



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(sentence2);
                return 0;
            case 2:
                if (is_full()) {
                    print_string(sentence5);
                } else {
                    print_string(sentence6);
                }
                break;
            case 3:
                if (is_empty()) {
                    print_string(sentence4);
                } else {
                    print_string(sentence3);
                }
                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 o;}

```

ASSEMBLY CODE

C:\Users\Ahmed Elayed\Downloads\co project - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

co project

```

1 #array
2 #push pop isempty isfull peek
3 #array_max_size top
4 # 00 --> top of array
5 #push .pop and peek use 00
6 #t1 hold temp address in pop to call the peek
7 #t2 hold temp address in reverse to call pop and peek
8 #isempty and isfull use t0
9 .data
10 array: .byte 100 # (up to 10 items) * (1 bytes)
11 string1: .space 64
12 string2: .space 64
13 correct: .ascii "The word is a palindrome"
14 error: .ascii "The word is not a palindrome"
15 newline: .ascii "\n"
16 options: .ascii "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 - reverse \n8 - reset \n9 - check \n"
17 sentence1: .ascii "Please enter the word \n"
18 sentence2: .ascii "THANK YOU \n"
19 sentence3: .ascii "Not empty \n"
20 sentence4: .ascii "Stack is empty \n"
21 sentence5: .ascii "Stack is full \n"
22 sentence6: .ascii "Stack still has free space \n"
23 sentence7: .ascii "Please insert a character \n"
24 peekvalue: .ascii "The peek value in the stack is \n"
25 resetmsg: .ascii "Stack reset \n"
26 sentencecheck: .ascii "Please enter the word you want to check \n"
27 .text

```

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
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000

co project

```

28 main:
29 addi $v0, $zero, 0
30 addi $t2, $zero, 1
31
32 loop_display:
33 li $v0, 4 # syscall code for print string
34 la $a0, options # load address of the prompt string
35 syscall
36 li $v0, 5 # syscall code for read integer
37 syscall
38 move $t0, $v0
39 beq $t0, 1, exit
40 beq $t0, 2, doisfull
41 beq $t0, 3, doisempty
42 beq $t0, 4, dopeek
43 beq $t0, 5, dopush
44 beq $t0, 6, dopop
45 beq $t0, 7, dotolower
46 beq $t0, 8, dotoupper
47 beq $t0, 9, doreverse
48 beq $t0, 10, docheck_palindrome
49 beq $t0, 11, doreset
50 j loop_display
51

```

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
\$s8	24	0x00000000
\$s9	25	0x00000000

co project

```

52 dotolower:
53 li $v0, 4 # syscall code for print string
54 la $a0, sentence1 # load address of the prompt string
55 syscall
56 li $v0, 8 # Load syscall number for reading a string
57 la $a0, string1 # Load address of the buffer to store the string
58 li $a1, 64 # Maximum number of characters to read (adjust as needed)
59 syscall
60 jal tolower
61 jal pprint
62 j loop_display
63
64 dotoupper:
65 li $v0, 4 # syscall code for print string
66 la $a0, sentence2 # load address of the prompt string
67 syscall
68 li $v0, 8 # Load syscall number for reading a string
69 la $a0, string1 # Load address of the buffer to store the string
70 li $a1, 64 # Maximum number of characters to read (adjust as needed)
71 syscall
72 jal toupper
73 jal pprint
74 j loop_display
75
76 doisfull:
77 jal isfull
78 beq $t0, $zero, fady
79 li $v0, 4 # syscall code for print string
80 la $a0, sentence3 # Load address of the prompt string
81 syscall
82 beq $t0, $t2, loop_display

```

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
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$fp	29	0x7fffffc0
\$fpc	30	0x00000000
\$ra	31	0x00000000
\$pc	32	0x00000000

Edit		Execute	Registers			Coproc 1	Coproc 0
co project			Name	Number	Value		
82	beq \$t0, \$t2, loop_display		\$zero	0	0x00000000		
83	fade:		\$at	1	0x00000000		
84	li \$v0, 4	# syscall code for print string	\$v0	2	0x00000000		
85	la \$a0, sentince4	# load address of the prompt string	\$v1	3	0x00000000		
86	syscall		\$a0	4	0x00000000		
87	h loop_display		\$a1	5	0x00000000		
88			\$a2	6	0x00000000		
89			\$a3	7	0x00000000		
90	doempty:		\$t0	8	0x00000000		
91	beq \$t0, \$zero, mswady		\$t1	9	0x00000000		
92	li \$v0, 4	# syscall code for print string	\$t2	10	0x00000000		
93	la \$a0, sentince4	# load address of the prompt string	\$t3	11	0x00000000		
94	syscall		\$t4	12	0x00000000		
95	beq \$t0, \$t2, loop_display		\$t5	13	0x00000000		
96	mswady:		\$t6	14	0x00000000		
97	li \$v0, 4	# syscall code for print string	\$t7	15	0x00000000		
98	la \$a0, sentince3	# load address of the prompt string	\$t8	16	0x00000000		
99	syscall		\$t9	17	0x00000000		
100	h loop_display		\$a2	18	0x00000000		
101			\$a3	19	0x00000000		
102	dopush:		\$a4	20	0x00000000		
103	li \$v0, 4	# syscall code for print string	\$a5	21	0x00000000		
104	la \$a0, sentince7	# load address of the prompt string	\$a6	22	0x00000000		
105	syscall		\$a7	23	0x00000000		
106	li \$v0, 12		\$a8	24	0x00000000		
107	syscall		\$a9	25	0x00000000		
108	jal push		\$a10	26	0x00000000		
109	li \$v0, 4		\$a11	27	0x00000000		
110	la \$a0, newline		\$a12	28	0x00000000		
111	syscall		\$a13	29	0x00000000		
112	h loop_display		\$a14	30	0x00000000		
113			\$a15	31	0x00000000		
114	dopop:		\$a16	32	0x00000000		
115	li \$v0, 4		\$a17	33	0x00000000		
116	la \$a0, newline		\$a18	34	0x00000000		
117	syscall		\$a19	35	0x00000000		
118	jal pop		\$a20	36	0x00000000		
119	move \$a0, \$v0		\$a21	37	0x00000000		
120	li \$v0, 11		\$a22	38	0x00000000		
121	syscall		\$a23	39	0x00000000		
122	li \$v0, 4		\$a24	40	0x00000000		
123	la \$a0, newline		\$a25	41	0x00000000		
124	syscall		\$a26	42	0x00000000		
125	h loop_display		\$a27	43	0x00000000		
126			\$a28	44	0x00000000		
127	dopeek:		\$a29	45	0x00000000		
128	li \$v0, 4		\$a30	46	0x00000000		
129	la \$a0, peekvalue		\$a31	47	0x00000000		
130	syscall		\$a32	48	0x00000000		
131	jal peek		\$a33	49	0x00000000		
132	move \$a0, \$v0		\$a34	50	0x00000000		
133	li \$v0, 11		\$a35	51	0x00000000		
134	syscall		\$a36	52	0x00000000		
135	li \$v0, 4		\$a37	53	0x00000000		
136	la \$a0, newline		\$a38	54	0x00000000		
137	syscall		\$a39	55	0x00000000		
138	h loop_display		\$a40	56	0x00000000		
139			\$a41	57	0x00000000		
140	li \$v0, 4	# syscall code for print string	\$a42	58	0x00000000		
141	la \$a0, sentince	# load address of the prompt string	\$a43	59	0x00000000		
142	syscall		\$a44	60	0x00000000		
143	li \$v0, 8	# load syscall number for reading a string	\$a45	61	0x00000000		
144	la \$a0, string1	# load address of the buffer to store the string	\$a46	62	0x00000000		
145	li \$a1, 64	# Maximum number of characters to read (adjust as needed)	\$a47	63	0x00000000		
146	syscall		\$a48	64	0x00000000		
147	jal reverse		\$a49	65	0x00000000		
148	li \$v0, 4		\$a50	66	0x00000000		
149	la \$a0, string2		\$a51	67	0x00000000		
150	syscall		\$a52	68	0x00000000		
151	li \$v0, 4		\$a53	69	0x00000000		
152	la \$a0, newline		\$a54	70	0x00000000		
153	syscall		\$a55	71	0x00000000		
154	h loop_display		\$a56	72	0x00000000		
155			\$a57	73	0x00000000		
156	docheck_palindrome:		\$a58	74	0x00000000		
157	li \$v0, 4	# syscall code for print string	\$a59	75	0x00000000		
158	la \$a0, sentincecheck	# load address of the prompt string	\$a60	76	0x00000000		
159	syscall		\$a61	77	0x00000000		
160	li \$v0, 8	# load syscall number for reading a string	\$a62	78	0x00000000		
161	la \$a0, string1	# load address of the buffer to store the string	\$a63	79	0x00000000		
162	li \$a1, 64	# Maximum number of characters to read (adjust as needed)	\$a64	80	0x00000000		
163	syscall		\$a65	81	0x00000000		
164	jal check_palindrome		\$a66	82	0x00000000		
165	li \$v0, 4		\$a67	83	0x00000000		
166	la \$a0, newline		\$a68	84	0x00000000		
167	syscall		\$a69	85	0x00000000		
168	h loop_display		\$a70	86	0x00000000		
169			\$a71	87	0x00000000		
170	exit:		\$a72	88	0x00000000		
171	li \$v0, 4	# syscall code for print string	\$a73	89	0x00000000		
172	la \$a0, sentince2	# load address of the prompt string	\$a74	90	0x00000000		
173	syscall		\$a75	91	0x00000000		
174	h loop_display		\$a76	92	0x00000000		
175			\$a77	93	0x00000000		
176			\$a78	94	0x00000000		
177			\$a79	95	0x00000000		
178			\$a80	96	0x00000000		
179	li \$v0, 4	# syscall code for print string	\$a81	97	0x00000000		
180	la \$a0, sentince2	# load address of the prompt string	\$a82	98	0x00000000		
181	syscall		\$a83	99	0x00000000		
Line: 307 Column: 11 <input checked="" type="checkbox"/> Show Line Numbers							
181	syscall		\$zero	0	0x00000000		
182	li \$v0, 10		\$at	1	0x00000000		
183	syscall		\$v0	2	0x00000000		
184			\$v1	3	0x00000000		
185	check_palindrome:		\$a0	4	0x00000000		
186	move \$a2, \$a1		\$a1	5	0x00000000		
187	jal tolower		\$a2	6	0x00000000		
188	jal reverse		\$t0	8	0x00000000		
189	move \$a1, \$a2		\$t1	9	0x00000000		
190	move \$t2, \$zero		\$t2	10	0x00000000		
191	loop73:		\$t3	11	0x00000000		
192	lb \$a3, string1(\$t2)		\$t4	12	0x00000000		
193	lb \$a4, string2(\$t2)		\$t5	13	0x00000000		
194	beq \$a3, '\n', true		\$t6	14	0x00000000		
195	bnz \$a4, \$a3, false		\$t7	15	0x00000000		
196	add \$t6, \$t6, \$t2		\$a0	16	0x00000000		
197			\$a1	17	0x00000000		
198	jal loop73		\$a2	18	0x00000000		
199			\$a3	19	0x00000000		
200			\$a4	20	0x00000000		
201	true:		\$a5	21	0x00000000		
202	li \$v0, 4		\$a6	22	0x00000000		
203	la \$a0, correct		\$a7	23	0x00000000		
204	syscall		\$a8	24	0x00000000		
205	j jump		\$t9	25	0x00000000		
206			\$a10	26	0x00000000		
207	false:		\$a11	27	0x00000000		
208	li \$v0, 4		\$a12	28	0x00000000		
209	la \$a0, error		\$a13	29	0x00000000		
210	syscall		\$a14	30	0x00000000		
211	j jump		\$a15	31	0x00000000		
212			\$a16	32	0x00000000		
213	tolower:		\$a17	33	0x00000000		
214			\$a18	34	0x00000000		

co project

244

245 push:

246 add \$a0,\$a0,\$t2

247 sb \$v0,array(\$a0)

248 jr \$ra

249

250 pop:

251 add \$t1,\$ra,\$zero

252 jal peek

253 add \$t1,\$t1,\$zero

254 addi \$t0,\$a0,-1

255 jr \$ra

256

257

258

259 peek:

260 lb \$v0,array(\$a0)

261 jr \$ra

262

263

264 isempty:

265 add \$t0,\$zero,\$zero

266 bne \$a0,\$zero,jump

267 addi \$t0,\$zero,1

268 j jump

269

270 isfull:

271 add \$t0,\$zero,\$zero

272 bne \$a0,100,jump

273 addi \$t0,\$zero,1

274 j jump

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

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
\$a0	16	0x00000000
\$a1	17	0x00000000
\$a2	18	0x00000000
\$a3	19	0x00000000
\$a4	20	0x00000000
\$a5	21	0x00000000
\$a6	22	0x00000000
\$a7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$fp	29	0xffffffff
\$ra	30	0x00000000
\$ra	31	0x00000000
\$pc		0x00400000
\$hi		0x00000000
\$lo		0x00000000

co project

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

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
\$a0	16	0x00000000
\$a1	17	0x00000000
\$a2	18	0x00000000
\$a3	19	0x00000000
\$a4	20	0x00000000
\$a5	21	0x00000000
\$a6	22	0x00000000
\$a7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$fp	29	0xffffffff
\$ra	30	0x00000000
\$ra	31	0x00000000
\$pc		0x00400000
\$hi		0x00000000
\$lo		0x00000000

co project

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

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
\$a0	16	0x00000000
\$a1	17	0x00000000
\$a2	18	0x00000000
\$a3	19	0x00000000
\$a4	20	0x00000000
\$a5	21	0x00000000
\$a6	22	0x00000000
\$a7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$fp	29	0xffffffff
\$ra	30	0x00000000
\$ra	31	0x00000000
\$pc		0x00400000
\$hi		0x00000000
\$lo		0x00000000

RUN CODE

Mars Messages	Run I/O
<div>Clear</div>	reset: reset completed.
	Please type in one of the number below and press enter:
	1 - Exit program
	2 - IsFull
	3 - IsEmpty
	4 - Peek
	5 - push
	6 - Pop
	7 - Convert the word to lowercase
	8 - Convert the word to Uppercase
	9 - Reverse the word
	10 - Check the word is palindrome
	11 - Reset

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, sentence5   # 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, sentence7   # 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.

```

213 toLower:
214
215     add $t5,$zero,$zero
216     loopx:
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 loopx
223

```

\$t0	4	0x00000000
\$t1	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 o;

}

```

```

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    $a7, noEle
    # check if linked list is empty
    # If head pointer is zero, branch to noEle label

    la     $a0, currentIs

# ... (omitted code) ...

    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 pointer)
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

Mars Messages	Run I/O
<div>Clear</div>	<p>There is no node yet</p> <p>Please type in one of the numbers below and press enter:</p> <ul style="list-style-type: none"> 1 - exit program 2 - next node 3 - previous node 4 - insert after current node 5 - delete current node 6 - reset 7 - debug

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%