STRING FUNCTION

USING

STACK AND LINKED LIST

SUPERVISED BY

DR. Lamia ALREFAAI

Nayra Fouad Ahmed	231903617
Ahmed Elsayed Ahmed	231903635
Omar Mostafa Salem	231903646
Mohamed Ashraf Eldesoky	231903632
Youssef Waheed	231903716
Yehia Mohamed Sadek	231903599
Adham Mohamed Ayad	231903571

ELECTRICAL ENGINEERING
COMMUNICATION AND COMPUTER ENGINEERING
CCE 307-ELE251 – COURSE PROJECT
(TERM 242)

Course Project cover page

S #	Student Name	Edu Email	Student ID	Marks			
				Report & Slides (30)	Implementation (50)	Presentation (20)	Total (100)
1	Nayra Fouad Ahmed	naira402938@ feng.bu.edu.eg	231903617				
2	Ahmed Elsayed Ahmed	ahmed402811@feng.bu.edu.eg	231903635				
3	Omar Mostafa Salem	omar402801@feng.bu.edu.eg	231903646				
4	Mohamed Ashraf Eldesoky	mohamed402784@feng.bu.edu.eg	231903632				
5	Youssef Waheed	youssef402800@ feng.bu.edu.eg	231903716				
6	Yehia Mohamed Sadek	yehia403020@ feng.bu.edu.eg	231903599				
7	Adham Mohamed Ayad	adham402503@ feng.bu.edu.eg	231903571				

Date handed in: / / 2024

Chapter one
“Stack”

Introduction

This program we implement a stack and its functions and palindrome checkup program and some String functions.

What is the stack is: A stack is a data structure that follows the Last In, First Out **(LIFO)** principle. Think of it like a stack of plates — you can only add or remove the top plate. Stacks are commonly used in programming for managing function calls, expression evaluation, and undo mechanisms.

In this project we implement a stack use function of stack like pop, push, peek,.....
We make a program to check a string palindrome or not.

THE C CODE

```
    #include <stdio.h>
#include <string.h>
#include <ctype.h>

#define STACK_SIZE 100
#define MAX_STRING_LENGTH 64

char array[STACK_SIZE];
char string1[MAX_STRING_LENGTH];
char string2[MAX_STRING_LENGTH];

const char *correct = "The word is a palindrome";
const char *error = "The word is not a palindrome";
const char *newline = "\n";
const char *options = "Please type in one of the number below and press enter: \n1 - Exit program \n2 - IsFull \n3 - IsEmpty \n4 - Peek \n5 - push \n6 - Pop \n7 - Convert the word to lowercase\n8 - Convert the word to Uppercase \n9 - Reverse the word \n10 - Check the word is palindrome \n11 - Reset\n";
const char *sentince = "Please enter the word\n";
const char *sentince2 = "THANK YOU:)\n";
const char *sentince3 = "Not empty\n";
const char *sentince4 = "Stack is empty\n";
const char *sentince5 = "Stack is full\n";
const char *sentince6 = "Stack still has free space\n";
const char *sentince7 = "Please insert a character\n";
const char *peekvalue = "The peek value in the stack is \n";
const char *resetmsg = "Stack reseted\n";
const char *sentincecheck = "Please enter the word you want to check\n";

int stack_pointer = 0;
```

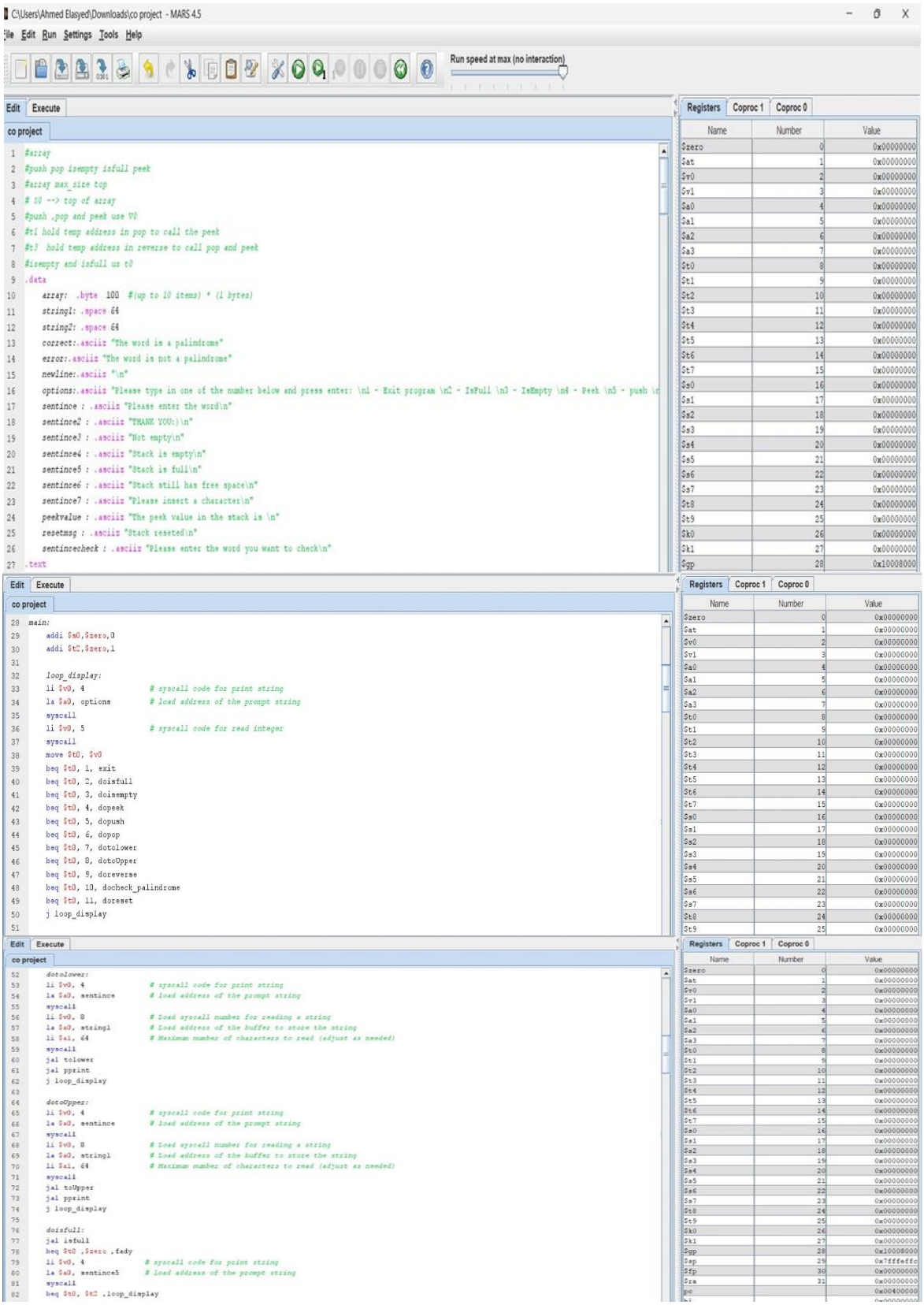
```
void print_string(const char *str) {
    printf("%s", str);
}

void read_string(char *buffer, int max_length) {
    fgets(buffer, max_length, stdin);
    // Remove newline character if present
    buffer[strcspn(buffer, "\n")] = '\0';
}

void push(char value) {
    if (stack_pointer < STACK_SIZE) {
        array[stack_pointer++] = value;
    }
}

char pop() {
    if (stack_pointer > 0) {
        return array[--stack_pointer];
    }
    return '\0'; // Empty stack
}

char peek() {
    if (stack_pointer > 0) {
        return array[stack_pointer - 1];
    }
    return '\0'; // Empty stack
}

int is_empty() {
    return stack_pointer == 0;
}

int is_full() {
    return stack_pointer == STACK_SIZE;
}

void reset_stack() {
    stack_pointer = 0;
}

void convert_to_lowercase(char *str) {
    for (int i = 0; str[i] != '\0'; i++) {
        str[i] = tolower(str[i]);
    }
}

void convert_to_uppercase(char *str) {
    for (int i = 0; str[i] != '\0'; i++) {
```

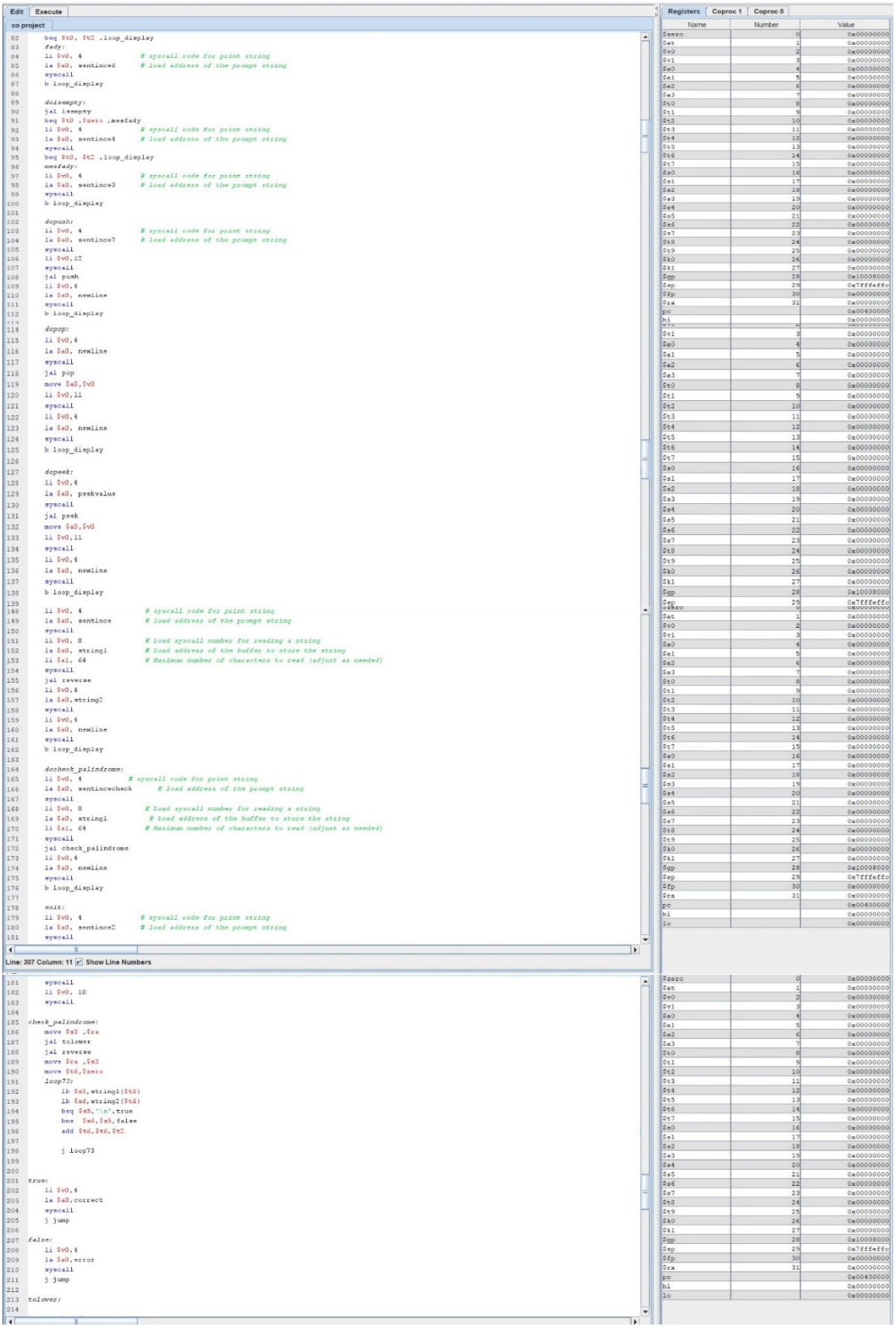
```
        str[i] = toupper(str[i]);
    }
}

void reverse_string(const char *input, char *output) {
    int length = strlen(input);
    for (int i = 0; i < length; i++) {
        output[i] = input[length - 1 - i];
    }
    output[length] = '\0';
}

int check_palindrome(const char *str) {
    int length = strlen(str);
    for (int i = 0; i < length / 2; i++) {
        if (str[i] != str[length - 1 - i]) {
            return 0; // Not a palindrome
        }
    }
    return 1; // Palindrome
}

int main() {
    int option;
    char input_char;

    while (1) {
        print_string(options);
        scanf("%d", &option);
        getchar(); // Consume newline character

        switch (option) {
            case 1:
                print_string(sentince2);
                return 0;
            case 2:
                if (is_full()) {
                    print_string(sentince5);
                } else {
                    print_string(sentince6);
                }
                break;
            case 3:
                if (is_empty()) {
                    print_string(sentince4);
                } else {
                    print_string(sentince3);
                }
                break;
            case 4:
```

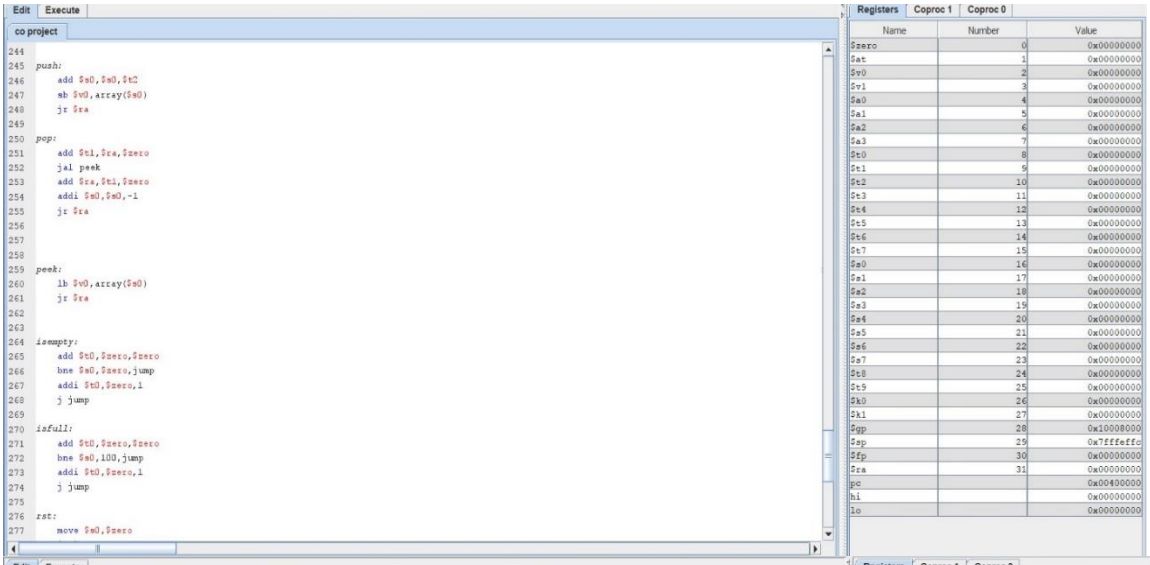
```
            print_string(peekvalue);
            printf("%c\n", peek());
            break;
        case 5:
            print_string(sentince7);
            input_char = getchar();
            getchar(); // Consume newline character
            push(input_char);
            break;
        case 6:
            printf("%c\n", pop());
            break;
        case 7:
            print_string(sentince);
            read_string(string1, MAX_STRING_LENGTH);
            convert_to_lowercase(string1);
            print_string(string1);
            print_string(newline);
            break;
        case 8:
            print_string(sentince);
            read_string(string1, MAX_STRING_LENGTH);
            convert_to_uppercase(string1);
            print_string(string1);
            print_string(newline);
            break;
        case 9:
            print_string(sentince);
            read_string(string1, MAX_STRING_LENGTH);
            reverse_string(string1, string2);
            print_string(string2);
            print_string(newline);
            break;
        case 10:
            print_string(sentincecheck);
            read_string(string1, MAX_STRING_LENGTH);
            if (check_palindrome(string1)) {
                print_string(correct);
            } else {
                print_string(error);
            }
            print_string(newline);
            break;
        case 11:
            reset_stack();
            print_string(resetmsg);
            break;
        default:
            break;
    }
```

```
}
return 0;}
```

ASSEMBLY CODE

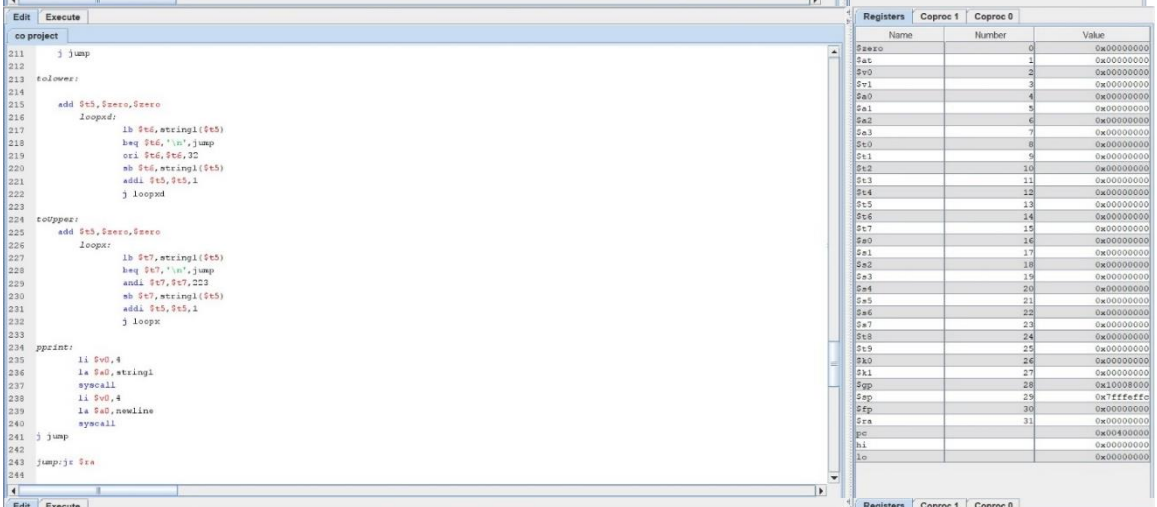



Edit
Execute
co project
beq $t0, $t2 ,loop_display
fady:
li $v0, 4 # syscall code for print string
la $a0, sentince6 # load address of the prompt string
syscall
b loop_display
doisempty:
jal isempty
beq $t0 ,$zero ,mesfady
li $v0, 4 # syscall code for print string
la $a0, sentince4 # load address of the prompt string
syscall
beq $t0, $t2 ,loop_display
mesfady:
li $v0, 4 # syscall code for print string
la $a0, sentince3 # load address of the prompt string
syscall
b loop_display
dopush:
li $v0, 4 # syscall code for print string
la $a0, sentince7 # load address of the prompt string
syscall
li $v0,12
syscall
jal push
li $v0,4
la $a0, newline
syscall
b loop_display
dopop:
li $v0,4
la $a0, newline
syscall
jal pop
move $a0,$v0
li $v0,11
syscall
li $v0,4
la $a0, newline
syscall
b loop_display
dopeek:
li $v0,4
la $a0, peekvalue
syscall
jal peek
move $a0,$v0
li $v0,11
syscall
li $v0,4
la $a0, newline
syscall
b loop_display
li $v0, 4 # syscall code for print string
la $a0, sentince # load address of the prompt string
syscall
li $v0, 8 # load syscall number for reading a string
la $a0, string1 # load address of the buffer to store the string
li $a1, 64 # Maximum number of characters to read (adjust as needed)
syscall
jal reverse
li $v0,4
la $a0,string2
syscall
li $v0,4
la $a0, newline
syscall
b loop_display
docheck_palindrome:
li $v0, 4 # syscall code for print string
la $a0, sentincecheck # load address of the prompt string
syscall
li $v0, 8 # load syscall number for reading a string
la $a0, string1 # load address of the buffer to store the string
li $a1, 64 # Maximum number of characters to read (adjust as needed)
syscall
jal check_palindrome
li $v0,4
la $a0, newline
syscall
b loop_display
exit:
li $v0, 4 # syscall code for print string
la $a0, sentince2 # load address of the prompt string
syscall
Line: 307 Column: 11
Show Line Numbers
syscall
li $v0, 10
syscall
check_palindrome:
move $s3 ,$ra
jal tolower
jal reverse
move $ra ,$s3
move $t6,$zero
loop73:
lb $s5,string1($t6)
lb $s6,string2($t6)
beq $s5,'\n',true
bne $s6,$s5,false
add $t6,$t6,$t2
j loop73
true:
li $v0,4
la $a0,correct
syscall
j jump
false:
li $v0,4
la $a0,error
syscall
j jump
tolower:
Registers
Coproc 1
Coproc 0
Name
Number
Value
$zero 0 0x00000000
$at 1 0x00000000
$v0 2 0x00000000
$v1 3 0x00000000
$a0 4 0x00000000
$a1 5 0x00000000
$a2 6 0x00000000
$a3 7 0x00000000
$t0 8 0x00000000
$t1 9 0x00000000
$t2 10 0x00000000
$t3 11 0x00000000
$t4 12 0x00000000
$t5 13 0x00000000
$t6 14 0x00000000
$t7 15 0x00000000
$s0 16 0x00000000
$s1 17 0x00000000
$s2 18 0x00000000
$s3 19 0x00000000
$s4 20 0x00000000
$s5 21 0x00000000
$s6 22 0x00000000
$s7 23 0x00000000
$t8 24 0x00000000
$t9 25 0x00000000
$k0 26 0x00000000
$k1 27 0x00000000
$gp 28 0x10008000
$sp 29 0x7fffeffc
$fp 30 0x00000000
$ra 31 0x00000000
pc 0x00400000
hi 0x00000000
lo 0x00000000



```
244
245 push:
246     add $s0,$s0,$t2
247     sb $v0,array($s0)
248     jr $ra
249
250 pop:
251     add $t1,$ra,$zero
252     jal peek
253     add $ra,$t1,$zero
254     addi $s0,$s0,-1
255     jr $ra
256
257
258
259 peek:
260     lb $v0,array($s0)
261     jr $ra
262
263
264 isempty:
265     add $t0,$zero,$zero
266     bne $s0,$zero,jump
267     addi $t0,$zero,1
268     j jump
269
270 isfull:
271     add $t0,$zero,$zero
272     bne $s0,100,jump
273     addi $t0,$zero,1
274     j jump
275
276 rst:
277     move $s0,$zero
```

Registers	Coproc 1	Coproc 0
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000



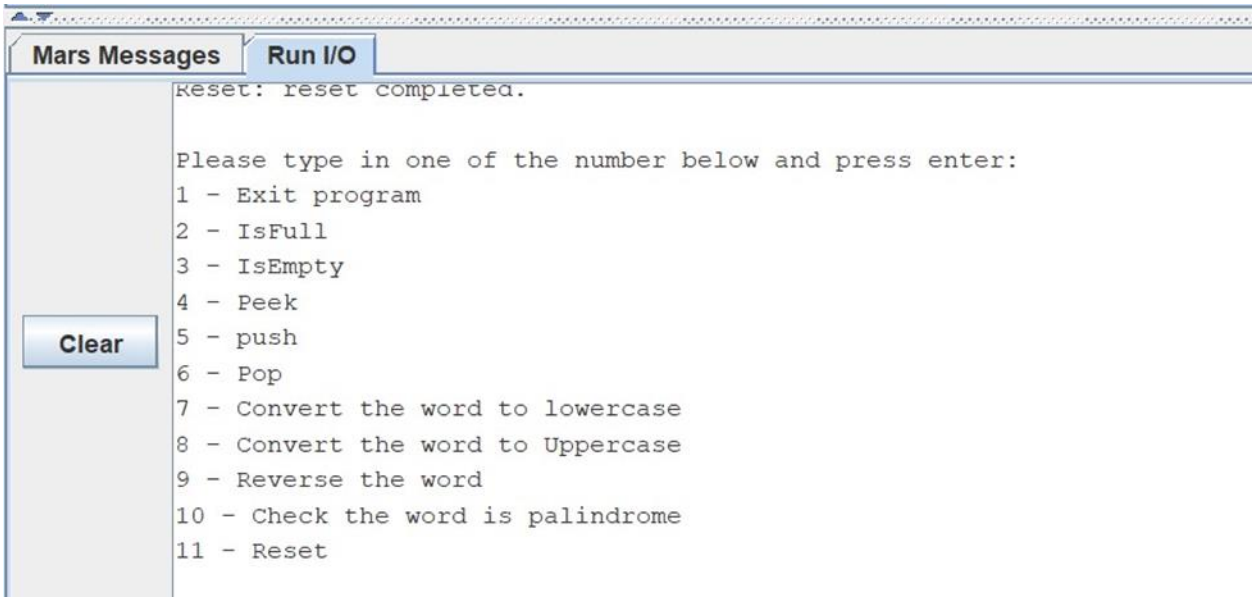
```
211     j jump
212
213 tolower:
214
215     add $t5,$zero,$zero
216         loopxd:
217                 lb $t6,string1($t5)
218                 beq $t6,'\n',jump
219                 ori $t6,$t6,32
220                 sb $t6,string1($t5)
221                 addi $t5,$t5,1
222                 j loopxd
223
224 toUpper:
225     add $t5,$zero,$zero
226         loopx:
227                 lb $t7,string1($t5)
228                 beq $t7,'\n',jump
229                 andi $t7,$t7,223
230                 sb $t7,string1($t5)
231                 addi $t5,$t5,1
232                 j loopx
233
234 pprint:
235         li $v0,4
236         la $a0,string1
237         syscall
238         li $v0,4
239         la $a0,newline
240         syscall
241 j jump
242
243 jump:jr $ra
244
```

Registers	Coproc 1	Coproc 0
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

```
274     j jump
275
276 rst:
277     move $s0,$zero
278     jr $ra
279
280
281 reverse:
282
283     move $t3,$ra
284     move $t6,$zero
285
286     loop1:
287         lb $s6,string1($t6)
288         beq $s6,'\n',print_2
289         move $v0,$s6
290         jal push
291         addi $t6,$t6,1
292         j loop1
293
294 print_2:
295     move $t6,$zero
296     loop2:
297         jal isempty
298         beq $t0,1,goto_jump
299         jal pop
300         sb $v0,string2($t6)
301         addi $t6,$t6,1
302     j loop2
303
304
305 goto_jump:move $ra,$t3
306     sb $zero,string2($t6)
307     jr $ra
```

Registers	Coproc 1	Coproc 0
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

RUN CODE



EXPLANATION OF FUNCTION

1- **Pop function**: The pop function is used in programming to remove and return the top element from a stack. It essentially "pops" off the last item that was added to the stack. This operation modifies the stack by removing the top element.

```
88
89      doisempty:
90      jal isempty
91      beq $t0 ,$zero ,mesfady
92      li $v0, 4               # syscall code for print string
93      la $a0, sentince4       # load address of the prompt string
94      syscall
95      beq $t0, $t2 ,loop_display
96      mesfady:
97      li $v0, 4               # syscall code for print string
98      la $a0, sentince3       # load address of the prompt string
99      syscall
100     b loop_display
101
113
114     dopop:
115     li $v0,4
116     la $a0, newline
117     syscall
118     jal pop
119     move $a0,$v0
120     li $v0,11
121     syscall
122     li $v0,4
123     la $a0, newline
124     syscall
125     b loop_display
```

```
6


M
```

2- **Push function :** The push function is used in programming to add an element to the top of a stack. It "pushes" the element onto the stack, making it the new top element. This operation modifies the stack by adding an element to it.

```
doisfull:
jal isfull
beq $t0 ,$zero ,fady
li $v0, 4               # syscall code for print string
la $a0, sentince5       # load address of the prompt string
syscall
beq $t0, $t2 ,loop_display
```

```
101
102     dopush:
103     li $v0, 4             # syscall code for print string
104     la $a0, sentince7     # load address of the prompt string
105     syscall
106     li $v0,12
107     syscall
108     jal push
109     li $v0,4
110     la $a0, newline
111     syscall
112     b loop_display
113
```

```
5
Please insert a character
M
```

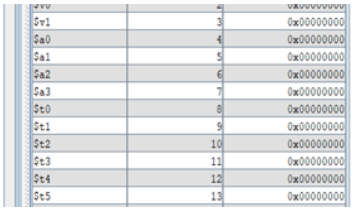
3-**peek function:** The peek function in programming allows you to view the top element of a stack without removing it. It's like taking a sneak peek at the top item without actually modifying the stack. This is useful for checking what the next element to be removed will be.

```
dopeek:
li $v0,4
la $a0, peekvalue
syscall
jal peek
move $a0,$v0
li $v0,11
syscall
li $v0,4
la $a0, newline
syscall
b loop_display
```

```
4
The peek value in the stack is
d
```

4-Lower case: Lower case refers to the set of alphabetical characters that are not capitalized. It includes letters from 'a' to 'z'. In programming, converting text to lower case often involves using a function or method to change all uppercase letters in a string to their corresponding lowercase counterparts.

```
tolower:

    add $t5,$zero,$zero
        loopxd:
                lb $t6,string1($t5)
                beq $t6,'\n',jump
                ori $t6,$t6,32
                sb $t6,string1($t5)
                addi $t5,$t5,1
                j loopxd
```



$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000

```
7
Please enter the word
MOHAMED
mohamed
```

5-upper case: refers to the set of alphabetical characters that are capitalized. It includes letters from 'A' to 'Z'. In programming, converting text to upper case typically involves using a function or method to change all lowercase letters in a string to their corresponding uppercase counterparts.

```
toUpper:
    add $t5,$zero,$zero
        loopx:
                lb $t7,string1($t5)
                beq $t7,'\n',jump
                andi $t7,$t7,223
                sb $t7,string1($t5)
                addi $t5,$t5,1
                j loopx
```

```
8
Please enter the word
mohamed
MOHAMED
```

6- Do reverse: To reverse something means to change its order or direction to be opposite of what it was. In programming, reversing typically refers to reversing the order of elements in a sequence, such as reversing the characters in a string, reversing the order of elements in an array, or reversing the order of items in a list.

```
doreverse:
li $v0, 4              # syscall code for print string
la $a0, sentince       # load address of the prompt string
syscall
li $v0, 8              # Load syscall number for reading a string
la $a0, string1        # Load address of the buffer to store the string
li $a1, 64             # Maximum number of characters to read (adjust as needed)
syscall
jal reverse
li $v0,4
la $a0,string2
syscall
li $v0,4
la $a0, newline
syscall
b loop_display
```

```
9
Please enter the word
computer
retupmoc
```

7-check palindrome: To check if a string is a palindrome means to determine whether it reads the same forwards and backwards. In programming, you would typically compare the string to its reverse to see if they are identical, thus confirming if it's a palindrome.

```
    la $a0, sentincecheck      # load address of the prompt string
    syscall
    li $v0, 8              # Load syscall number for reading a string
    la $a0, string1         # Load address of the buffer to store the string
    li $a1, 64             # Maximum number of characters to read (adjust as needed)
    syscall
    jal check_palindrome
    li $v0,4
    la $a0, newline
    syscall
    b loop_display

    exit:
    li $v0, 4             # syscall code for print string
    la $a0, sentince2      # load address of the prompt string
    syscall
    li $v0, 10
    syscall

check_palindrome:
    move $s3 ,$ra
    jal tolower
    jal reverse
    move $ra ,$s3
    move $t6,$zero
    loop73:
        lb $s5,string1($t6)
        lb $s6,string2($t6)
        beq $s5,'\n',true
        bne  $s6,$s5,false
        add $t6,$t6,$t2

        j loop73
```

```
10
Please enter the word you want to check
mohamed
The word is not a palindrome
```

Chapter Two
“Linked List”

Introduction

What is the linked list: A linked list is a linear data structure consisting of a sequence of elements called nodes. Each node contains a data element and a reference (or pointer) to the next node in the sequence. Unlike arrays, linked lists do not have a fixed size in memory, and their elements can be dynamically allocated. There are various types of linked lists, such as singly linked lists (each node points to the next node), doubly linked lists (each node points to both the next and previous nodes), and circular linked lists (the last node points back to the first node). Linked lists are commonly used in programming for their flexibility in dynamic memory allocation and insertion/deletion operations.

THE C CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Global variables

char options[] = "Please type in one of the numbers below and press enter:\n1 - exit program\n2 - next node\n3 - previous node\n4 - insert after current node\n5 - delete current node\n6 - reset\n7 - debug\n";

char insertMessage[] = "Please type a string up to 10 characters and press enter\n";

char character[10];

char empty[] = "There is no node yet\n";

char doneAdding[] = "\nAdding is done\n";

char currentIs[] = "The current node: ";

char emptyLine[] = "\n";

char array[] = "All elements in the string: \n";

char sep[] = "\t";


struct Node {

    char data[10];

    struct Node* next;
```

```
    struct Node* prev;
};

struct Node* head = NULL;
struct Node* current = NULL;

// Function prototypes
void consolePrint(char* str);
void addNode();
void deleteNode();
void moveToNext();
void moveToPrevious();
void resetList();
void printList();

int main() {
    int choice;
    while (1) {
        consolePrint(options);
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                exit(0);
            case 2:
                moveToNext();
                break;
```

```
            case 3:
                moveToPrevious();
                break;
            case 4:
                addNode();
                break;
            case 5:
                deleteNode();
                break;
            case 6:
                resetList();
                break;
            case 7:
                printList();
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}

void consolePrint(char* str) {
    printf("%s", str);
}

void addNode() {
```

```
    consolePrint(insertMessage);
    scanf("%s", character);

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->data, character);

    if (head == NULL) {
        head = newNode;
        current = newNode;
    } else {
        struct Node* nextNode = current->next;
        current->next = newNode;
        newNode->prev = current;
        newNode->next = nextNode;
        if (nextNode != NULL) {
            nextNode->prev = newNode;
        }
        current = newNode;
    }

    consolePrint(doneAdding);
}

void deleteNode() {
    if (head == NULL) {
        consolePrint(empty);
        return;
```

```
    }

    if (current == head) {
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
            current = head;
        } else {
            current = NULL;
        }
        consolePrint(doneAdding);
        return;
    }

    struct Node* prevNode = current->prev;
    prevNode->next = current->next;
    if (current->next != NULL) {
        current->next->prev = prevNode;
    }
    free(current);
    current = prevNode;
    consolePrint(doneAdding);
}

void moveToNext() {
    if (head == NULL) {
        return;
```

```
    }
    if (current == NULL) {
        current = head;
    } else {
        current = current->next;
    }
}
void moveToPrevious() {
    if (head == NULL || current == NULL || current == head) {
        return;
    }
    current = current->prev;
}
void resetList() {
    head = NULL;
    current = NULL;
}
void printList() {
    consolePrint(array);
    struct Node* temp = head;
    while (temp != NULL) {
        consolePrint(temp->data);
        consolePrint(sep);
        temp = temp->next;
    }
    consolePrint(emptyLine);
}
```

ASSEMBLY CODE

```
.data
options:        .asciiz "Please type in one of the numbers below and press enter: \n 1 - exit program \n 2 - next node \n 3 - previous node \n 4 - insert after

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"

.text
main:
# Main function entry point

start:
# Entry point of the program

    beqz    $s7, noEle
    # Check if linked list is empty
    # If head pointer is zero, branch to noEle label

    la      $a0, currentIs
```

```
    la      $a0, currentIs
    # Load address of message indicating current node
    jal     consolePrint
    # Jump and link to consolePrint function to print message
    move    $a0, $a3
    # Move content of current node address to argument register $a0
    jal     consolePrint
    # Jump and link to consolePrint function to print current node address
    la      $a0, emptyLine
    # Load address of empty line message
    jal     consolePrint
    # Jump and link to consolePrint function to print empty line

optionMenu:
# Menu to display options and get user input

    la      $a0, options
    # Load address of options message
    jal     consolePrint
    # Jump and link to consolePrint function to print options message

    li      $v0, 5
    # Load system call code for reading integer input
    syscall
    # Execute system call
    move    $t0, $v0
    # Move user input to temporary register $t0

    beq     $t0, 1, exit
    # Branch to exit label if input is 1
    beq     $t0, 2, next
    # Branch to next label if input is 2
    beq     $t0, 3, previous
    # Branch to previous label if input is 3
    beq     $t0, 4, insert
    # Branch to insert label if input is 4
    beq     $t0, 5, del
    # Branch to del label if input is 5
    beq     $t0, 6, reset
    # Branch to reset label if input is 6
```

```
    beq     $t0, 6, reset
    # Branch to reset label if input is 6
    beq     $t0, 7, debug
    # Branch to debug label if input is 7

exit:
# Exit the program

    li      $v0, 17
    # Load system call code for exit
    syscall
    # Execute system call

insert:
# Insert a new node

    j       addnode
    # Jump to addnode label

del:
# Delete a node

    jal     delnode
    # Jump and link to delnode function
    j       start
    # Unconditionally jump to start label

next:
# Move to the next node

    beqz    $s7, start
    # Branch to start label if list is empty
    lw      $t5, 12($a3)
    # Load address of next node
    bnez    $t5, nextNode
    # Branch to nextNode label if there is a next node
    j       start
    # Unconditionally jump to start label

previous:
# Move to the previous node
```

```
# Move to the previous node

    beqz    $s7, start
    # Branch to start label if list is empty
    beq     $s7, $a3, start
    # Branch to start label if already at head node
    jal     goBack
    # Jump and link to goBack function
    j       start
    # Unconditionally jump to start label

reset:
# Reset to the first node

    move    $a3, $s7
    # Move head pointer to current node
    j       start
    # Unconditionally jump to start label

debug:
# Print the entire linked list

    jal     printEverything
    # Jump and link to printEverything function
    j       start
    # Unconditionally jump to start label

noEle:
# Handle case when there are no nodes

    la      $a0, empty
    # Load address of empty message
    jal     consolePrint
    # Jump and link to consolePrint function to print empty message
    j       optionMenu
    # Unconditionally jump to optionMenu label

addnode:
# Add a new node
```

```
# Add a new node

    la      $a0, insertMessage
    # Load address of insert message
    jal     consolePrint
    # Jump and link to consolePrint function to print insert message

    jal     alloSpace
    # Jump and link to alloSpace function to allocate memory space
    move    $t1, $v0
    # Move return value of alloSpace (memory address) to $t1

    sw      $zero, ($t1)
    # Store 0 at address pointed to by $t1 (initialize previous pointe
    sw      $zero, 16($t1)
    # Store 0 at address $t1 + 16 (initialize next pointer)

    li      $v0, 8
    # Load system call code for reading string input
    la      $a0, 4($t1)
    # Load address where string input will be stored
    li      $a1, 10
    # Load maximum number of characters to read
    syscall
    # Execute system call to read string input

    beqz    $s7, declareFirstNode
    # Branch to declareFirstNode label if list is empty

    lw      $t2, 16($a3)
    # Load address of next node
    beqz    $t2, noNextNode
    # Branch to noNextNode label if there is a next node

    move    $t0, $t2
    # Move address of next node to temporary register $t0
    la      $t2, 16($t1)
    # Load address where next node will point to
    la      $t0, -4($t0)
    # Load address of previous pointer of next node
    sw      $t2, ($t0)
```

```
    # Load address of previous node
    beqz    $t2, delHead
    # Branch to delHead label if no previous node

    lw      $t3, 12($a3)
    # Load address of next node
    beqz    $t3, delTail
    # Branch to delTail label if no next node

    lw      $t3, 12($a3)
    # Load address of next node
    sw      $t2, -4($t3)
    # Store address of previous node in previous pointer of next node

    lw      $t2, 12($a3)
    # Load address of next node
    lw      $t3, -4($a3)
    # Load address of previous node
    sw      $t2, 12($t3)
    # Store address of next node in next pointer of previous node

    la      $a3, ($t2)
    # Move current node pointer to next node
    doneDel:
        jr      $ra


delHead:
# Label for deleting head node

    lw      $t2, 12($a3)
    # Load address of next node
    sw      $zero, -4($t2)
    # Store 0 in previous pointer of next node
    la      $s7, ($t2)
    # Move head pointer to next node
    la      $a3, ($t2)
    # Move current node pointer to next node
    j       doneDel
    # Unconditionally jump to doneDel label
```

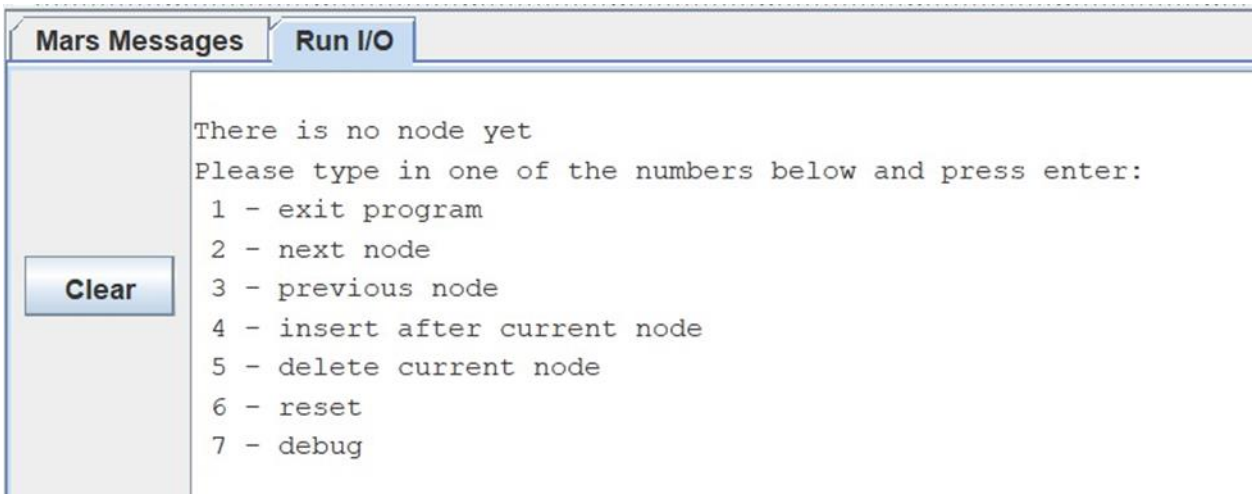
```
    # Load address of previous pointer of next node
    sw      $t2, ($t0)
    # Store address of next node in previous pointer of next node

noNextNode:
# Label for handling when there is no next node

    lw      $t2, 12($a3)
    # Load address of next node
    sw      $t2, 16($t1)
    # Store address of next node in next pointer of new node

    la      $t0, 4($t1)
    # Load address of current node's string
    sw      $t0, 12($a3)
    # Store address of current node's string in next pointer of current node

    la      $t2, ($a3)
    # Load address of current node
    sw      $t2, ($t1)
    # Store address of current node in previous pointer of new node

    la      $a3, 4($t1)
    # Move current node pointer to new node

    la      $a0, doneAdding
    # Load address of doneAdding message
    jal     consolePrint
    # Jump and link to consolePrint function to print doneAdding message
    j       start
    # Unconditionally jump to start label

delnode:
# Delete a node

    beqz    $s7, start
    # Branch to start label if list is empty

    lw      $t2, -4($a3)
    # Load address of previous node
    beqz    $t2, delHead
```

```
delTail:
# Label for deleting tail node

    lw      $t2, -4($a3)
    # Load address of previous node
    sw      $zero, 12($t2)
    # Store 0 in next pointer of previous node
    la      $a3, ($t2)
    # Move current node pointer to previous node
    j       doneDel
    # Unconditionally jump to doneDel label

nextNode:
# Move to the next node

    la      $t5, 12($a3)
    # Load address of next pointer
    lw      $a3, ($t5)
    # Move current node pointer to next node
    j       start
    # Unconditionally jump to start label

goBack:
# Move to the previous node

    la      $t5, -4($a3)
    # Load address of previous pointer
    lw      $a3, ($t5)
    # Move current node pointer to previous node
    jr      $ra
    # Jump to return address

printEverything:
# Print all elements in the list

    la      $a0, array
    # Load address of array message
    jal     consolePrint
    # Jump and link to consolePrint function to print array message
```

```
    jal     consolePrint
    # Jump and link to consolePrint function to print array message
    la      $t1, ($s7)
    # Load address of head node
    beqz    $t1, start
    # Branch to start label if head node is null

printEle:
# Print element in list

    move    $a0, $t1
    # Move address of current node's string to argument register $a0
    jal     consolePrint
    # Jump and link to consolePrint function to print current node's string

    la      $a0, sep
    # Load address of separator message
    jal     consolePrint
    # Jump and link to consolePrint function to print separator message

    lw      $t2, 12($t1)
    # Load address of next node
    beqz    $t2, start
    # Branch to start label if next node is null

    la      $t1, ($t2)
    # Move address of next node to $t1
    j       printEle
    # Unconditionally jump to printEle label

alloSpace:
# Allocate memory space for a new node

    li      $v0, 9
    # Load system call code for memory allocation
    li      $a0, 20
    # Load size of memory to allocate
    syscall
    # Execute system call
    jr      $ra
    # Jump to return address
```

```
    jr      $ra
    # Jump to return address

declareFirstNode:
# Declare the first node when list is empty

    la      $s7, 4($t1)
    # Move head pointer to first node
    la      $a3, 4($t1)
    # Move current node pointer to first node
    la      $a0, doneAdding
    # Load address of doneAdding message
    jal     consolePrint
    # Jump and link to consolePrint function to print doneAdding message
    j       start
    # Unconditionally jump to start label

consolePrint:
# Print a message to the console

    li      $v0, 4
    # Load system call code for printing string
    syscall
    # Execute system call
    jr      $ra
    # Jump to return address
```

RUN CODE



EXPLANATION OF FUNCTION

To insert a node into a linked list, you typically follow these steps:

1. Allocate memory for the new node.

2. Set the data of the new node.

3. Adjust the pointers of the surrounding nodes to include the new node in the list.

```
insert:
# Insert a new node

    j       addnode
    # Jump to addnode label
```

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

Adding is done
The current node: car
```

2-Move to next: To move to the next node in a linked list, you simply follow the pointer from the current node to the next node

```
next:
# Move to the next node

    beqz    $s7, start
    # Branch to start label if list is empty
    lw      $t5, 12($a3)
    # Load address of next node
    bnez    $t5, nextNode
    # Branch to nextNode label if there is a next node
    j       start
    # Unconditionally jump to start label
```

```
2
The current node: name
```

3-previous node: In a singly linked list, you typically don't have direct access to the previous node from a given node because each node only contains a reference to the next node. However, if you need to find the previous node, you usually start from the head of the list and traverse through the list until you find the node whose next node is the node you're interested in.

```
previous:
# Move to the previous node

    beqz    $s7, start
    # Branch to start label if list is empty
    beq     $s7, $a3, start
    # Branch to start label if already at head node
    jal     goBack
    # Jump and link to goBack function
    j       start
    # Unconditionally jump to start label
```

```
3
The current node: car
```

4_ insert after first node: is used to add a new node into the list at a specified position. There are several scenarios when inserting a node.

```
insert:
# Insert a new node

    j       addnode
    # Jump to addnode label
```

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

Adding is done
The current node: name
```

5-Delete node: To delete a node from a linked list, you typically follow these steps:

1. Find the previous node of the node to be deleted.

2. Adjust the pointers to skip over the node to be deleted.

```
del:
# Delete a node

    jal     delnode
    # Jump and link to delnode function
    j       start
    # Unconditionally jump to start label
```

```
5
There is no node yet
```

6-Reset: reset to the first node of a linked list, you simply need to move the pointer back to the head node. Here's how you can do it:

```
reset:
# Reset to the first node

    move    $a3, $s7
    # Move head pointer to current node
    j       start
    # Unconditionally jump to start label
```

```
6
The current node: car
```

7- debug: Debugging is the process of finding and fixing errors in code. The print Everything function might be used to print the contents of the linked list, which could help the programmer identify the source of an error.

```
debug:
# Print the entire linked list

    jal     printEverything
    # Jump and link to printEverything function
    j       start
    # Unconditionally jump to start label
```

```
7
All elements in the string:
car
        cat
        apple
        The current node: apple
```

Task Management

Name	Coding	Report	Presentation
Nayra Fouad Ahmed	14.2% + helped with code syntax	14.2% + editing the report	14.2%
Ahmed Elsayed Ahmed	14.2%	14.2% + editing the report	14.2%
Mohamed Ashraf Eldesoky	14.2% **+PowerPoint design**	14.2%	14.2%
Youssef Waheed	14.2% + helped with code syntax	14.2%	14.2%
Yehia Mohamed Sadek	14.2% + helped with code syntax	14.2%	14.2%
Adham Mohamed Ayad	14.2% + helped with code syntax	14.2%	14.2% **+ PowerPoint design**
Omar Mostafa Salem	14.2%	14.2%	14.2%