t2, 16($a3) # Check for the next node of the current node
        beqz $t2, noNextNode

        # If there's a next node, adding starts here
        move $t0, $t2 # Moving pointers into a temporary pointer
        la $t2, 16($t1) # Load the address of the new node's string
        la $t0, -4($t0) # Load the address of the previous field of the current node
        sw $t2, ($t0) # Store the new string's address into the previous field

    noNextNode:
        # If there's no next node, adding can start from here
        lw $t2, 12($a3) # Get the address of the next field of the current node
        sw $t2, 16($t1) # Store that address in the new node's next field
        la $t0, 4($t1) # Get the address of the current string
        sw $t0, 12($a3) # Store that address into the current node's next field
        la $t2, ($a3) # Load the address of the current node's string
        sw $t2, ($t1) # Store that address into the current node's previous field
        la $a3, 4($t1) # Reset current to be the new node
        # Done adding a new node, declare that adding is done and jump back to the main
        la $a0, doneAdding
        jal consolePrint
        j start

    delnode:
        beqz $s7, start # If the list is empty, go back to the menu
        lw $t2, -4($a3) # Load the address of the previous node
        beqz $t2, delHead # If no previous node, this is a head node
        lw $t3, 12($a3) # Load the address of the next node
        beqz $t3, delTail # If no previous node, this is a tail node
        lw $t3, 12($a3) # Load the address of the next node
        sw $t2, -4($t3) # Store the address of the previous node in the next node's previous field
        lw $t2, 12($a3) # Load the address of the next node
        lw $t3, -4($a3) # Load the address of the previous node
        sw $t2, 12($t3) # Store the address of the next node in the previous node's next field
        la $a3, ($t2)

    doneDel:
        jr $ra
```

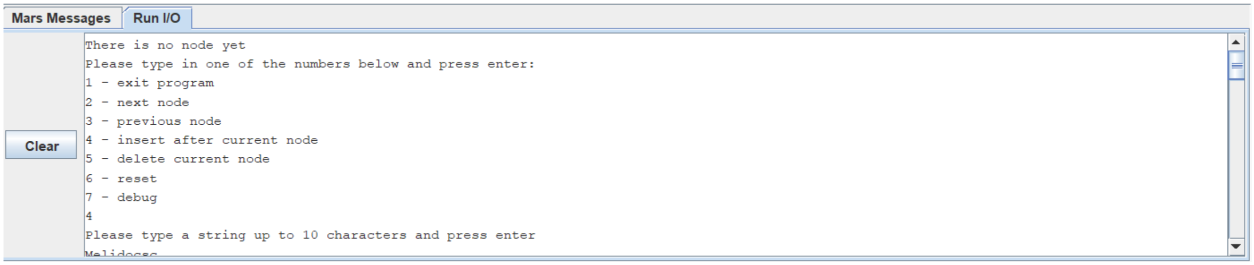


```
    delHead:
        lw $t2, 12($a3)
        sw $zero, -4($t2)
        la $s7, ($t2)
        la $a3, ($t2)
        j doneDel

    delTail:
        lw $t2, -4($a3)
        sw $zero, 12($t2)
        la $a3, ($t2)
        j doneDel

    nextNode:
        la $t5, 12($a3)
        lw $a3, ($t5)
        j start

    goBack:
        la $t5, -4($a3)
        lw $a3, ($t5)
        jr $ra

    printEverything:
        la $a0, array
        jal consolePrint
        la $t1, ($s7)
        beqz $t1, start

    printEle:
        move $a0, $t1
        jal consolePrint
        la $a0, sep
        jal consolePrint
        lw $t2, 12($t1)
        beqz $t2, start
        la $t1, ($t2)
        j printEle
```

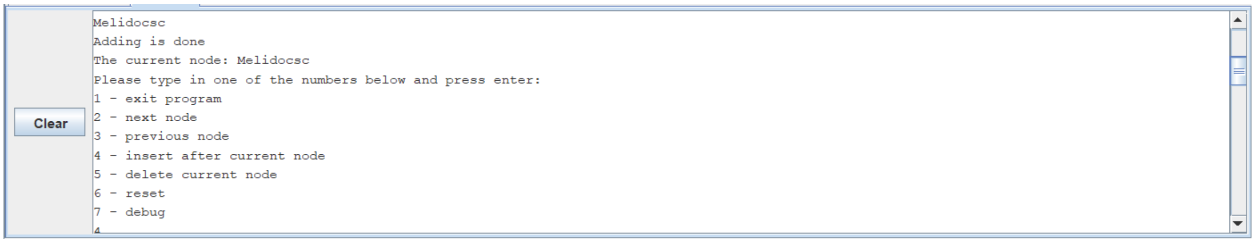


```

    alloSpace:
        li $v0, 9
        li $a0, 20
        syscall
        jr $ra

    declareFirstNode:
        la $s7, 4($t1)
        la $a3, 4($t1)
        la $a0, doneAdding
        jal consolePrint
        j start

    consolePrint:
        li $v0, 4
        syscall
        jr $ra

```

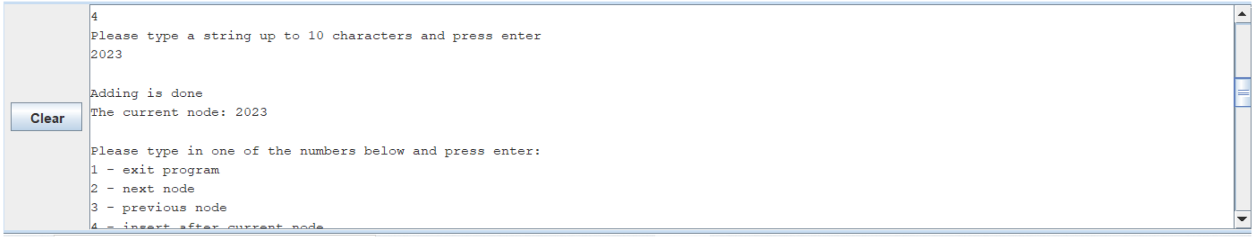
Results



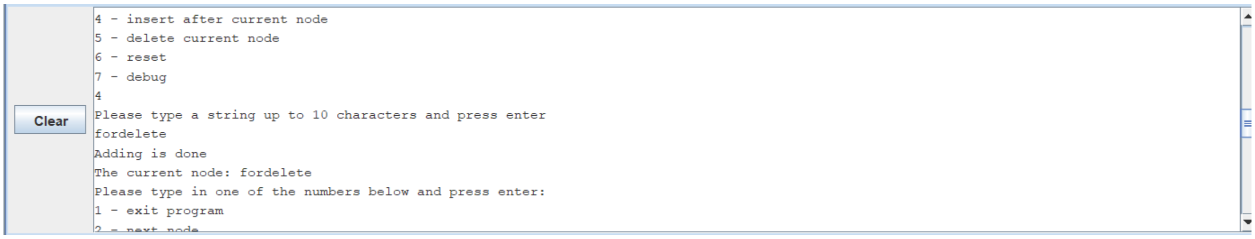
```
There is no node yet
Please type in one of the numbers below and press enter:
1 - exit program
2 - next node
3 - previous node
4 - insert after current node
5 - delete current node
6 - reset
7 - debug
4
Please type a string up to 10 characters and press enter
Melidocsc
```



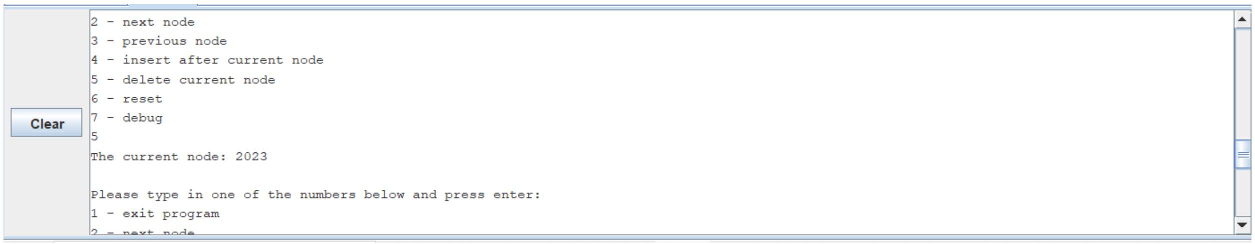
```
Melidocsc
Adding is done
The current node: Melidocsc
Please type in one of the numbers below and press enter:
1 - exit program
2 - next node
3 - previous node
4 - insert after current node
5 - delete current node
6 - reset
7 - debug
4
```



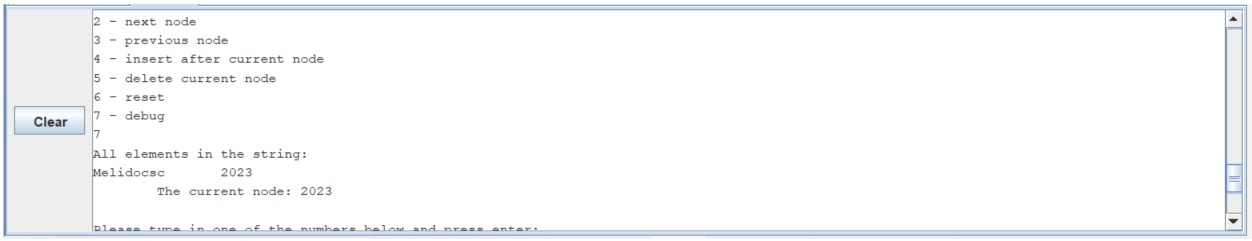
```
4
Please type a string up to 10 characters and press enter
2023

Adding is done
The current node: 2023

Please type in one of the numbers below and press enter:
1 - exit program
2 - next node
3 - previous node
4 - insert after current node
```



```
4 - insert after current node
5 - delete current node
6 - reset
7 - debug
4
Please type a string up to 10 characters and press enter
fordelete
Adding is done
The current node: fordelete
Please type in one of the numbers below and press enter:
1 - exit program
2 - next node
```



```
2 - next node
3 - previous node
4 - insert after current node
5 - delete current node
6 - reset
7 - debug
5
The current node: 2023

Please type in one of the numbers below and press enter:
1 - exit program
2 - next node
```



```
2 - next node
3 - previous node
4 - insert after current node
5 - delete current node
6 - reset
7 - debug
7
All elements in the string:
Melidocsc       2023
        The current node: 2023

Please type in one of the numbers below and press enter:
```

Explanation of results

The program starts by informing the user that there are no nodes (data elements) in the list yet.

It presents a menu of options, each represented by a number from 1 to 7. These options are as follows:

Option 1: Exit the program.

Option 2: Move to the next node in the list.

Option 3: Move to the previous node in the list.

Option 4: Insert a new node after the current node.

Option 5: Delete the current node.

Option 6: Reset the current node to the beginning of the list.

Option 7: Display a debug view of all elements in the list.

The user selects option 4 to insert a new node. They are prompted to enter a string of up to 10 characters. In this case, they enter "Melidocsc."

After entering the string, the program confirms that the addition is done and sets the current node's data to "Melidocsc."

The menu is shown again, and the user selects option 4 to insert another node. This time, they enter "2023."

Similar to before, the program confirms the addition and updates the current node's data to "2023."

The menu is shown once more, and the user selects option 4 again to insert a third node, this time entering "fordelete."

The program confirms the addition and updates the current node's data to "fordelete."

The user decides to delete the current node by choosing option 5. The program removes "2023" from the list.

The menu is displayed again, and the user selects option 7 to view all elements in the list. It shows "Melidocsc" and "fordelete," separated by tabs. The current node is indicated as "2023."

The menu continues to be displayed, allowing the user to perform more actions or exit the program.

Conclusion

In conclusion, the provided code represents a simple interactive program that manages a linked list of nodes, each containing a string of up to 10 characters. The program offers a user-friendly menu with options to insert new nodes, delete the current node, navigate through the list, reset the current position, view the entire list, and exit the program. Users can input strings, which are added as nodes to the list. The program provides clear feedback for each operation, such as confirming the addition of a new node and displaying the current node's data. It also handles scenarios where there are no nodes in the list or when deleting a node.