190905514 **MOHAMMAD TOFIK**

WEEK 8 LAB 8 :

1.Implement a queue using singly linked list without header node.

```
#include<stdio.h>
#include <stdlib.h>
typedef struct node *NODEPTR;
struct node
{
int info;
NODEPTR link;
};
NODEPTR getNode()
{
NODEPTR temp;
temp = (NODEPTR)malloc(sizeof(struct node));
if (temp == NULL)
{
printf("NO MEMEORY");
exit(0);
}
return temp;
}
void enqueue(NODEPTR *front, NODEPTR *rear, int data)
{
NODEPTR temp = getNode();
temp->link = NULL;
temp->info = data;
if (*front == NULL)
{
*front = *rear = temp;
}
else
{
(*rear)->link = temp;
```

```c
(*rear) = temp;
}
}
int dequeue(NODEPTR *front, NODEPTR *rear)
{
NODEPTR temp;
if (*front == NULL)
{
printf("EMPTY\n");
return -1;
}
temp = (*front);
if (*front == *rear)
{
*front = *rear = NULL;
}
else
{
*front = temp->link;
}
int x = temp->info;
free(temp);
return x;
}
void display(NODEPTR *front, NODEPTR *rear)
{
NODEPTR temp;
if (*front == NULL)
{
printf("EMPTY\n");
return;
}
temp = (*front);
printf("QUEUE : ");
for (; temp != *rear; temp = temp->link)
{
printf("%d ", temp->info);
```

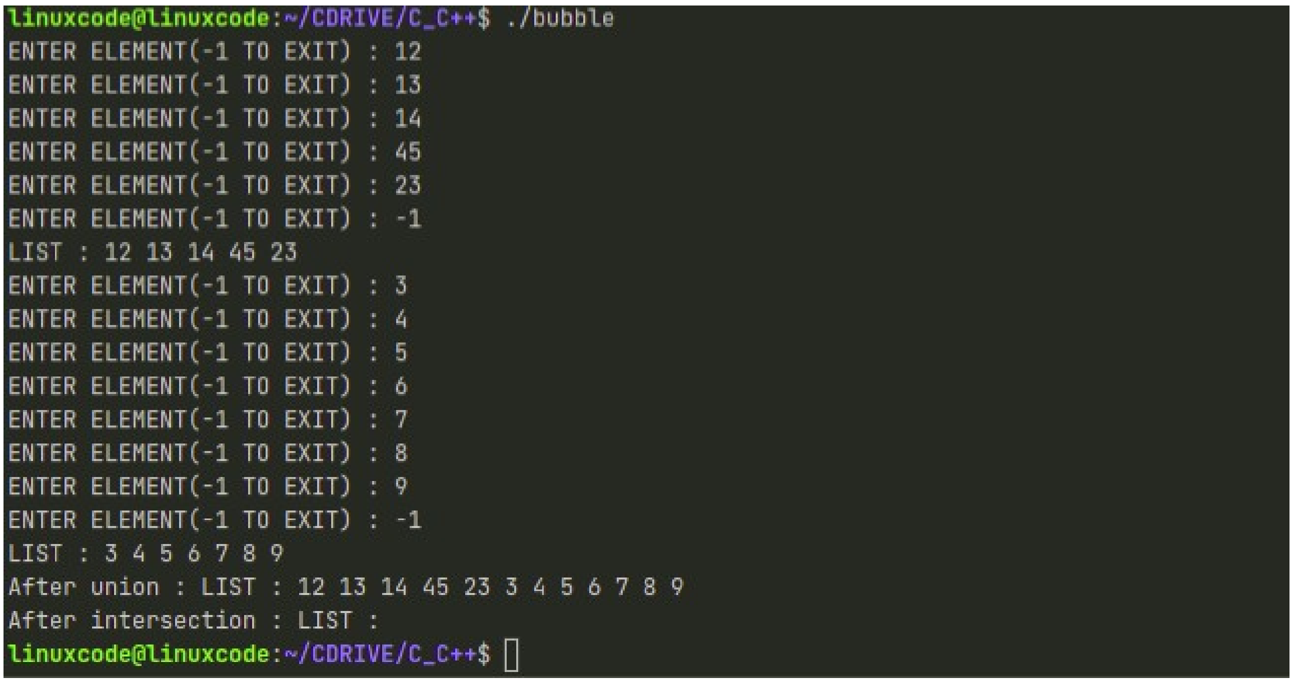
```c
}
printf("%d ", temp->info);
printf("\n");
}
int main()
{
NODEPTR front = NULL;
NODEPTR rear = NULL;
int choice, x;
while (1)
{
printf("\n1. INSEET AN ELEMENT");
printf("\n2. DELETE AN ELEMENT");
printf("\n3. DISPLAY ALL QUEUE ELEMENTS");
printf("\n4. QUIT");
printf("\n\nENTER CHOICE -");
scanf("%d", &choice);
switch (choice)
{
case 1:
printf("ENTER ELEMENT -");
scanf("%d", &x);
enqueue(&front, &rear, x);
break;
case 2:
x = dequeue(&front, &rear);
if (x != -1)
printf("ELEMENT DELETED - %d\n", x);
break;
case 3:
display(&front, &rear);
break;
case 4:
printf("EXITITNG\n");
exit(0);
default:
printf("EXITING");
return 0;
}
}
return 0;
}
```

2.Implement a queue using singly linked list without header node.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node *NODEPTR;
struct node
{
int info;
NODEPTR link;
};
NODEPTR getNode()
{
NODEPTR temp;
temp = (NODEPTR)malloc(sizeof(struct node));
if (temp == NULL)
{
printf("NO MEMEORY");
exit(0);
}
return temp;
}
NODEPTR insertlast(NODEPTR last, int data)
{
NODEPTR temp = getNode();
temp->link = NULL;
temp->info = data;
last->link = temp;
last = temp;
return last;
}
NODEPTR createlist()
{
```

```c
NODEPTR first = getNode();
NODEPTR last = first;
int x;
printf("ENTER ELEMENT(-1 TO EXIT) : ");
scanf("%d", &x);
while (x ≠ -1)
{
last = insertlast(last, x);
printf("ENTER ELEMENT(-1 TO EXIT) : ");
scanf("%d", &x);
}
NODEPTR temp = first;
first = temp→link;
free(temp);
return first;
}
void display(NODEPTR front)
{
NODEPTR temp;
temp = (front);
printf("LIST : ");
for (; temp ≠ NULL; temp = temp→link)
{
printf("%d ", temp→info);
}
printf("\n");
}
int ismember(NODEPTR a, int x)
{
NODEPTR temp = a;
while (temp ≠ NULL)
{
if (temp→info == x)
{
return 1;
}
temp = temp→link;
}
return 0;
}
NODEPTR dunion(NODEPTR a, NODEPTR b)
{
NODEPTR c = getNode();
NODEPTR x = c;
NODEPTR temp = a;
while (temp ≠ NULL)
{
x = insertlast(x, temp→info);
temp = temp→link;
}
temp = b;
while (temp ≠ NULL)
{
if (ismember(a, temp→info) == 0)
x = insertlast(x, temp→info);
temp = temp→link;
}
temp = c;
c = temp→link;
free(temp);
return c;
}
NODEPTR inter(NODEPTR a, NODEPTR b)
{
NODEPTR c = getNode();
NODEPTR x = c;
NODEPTR temp = b;
while (temp ≠ NULL)
{
if (ismember(a, temp→info) == 1)
x = insertlast(x, temp→info);
temp = temp→link;
}
temp = c;
```

```c
c = temp→link;
free(temp);
return c;
}
int main()
{
NODEPTR a = createlist();
display(a);
NODEPTR b = createlist();
display(b);
printf("After union : ");
NODEPTR c = dunion(a, b);
display(c);
printf("After intersection : ");
NODEPTR d = inter(a, b);
display(d);
return 0;
}
```



```
linuxcode@linuxcode:~/CDRIVE/C_C++$ ./bubble
ENTER ELEMENT(-1 TO EXIT) : 12
ENTER ELEMENT(-1 TO EXIT) : 13
ENTER ELEMENT(-1 TO EXIT) : 14
ENTER ELEMENT(-1 TO EXIT) : 45
ENTER ELEMENT(-1 TO EXIT) : 23
ENTER ELEMENT(-1 TO EXIT) : -1
LIST : 12 13 14 45 23
ENTER ELEMENT(-1 TO EXIT) : 3
ENTER ELEMENT(-1 TO EXIT) : 4
ENTER ELEMENT(-1 TO EXIT) : 5
ENTER ELEMENT(-1 TO EXIT) : 6
ENTER ELEMENT(-1 TO EXIT) : 7
ENTER ELEMENT(-1 TO EXIT) : 8
ENTER ELEMENT(-1 TO EXIT) : 9
ENTER ELEMENT(-1 TO EXIT) : -1
LIST : 3 4 5 6 7 8 9
After union : LIST : 12 13 14 45 23 3 4 5 6 7 8 9
After intersection : LIST :
linuxcode@linuxcode:~/CDRIVE/C_C++$
```

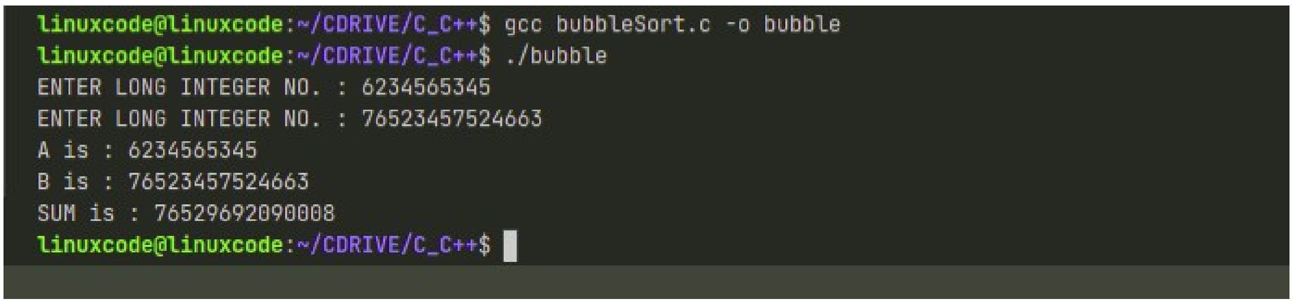
3.Addition of long integer in c using doubly linked list .

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct node *NODEPTR;
struct node
{
int info;
NODEPTR rlink;
NODEPTR llink;
};
NODEPTR getNode()
{
NODEPTR temp;
temp = (NODEPTR)malloc(sizeof(struct node));
if (temp == NULL)
{
printf("NO MEMEORY");
exit(0);
}
return temp;
}
NODEPTR insertfront(NODEPTR head, int data)
{
NODEPTR temp = getNode();
temp->info = data;
NODEPTR x = head->rlink;
temp->rlink = x;
head->rlink = temp;
x->llink = temp;
temp->llink = head;
return head;
}
NODEPTR readlongint()
{
NODEPTR head = getNode();
head->llink = head->rlink = head;
```

```c
char s[100];
int i, n;
printf("ENTER LONG INTEGER NO. : ");
scanf("%s", s);
n = strlen(s);
for (i = n - 1; i >= 0; i--)
{
head = insertfront(head, s[i] - '0');
}
return head;
}
void display(NODEPTR head)
{
NODEPTR p = head->rlink;
if (head->rlink == head)
{
printf("Empty\n");
return;
}
printf(": ");
while (p != head)
{
printf("%d", p->info);
p = p->rlink;
}
printf("\n");
}
NODEPTR addlongint(NODEPTR a, NODEPTR b)
{
int x, y, z = 0;
NODEPTR s = getNode();
s->rlink = s->llink = s;
NODEPTR c = a;
NODEPTR d = b;
NODEPTR r, R;
a = a->llink;
b = b->llink;
```

```
while (c ≠ a && d ≠ b)
{
y = a→info + b→info + z;
x = y % 10;
s = insertfront(s, x);
z = y / 10;
b = b→llink;
a = a→llink;
}
if (a ≠ c)
{
r = a;
R = c;
}
else
{
r = b;
R = d;
}
while (r ≠ R)
{
y = r→info + z;
x = y % 10;
s = insertfront(s, x);
z = y / 10;
r = r→llink;
}
if (z ≠ 0)
s = insertfront(s, z);
return s;
}
int main()
{
NODEPTR a = readlongint();
NODEPTR b = readlongint();
printf("A is ");
display(a);
```

```c
printf("B is ");
display(b);
NODEPTR sum = addlongint(a, b);
printf("SUM is ");
display(sum);
return 0;
}
```



```
linuxcode@linuxcode:~/CDRIVE/C_C++$ gcc bubbleSort.c -o bubble
linuxcode@linuxcode:~/CDRIVE/C_C++$ ./bubble
ENTER LONG INTEGER NO. : 6234565345
ENTER LONG INTEGER NO. : 76523457524663
A is : 6234565345
B is : 76523457524663
SUM is : 76529692090008
linuxcode@linuxcode:~/CDRIVE/C_C++$
```

4.

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
typedef struct node
{
int key;
struct node *left, *right;
} * NODE;
typedef struct
{
NODE S[MAX];
int tos;
} STACK;
NODE newNODE(int item)
{
NODE temp = (NODE)malloc(sizeof(struct node));
temp->key = item;
temp->left = temp->right = NULL;
return temp;
}
void push(STACK *s, NODE n)
```

```c
{
s->S[++(s->tos)] = n;
}
NODE pop(STACK *s)
{
return s->S[(s->tos)--];
}
void inorder(NODE root)
{
NODE curr;
curr = root;
STACK S;
S.tos = -1;
push(&S, root);
curr = curr->left;
while (S.tos != -1 || curr != NULL)
{
while (curr != NULL)
{
push(&S, curr);
curr = curr->left;
}
curr = pop(&S);
printf("%d\t", curr->key);
curr = curr->right;
}
}
NODE insert(NODE node, int key)
{
if (node == NULL)
return newNODE(key);
if (key < node->key)
node->left = insert(node->left, key);
else if (key > node->key)
node->right = insert(node->right, key);
return node;
}
```

```c
NODE minValueNode(NODE node)
{
NODE current = node;
while (current && current->left != NULL)
current = current->left;
return current;
}
NODE deleteNode(NODE root, int key)
{
if (root == NULL)
return root;
if (key < root->key)
root->left = deleteNode(root->left, key);
else if (key > root->key)
root->right = deleteNode(root->right, key);
else
{
if (root->left == NULL)
{
NODE temp = root->right;
free(root);
return temp;
}
else if (root->right == NULL)
{
NODE temp = root->left;
free(root);
return temp;
}
NODE temp = minValueNode(root->right);
root->key = temp->key;
root->right = deleteNode(root->right, temp->key);
}
return root;
}
void main()
{
```

```c
NODE root = NULL;
int k;
printf("Enter the root:\t");
scanf("%d", &k);
root = insert(root, k);
int ch;
do
{
printf("\nEnter your choice:");
printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit:\
n");
scanf("%d", &ch);
switch (ch)
{
case 1:
printf("Enter element to be inserted:\t");
scanf("%d", &k);
root = insert(root, k);
break;
case 2:
printf("Enter element to be deleted:\t");
scanf("%d", &k);
root = deleteNode(root, k);
break;
case 3:
inorder(root);
break;
}
} while (ch < 4);
}
```