|  |
| --- |
| **Class:-SY.BCS**  **Subject:-Practical On Data Structure And Algorithums.**    **Subject Code:-23123**    **Assignment No:-0 Set:-A**    **Program NO:-0 Program Name:-SINGLYI.H** |

\*Program:-

// A complete working C program to demonstrate all insertion methods

// on Linked List

#include <stdio.h>

#include <stdlib.h>

// A linked list node

struct Node

{

int data;

struct Node \*next;

};

/\* Given a reference (pointer to pointer) to the head of a list and

an int, inserts a new node on the front of the list. \*/

void insertAtFirst(struct Node\*\* head\_ref, int new\_data)

{

/\* 1. allocate node \*/

struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

/\* 2. put in the data \*/

new\_node->data = new\_data;

/\* 3. Make next of new node as head \*/

new\_node->next = (\*head\_ref);

/\* 4. move the head to point to the new node \*/

(\*head\_ref) = new\_node;

}

/\* Given a node prev\_node, insert a new node after the given

prev\_node \*/

void insertAt(struct Node\*\* head\_ref,int pos, int new\_data)

{

/\* 1. allocate node \*/

struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

struct Node\* current = \*head\_ref;

/\* used in step 5\*/

/\* 2. put in the data \*/

new\_node->data = new\_data;

/\* 3. This new node is going to be the last node, so make next of

it as NULL\*/

new\_node->next = NULL;

/\* 4. If the Linked List is empty, then make the new node as head \*/

if (\*head\_ref == NULL)

{

\*head\_ref = new\_node;

return;

}

/\* 5. Else traverse till the last node \*/

while (current->next != NULL)

{

if(pos==1)

{

new\_node->next = current->next;

/\* 6. Change the next of last node \*/

current->next = new\_node;

return;

}else

{

pos--;

current=current->next;

}

}

}

/\* Given a reference (pointer to pointer) to the head

of a list and an int, appends a new node at the end \*/

void append(struct Node\*\* head\_ref, int new\_data)

{

/\* 1. allocate node \*/

struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

struct Node \*last = \*head\_ref; /\* used in step 5\*/

/\* 2. put in the data \*/

new\_node->data = new\_data;

/\* 3. This new node is going to be the last node, so make next of

it as NULL\*/

new\_node->next = NULL;

/\* 4. If the Linked List is empty, then make the new node as head \*/

if (\*head\_ref == NULL)

{

\*head\_ref = new\_node;

return;

}

/\* 5. Else traverse till the last node \*/

while (last->next != NULL)

last = last->next;

/\* 6. Change the next of last node \*/

last->next = new\_node;

}

// Deletes the node containing 'info'

// part as val and alter the head of

// the linked list (recursive method)

void delete\_dnode(struct Node\*\* head\_ref, int del\_data)

{

/\* 1. store head node \*/

struct Node\* current = \*head\_ref;

struct Node \*prev ;

/\* 2. if head node itself holds the key to be deleted \*/

if(current !=NULL && current-> data==del\_data)

{

\*head\_ref =current ->next; // head changed to next node

free(current);

return;

}

/\*3. search fot the data to be deleted, keep track of previous node as we need to change 'prev->next'

\*/

while(current !=NULL && current ->data != del\_data)

{

prev= current;

current= current-> next;

}

/\* 4. If the del\_data not present in Linked List \*/

if (current == NULL)

return;

//5. unlink the node from linked list

prev-> next= current-> next;

free(current); //Free memory

}

// This function prints contents of linked list starting from head

void printList(struct Node \*node)

{

while (node != NULL)

{

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

node = node->next;

}

}

int search(struct Node\*\* head\_ref, int x)

{

struct Node\* current=\*head\_ref; //Initialize current

while(current != NULL)

{

if(current->data==x)

return 1;

current=current->next;

}

return 0;

}

/\* Function to reverse the linked list \*/

static void reverse(struct Node\*\* head\_ref)

{

struct Node\* prev = NULL;

struct Node\* current = \*head\_ref;

struct Node\* next = NULL;

while (current != NULL) {

// Store next

next = current->next;

// Reverse current node's pointer

current->next = prev;

// Move pointers one position ahead.

prev = current;

current = next;

}

\*head\_ref = prev;

}