NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY

(An Institution of National Importance under MHRD, Govt. of India)

KARAIKAL – 609 609

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Roll Number: CS19B1009 **Name** ARUN KUMAR R

Semester: 2nd semester **Class:** B TECH, CSE

Subject Code: CS106 **Subject Name:** DATA STRUCTURES LABORATORY

1. SINGLY LINKED LIST

Date:05.02.20

AIM:

To implement storage and manipulation of data using singly linked list.

ALGORITHM:

1. start the program.
2. declare the variables.
3. implement the functions for inserting the first element, last element and nth element, deleting the first element, last element and nth element.
4. using dynamic memory allocation to allocate memory block for node, which contains the value and the address for the next element
5. For inserting, create a memory for node and copy the value to the node, if the node is to be inserted first update nodes next to the head, and if the node is to be inserted at last update the nodes next element to null and point the next of previous element to the created node.
6. get the input from the node .
7. output the result.
8. end the program.

PROGRAM:

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

```
#include <stdlib.h>
#define MAX 100
int count = 0;
struct node
{
	int value;
	struct node *next;
};
typedef struct node NODE;
struct node *HEAD = NULL;
int isEmpty()
{
	if (count == 0)
	{
	printf("The list is empty\n");
	return 1;
	}
	return 0;
}
int isFull()
{
	if (count == MAX)
	{
	printf("the list is full\n");
	return 1;
	}
	return 0;
}
void msg(char message[])
{
	printf("%s", message);
}
int position_chooser(int IS_FIRST_OR_LAST)
{
	int p;
	if (IS_FIRST_OR_LAST)
```

```
{
char choose;
printf("FIRST = f, LAST = l\n");
while (1)
{
scanf(" %c", &choose);

switch (choose)
{
case 'f':
        p = 0;
        break;
case 'l':
        p = count - 1;
        break;
default:
        p = -1;
        break;
}

if (p == -1)
{
        msg("Enter valid input, TRY AGAIN\n");

        continue;
}

break;
}
}
else
{
msg("Enter the position\n");
while (1)
{
scanf("%d", &p);
```

```
        if (p > count)
        {
                printf("Invalid position, there are only %d elements and position
starts from '0', TRY AGAIN\n", count);
                continue;
        }
        else
                break;
        }
        }

        return p;
}
int isNull(NODE *ptr)
{
        if (ptr == NULL)
        return 1;
        return 0;
}
int Remove(int rem_first_or_last)
{
        if (isEmpty())
        {
        return 0;
        }
        int value, p;
        NODE *NEXT, *DELETE;
        NEXT = HEAD;

        p = position_chooser(rem_first_or_last);

        count--;

        if (p == 0)
        {
        value = HEAD->value;
```

```
        DELETE = HEAD;
        HEAD = NEXT->next;
        free(DELETE);
        printf("The element %d was deleted", value);

        return 1;
        }
        for (int i = 1; i < p && NEXT->next != NULL; i++)
        NEXT = NEXT->next;

        DELETE = NEXT->next;
        value = DELETE->value;
        NEXT->next = DELETE->next;
        free(DELETE);

        printf("The element %d was deleted", value);
        return 1;
}
int update()
{
        NODE *NEXT;
        int p, value, fvalue;
        NEXT = HEAD;
        msg("Enter the new value\n");
        scanf("%d", &value);

        p = position_chooser(0);

        for (int i = 1; NEXT->next != NULL && i < p; i++)
        NEXT = NEXT->next;
        fvalue = NEXT->value;
        NEXT->value = value;
        printf("The element with old value %d at position %d was updated\n",
fvalue, p);
        return 1;
}
```

```
int insert(int ins_last_or_first)
{
        if (isFull())
        {
        return 0;
        }
        int k, p;
        NODE *NEXT;
        NODE *temp = (NODE *)malloc(sizeof(NODE));
        NEXT = HEAD;

        if (isNull(temp))
        printf("MEMORY OVERFLOW\n");

        printf("Enter the value\n");
        scanf("%d", &k);
        p = position_chooser(ins_last_or_first);
        temp->value = k;
        count++;
        if (p == 0)
        {
        msg("The element was inserted\n");
        temp->next = HEAD;
        HEAD = temp;
        return 1;
        }

        for (int i = 1; i < p && NEXT->next != NULL; i++)
        NEXT = NEXT->next;

        temp->next = NEXT->next;
        NEXT->next = temp;
        msg("The element was inserted\n");
        return 1;
}
int display()
```

```
{
        NODE *START = HEAD;

        if (isEmpty())
        return 0;

        printf("the elements in the list are:\n");

        while (START->next != NULL)
        {
        printf("%d\n", START->value);
        START = START->next;
        }
        printf("%d\n", START->value);
        return 1;
}
void help()
{
        msg("\n insert : 1 Remove : 2 Count : 3 isEmpty : 4 isFull : 5 insert in
first or last : 6 Remove in first or last : 7 Update : 8 Exit : 9 Display : 10\n\n");
        return;
}

int main()
{
        int input;
        help();
        while (1)
        {
        msg("\nEnter the operation to perform(for help - 0):\n");
        scanf("%d", &input);
        switch (input)
        {

        case 0:
        help();
```

```
break;
case 1:
if (!insert(0))
        msg("the value was not inserted to the list\n");
break;
case 2:
if (!Remove(0))
        msg("the node was not deleted\n");

break;
case 3:
printf("There are %d elements in the list \n", count);
break;
case 4:
if (!isEmpty())
        msg("the list is not empty\n");
break;
case 5:
if (!isFull())
        msg("The list was not full\n");
break;
case 6:
if (!insert(1))
        msg("The value was not inserted\n");
break;
case 7:
if (!Remove(1))
        msg("the node was not deleted\n");
break;
case 8:
if (!update())
        msg("The node was not updated\n");
break;
case 9:
exit(0);
case 10:
```

```
        display();
        break;

        default:
        break;
        }
        }
}
```

OUTPUT:

```
 insert : 1 Remove : 2 Count : 3 isEmpty : 4 isFull : 5 insert in first or last
: 6 Remove in first or last : 7 Update : 8 Exit : 9 Display : 10


Enter the operation to perform(for help - 0):
1
Enter the value
56
Enter the position
0
The element was inserted

Enter the operation to perform(for help - 0):
1
Enter the value
78
Enter the position
1
The element was inserted

Enter the operation to perform(for help - 0):
10
the elements in the list are:
56
78

Enter the operation to perform(for help - 0):
2
Enter the position
0
The element 56 was deleted
```

RESULT:

The program was executed successfully.

2.DOUBLY LINKED LIST

DATE : 05/02/20

AIM:

To implement storing and manipulation of data using doubly linked list.

ALGORITHM:

1. start the program.
2. declare the variables.
3. implement the functions for inserting the first element, last element and nth element, deleting the first element, last element and nth element.
4. using dynamic memory allocation to allocate memory block for node, which contains the value and the address for the next element
5. For inserting, create a memory for node and copy the value to the node, if the node is to be inserted first update nodes next to the head, previous to null and if the node is to be inserted at last update the nodes next element to null and point the next of previous element to the created node, previous to the previous element.
6. get the input from the user .
7. output the result.
8. end the program.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int count = 0;
struct node
{
	int value;
	struct node *next, *prev;
```

```
};
typedef struct node NODE;
struct node *HEAD = NULL;
int isEmpty()
{
	if (count == 0)
	{
	printf("The list is empty\n");
	return 1;
	}
	return 0;
}
int isFull()
{
	if (count == MAX)
	{
	printf("the list is full\n");
	return 1;
	}
	return 0;
}
void msg(char message[])
{
	printf("%s", message);
}
int position_chooser(int IS_FIRST_OR_LAST)
{
	int p;
	if (IS_FIRST_OR_LAST)
	{
	char choose;
	printf("FIRST = f, LAST = l\n");
	while (1)
	{
	scanf(" %c", &choose);
```

```
        switch (choose)
        {
        case 'f':
                p = 0;
                break;
        case 'l':
                p = count - 1;
                break;
        default:
                p = -1;
                break;
        }

        if (p == -1)
        {
                msg("Enter valid input, TRY AGAIN\n");

                continue;
        }

        break;
        }
        }
        else
        {
        msg("Enter the position\n");
        while (1)
        {
        scanf("%d", &p);
        if (p > count)
        {
                printf("Invalid position, there are only %d elements and position
starts from '0', TRY AGAIN\n", count);
                continue;
        }
        else
```

```
                break;
        }
        }

        return p;
}
int isNull(NODE *ptr)
{
        if (ptr == NULL)
        return 1;
        return 0;
}
int Remove(int rem_first_or_last)
{
        if (isEmpty())
        {
        return 0;
        }
        int value, p;
        NODE *NEXT, *DELETE;
        NEXT = HEAD;

        p = position_chooser(rem_first_or_last);

        count--;

        if (p == 0)
        {
        value = HEAD->value;
        DELETE = HEAD;
        HEAD = NEXT->next;
        HEAD->prev = NULL;
        free(DELETE);
        printf("The element %d was deleted", value);
        return 1;
        }
```

```
        for (int i = 1; i < p && NEXT->next != NULL; i++)
        NEXT = NEXT->next;

        DELETE = NEXT->next;
        value = DELETE->value;
        NEXT->next = DELETE->next;
        free(DELETE);
        printf("The element %d was deleted", value);
        return 1;
}
int update()
{
        NODE *NEXT;
        int p, value, fvalue;
        NEXT = HEAD;
        msg("Enter the new value\n");
        scanf("%d", &value);

        p = position_chooser(0);

        for (int i = 1; NEXT->next != NULL && i < p; i++)
        NEXT = NEXT->next;
        fvalue = NEXT->value;
        NEXT->value = value;
        printf("The element with old value %d at position %d was updated\n",
fvalue, p);
        return 1;
}
int insert(int ins_last_or_first)
{
        if (isFull())
        {
        return 0;
        }
        int k, p;
        NODE *NEXT;
```

```
        NODE *temp = (NODE *)malloc(sizeof(NODE));
        NEXT = HEAD;

        if (isNull(temp))
        printf("MEMORY OVERFLOW\n");

        printf("Enter the value\n");
        scanf("%d", &k);
        p = position_chooser(ins_last_or_first);
        temp->value = k;
        count++;
        if (p == 0)
        {
        msg("The element was inserted\n");
        temp->next = HEAD;
        temp->prev = NULL;
        HEAD->prev = temp;
        HEAD = temp;
        return 1;
        }

        for (int i = 1; i < p && NEXT->next != NULL; i++)
        NEXT = NEXT->next;

        temp->next = NEXT->next;
        temp->prev = NEXT;
        (NEXT->next)->prev = temp;
        NEXT->next = temp;
        msg("The element was inserted\n");
        return 1;
}
int display()
{
        NODE *START = HEAD;

        if (isEmpty())
```

```
        return 0;

        printf("the elements in the list are:\n");

        while (START->next != NULL)
        {
        printf("%d\n", START->value);
        START = START->next;
        }
        printf("%d\n", START->value);
        return 1;
}
void help()
{
        msg("\n insert : 1 Remove : 2 Count : 3 isEmpty : 4 isFull : 5 insert in
first or last : 6 Remove in first or last : 7 Update : 8 Exit : 9 Display : 10\n\n");
        return;
}

int main()
{
        int input;
        help();
        while (1)
        {
        msg("\nEnter the operation to perform(for help - 0):\n");
        scanf("%d", &input);
        switch (input)
        {

        case 0:
        help();
        break;
        case 1:
        if (!insert(0))
                msg("the value was not inserted to the list\n");
```

```
break;
case 2:
if (!Remove(0))
        msg("the node was not deleted\n");

break;
case 3:
printf("There are %d elements in the list \n", count);
break;
case 4:
if (!isEmpty())
        msg("the list is not empty\n");
break;
case 5:
if (!isFull())
        msg("The list was not full\n");
break;
case 6:
if (!insert(1))
        msg("The value was not inserted\n");
break;
case 7:
if (!Remove(1))
        msg("the node was not deleted\n");
break;
case 8:
if (!update())
        msg("The node was not updated\n");
break;
case 9:
exit(0);
case 10:
display();
break;

default:
```

```
        break;
        }
        }
}
```

OUTPUT:

```
 insert : 1 Remove : 2 Count : 3 isEmpty : 4 isFull : 5 insert in first or last
: 6 Remove in first or last : 7 Update : 8 Exit : 9 Display : 10 Reverse : 11 Re
verse display : 12


Enter the operation to perform(for help - 0):
1
Enter the value
21
Enter the position
0
The element was inserted

Enter the operation to perform(for help - 0):
1
Enter the value
65
Enter the position
1
The element was inserted

Enter the operation to perform(for help - 0):
10
the elements in the list are:
21
65

Enter the operation to perform(for help - 0):
2
Enter the position
0
The element 21 was deleted
```

RESULT:

The program was executed successfully.

3.CIRCULAR LINKED LIST

DATE: 05/02/20

AIM:

To storing and manipulation of data using circular linked list data structure.

ALGORITHM:

1. Start the program.
2. declare the variables.
3. create the structure for a node with info and pointer for the next node.
4. implement the functions for inserting, deleting, and displaying the elements in the linked list.
5. for creating the node use malloc function;
6. always points the last element to the head.
7. get the input from the user.
8. end the program.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int count = 0;
struct node
{
   int value;
   struct node *next;
};
typedef struct node NODE;
struct node *HEAD = NULL;
```

```
NODE *LAST = NULL;
int isEmpty()
{
   if (count == 0)
   {
      printf("The list is empty\n");
      return 1;
   }
   return 0;
}
int isFull()
{
   if (count == MAX)
   {
      printf("the list is full\n");
      return 1;
   }
   return 0;
}
void msg(char message[])
{
   printf("%s", message);
}
int position_chooser(int IS_FIRST_OR_LAST)
{
   int p;
   if (IS_FIRST_OR_LAST)
   {
      char choose;
      printf("FIRST = f, LAST = l\n");
      while (1)
      {
         scanf(" %c", &choose);

         switch (choose)
         {
```

```
        case 'f':
            p = 0;
            break;
        case 'l':
            p = count - 1;
            break;
        default:
            p = -1;
            break;
        }

        if (p == -1)
        {
            msg("Enter valid input, TRY AGAIN\n");

            continue;
        }

        break;
      }
   }
   else
   {
      msg("Enter the position\n");
      while (1)
      {
         scanf("%d", &p);
         if (p > count)
         {
            printf("Invalid position, there are only %d elements and position
starts from '0', TRY AGAIN\n", count);
            continue;
         }
         else
            break;
      }
```

```
    }

    return p;
}
int isNull(NODE *ptr)
{
    if (ptr == NULL)
        return 1;
    return 0;
}
int Remove(int rem_first_or_last)
{
    if (isEmpty())
    {
        return 0;
    }
    int value, p, i;
    NODE *NEXT, *DELETE;
    NEXT = HEAD;

    p = position_chooser(rem_first_or_last);

    count--;

    if (p == 0)
    {
        value = HEAD->value;
        DELETE = HEAD;
        HEAD = NEXT->next;
        if(LAST != NULL)
        LAST->next =HEAD;
        free(DELETE);
        printf("The element %d was deleted", value);

        return 1;
    }
```

```
   for (i = 1; i < p && NEXT->next != HEAD; i++)
      NEXT = NEXT->next;

   DELETE = NEXT->next;
   value = DELETE->value;
   if((NEXT->next)->next == HEAD)
   LAST = NEXT;
   NEXT->next = DELETE->next;
   free(DELETE);
   printf("The element %d was deleted", value);
   return 1;
}
int update()
{
   NODE *NEXT;
   int p, value, fvalue, i;
   NEXT = HEAD;
   msg("Enter the new value\n");
   scanf("%d", &value);

   p = position_chooser(0);

   for (i = 1; NEXT->next != NULL && i < p; i++)
      NEXT = NEXT->next;
   fvalue = NEXT->value;
   NEXT->value = value;
   printf("The element with old value %d at position %d was updated\n",
fvalue, p);
   return 1;
}
int insert(int ins_last_or_first)
{
   if (isFull())
   {
      return 0;
   }
```

```
int k, p, i;
NODE *NEXT;
NODE *temp = (NODE *)malloc(sizeof(NODE));
NEXT = HEAD;

if (isNull(temp))
   printf("MEMORY OVERFLOW\n");

printf("Enter the value\n");
scanf("%d", &k);
p = position_chooser(ins_last_or_first);
temp->value = k;
count++;
if (p == 0)
{

   temp->next = HEAD;
   if(LAST != NULL)
   LAST->next = temp;
   else
   LAST = temp;
   msg("The element was inserted\n");
   HEAD = temp;
   LAST->next = HEAD;
   return 1;
}

for (i = 1; i < p && NEXT->next != HEAD; i++)
   NEXT = NEXT->next;

temp->next = NEXT->next;
NEXT->next = temp;
  if(temp->next == HEAD){
     LAST = temp;
     LAST->next = HEAD;
}
```

```
     msg("The element was inserted\n");
   return 1;
}
void display()
{
   NODE *START = HEAD;

   if (isEmpty())
      return ;

   printf("the elements in the list are:\n");

  while (START->next != HEAD)
   {
      printf("%d\n", START->value);
      START = START->next;
   }
  printf("%d\n", (START->value));
   return ;
}
void help()
{
   msg("\n insert : 1 Remove : 2 Count : 3 isEmpty : 4 isFull : 5 insert in first
or last : 6 Remove in first or last : 7 Update : 8 Exit : 9 Display : 10\n\n");
   return;
}

int main()
{
   int input;
   help();
   while (1)
   {
      msg("\nEnter the operation to perform(for help - 0):\n");
      scanf("%d", &input);
```

```
switch (input)
{

case 0:
   help();
   break;
case 1:
   if (!insert(0))
      msg("the value was not inserted to the list\n");
   break;
case 2:
   if (!Remove(0))
      msg("the node was not deleted\n");

   break;
case 3:
   printf("There are %d elements in the list \n", count);
   break;
case 4:
   if (!isEmpty())
      msg("the list is not empty\n");
   break;
case 5:
   if (!isFull())
      msg("The list was not full\n");
   break;
case 6:
   if (!insert(1))
      msg("The value was not inserted\n");
   break;
case 7:
   if (!Remove(1))
      msg("the node was not deleted\n");
   break;
case 8:
   if (!update())
```

```
            msg("The node was not updated\n");
         break;
      case 9:
         exit(0);
         break;
      case 10:
         display();
         break;

      default:
         break;
      }
   }
}
```

OUTPUT:

```
 insert : 1 Remove : 2 Count : 3 isEmpty : 4 isFull :
 8 Exit : 9 Display : 10


Enter the operation to perform(for help - 0):
1
Enter the value
54
Enter the position
0
The element was inserted

Enter the operation to perform(for help - 0):
1
Enter the value
76
Enter the position
1
The element was inserted

Enter the operation to perform(for help - 0):
10
the elements in the list are:
54
76

Enter the operation to perform(for help - 0):
```

RESULT

The program was executed successfully.