**CODE:-**

#include <stdio.h>

#include <stdlib.h>

typedef struct node node;

struct node

{

node \*prev;

int data;

node \*next;

};

int dig(int n)

{

int count = 0;

while (n)

{

count++;

n /= 10;

}

return count;

}

node \*create\_linked\_list(node \*start)

{

node \*temp, \*p;

int n;

printf("Enter the number of nodes.\n");

scanf("%d", &n);

start = NULL;

if (n == 0)

return start;

for (int i = 0; i < n; i++)

{

temp = (node \*)malloc(sizeof(node));

printf("\nEnter the data for node %d: ", i + 1);

scanf("%d", &temp->data);

temp->next = NULL;

if (start == NULL)

{

temp->prev = NULL;

start = temp;

}

else

{

p = start;

while (p->next != NULL)

p = p->next;

temp->prev = p;

p->next = temp;

}

}

return start;

}

void display\_num(node \*start)

{

node \*p;

if (start == NULL)

{

printf("Linked list is empty\n");

return;

}

p = start;

while (p->next != NULL)

p = p->next;

printf("The number is: ");

while(p!=NULL)

{

printf("%d", p->data);

p = p->prev;

}

printf("\n");

}

void display\_linked\_list(node \*start)

{

node \*p;

if (start == NULL)

{

printf("Linked list is empty\n");

return;

}

p = start;

printf("Linked list is: \n");

while (p->next != NULL)

{

printf("%d->", p->data);

p = p->next;

}

printf("%d\n", p->data);

}

node \*swap(node \*start)

{

node \*p, \*q, \*r;

int count = 0;

p = start;

q = start->next;

while (q != NULL)

{

r = p->prev;

p->prev = p->next;

p->next = q->next;

q->next = q->prev;

q->prev = r;

if (count == 0)

start = q;

if (p->next != NULL)

p->next->prev = p;

if (q->prev != NULL)

q->prev->next = q;

p = p->next;

if (p != NULL)

q = p->next;

else

q = NULL;

count++;

}

return start;

}

node \*create\_num\_ll(node \*start, int num, int digmax)

{

node \*temp, \*p;

start = NULL;

if (digmax == 0)

return start;

for (int i = 0; i < digmax; i++)

{

temp = (node \*)malloc(sizeof(node));

temp->data = num % 10;

temp->next = NULL;

if (start == NULL)

{

temp->prev = NULL;

start = temp;

}

else

{

p = start;

while (p->next != NULL)

p = p->next;

temp->prev = p;

p->next = temp;

}

num /= 10;

}

return start;

}

node \*add\_ll(node \*start1, node \*start2, node \*start3)

{

node \*p = start1, \*q = start2, \*temp, \*r;

int carry = 0, sum, p\_dig, q\_dig;

while (p != NULL && q != NULL)

{

temp = (node \*)malloc(sizeof(node));

if (p == NULL)

p\_dig = 0;

else

p\_dig = p->data;

if (q == NULL)

q\_dig = 0;

else

q\_dig = q->data;

sum = carry + p\_dig + q\_dig;

if (sum >= 10)

{

carry = 1;

sum -= 10;

}

else

carry = 0;

temp->data = sum;

temp->next = NULL;

if (start3 == NULL)

{

temp->prev = NULL;

start3 = temp;

}

else

{

r = start3;

while (r->next != NULL)

r = r->next;

temp->prev = r;

r->next = temp;

}

p = p->next;

q = q->next;

}

if (carry == 1)

{

r = start3;

while (r->next != NULL)

r = r->next;

temp = (node \*)malloc(sizeof(node));

temp->data = 1;

temp->next = NULL;

temp->prev = r;

r->next = temp;

}

return start3;

}

int main()

{

int ch, num1, num2, digmax;

node \*start1 = NULL, \*start2 = NULL, \*start3 = NULL;

while (1)

{

printf("Enter 1 to swap elements of doubly linked list.\n");

printf("Enter 2 to create two DLL in which info info part contains digit of a given number.Also addition is performed on them.\n");

printf("Enter 3 to exit.\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

start1 = create\_linked\_list(start1);

printf("Before swapping:\n");

display\_linked\_list(start1);

start1 = swap(start1);

printf("After swapping:\n");

display\_linked\_list(start1);

break;

case 2:

printf("Enter the numbers 1 and 2.\n");

scanf("%d%d", &num1, &num2);

if (dig(num1) > dig(num2))

digmax = dig(num1);

else

digmax = dig(num2);

start1 = create\_num\_ll(start1, num1, digmax);

printf("Number 1 is:\n");

display\_linked\_list(start1);

start2 = create\_num\_ll(start2, num2, digmax);

printf("Number 2 is:\n");

display\_linked\_list(start2);

start3 = add\_ll(start1, start2, start3);

printf("The added number is:\n");

display\_linked\_list(start3);

display\_num(start3);

break;

case 3:

exit(1);

default:

printf("Erroneous input.\n");

}

}

}

**OUTPUT:-**



**CODE:-**

#include <stdio.h>

#include <stdlib.h>

typedef struct node node;

struct node

{

int info;

node \*link;

};

node \*create\_list(node \*last)

{

node \*temp, \*p;

int n;

printf("Enter the number of nodes.\n");

scanf("%d", &n);

if (n == 0)

return last;

printf("Enter the data.\n");

for (int i = 0; i < n; i++)

{

temp = (node \*)malloc(sizeof(node));

scanf("%d", &temp->info);

temp->link = NULL;

if (last == NULL)

{

last = temp;

last->link = last;

}

else

{

temp->link = last->link;

last->link = temp;

last = temp;

}

}

return last;

}

void \*display\_list(node \*last)

{

node \*p;

if (last == NULL)

{

printf("The circular linked list is empty.\n");

return;

}

p = last->link;

printf("The elements of the circular linked list are : ");

do

{

printf("%d ", p->info);

p = p->link;

} while (p != last->link);

printf("\n");

}

node \*del(node \*last, int data)

{

struct node \*tmp, \*p;

if (last->link == last && last->info == data)

{

tmp = last;

last = NULL;

free(tmp);

return last;

}

if (last->link->info == data)

{

tmp = last->link;

last->link = tmp->link;

free(tmp);

return last;

}

p = last->link;

while (p->link != last)

{

if (p->link->info == data)

{

tmp = p->link;

p->link = tmp->link;

free(tmp);

return last;

}

p = p->link;

}

if (last->info == data)

{

tmp = last;

p->link = last->link;

last = p;

free(tmp);

return last;

}

return last;

}

node \*delete\_alt\_nodes(node \*last)

{

node \*p, \*q, \*temp;

if (last == NULL)

return;

p = q = last->link;

int count = 0;

do

{

if (count % 2 == 1)

{

temp = p;

p = p->link;

last = del(last, temp->info);

}

else

p = p->link;

count++;

} while (p != q);

return last;

}

int main()

{

int ch;

node \*last = NULL;

while (1)

{

printf("Enter 1 to create circular LL and delete alternate nodes.\n");

printf("Enter 2 to exit.\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

last = create\_list(last);

printf("Circular list before removal:\n");

display\_list(last);

printf("Circular list after removal.\n");

last = delete\_alt\_nodes(last);

display\_list(last);

break;

case 2:

exit(1);

default:

printf("Erroneous input.\n");

}

}

}

**OUTPUT:-**

