**Q10. Design and implement a dynamic (doubly linked list) to store** **any information which needs a linear data structure.**

**Aim:** To implement a doubly linked list.

**Algorithm:**

**Insertion**

* Step 1: IF ptr = NULL

Write OVERFLOW

Go to Step 9

[END OF IF]

* Step 2: SET NEW\_NODE = ptr
* Step 3: SET ptr = ptr -> NEXT
* Step 4: SET NEW\_NODE -> DATA = VAL
* Step 5: SET NEW\_NODE -> PREV = NULL
* Step 6: SET NEW\_NODE -> NEXT = START
* Step 7: SET head -> PREV = NEW\_NODE
* Step 8: SET head = NEW\_NODE
* Step 9: EXIT

**Deletion**

* STEP 1: IF HEAD = NULL

WRITE UNDERFLOW

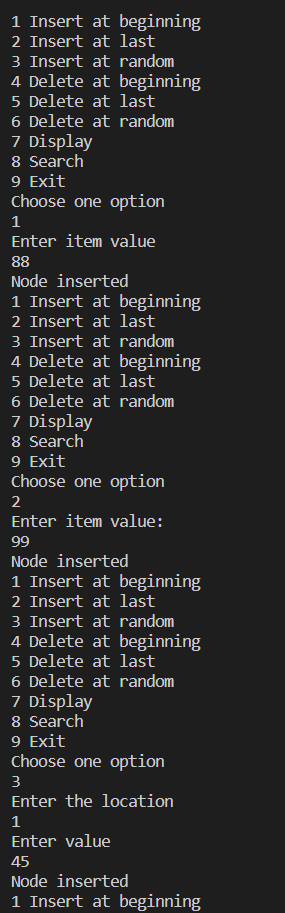
GO TO STEP 6

* STEP 2: SET PTR = HEAD
* STEP 3: SET HEAD = HEAD → NEXT
* STEP 4: SET HEAD → PREV = NULL
* STEP 5: FREE PTR
* STEP 6: EXIT

**Program:**

|  |
| --- |
| // Menu driven program of doubly linked list |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  |  |
|  | struct node |
|  | { |
|  | struct node \*prev; |
|  | struct node \*next; |
|  | int data; |
|  | }; |
|  | struct node \*head; |
|  |  |
|  | void insert\_begun(); |
|  | void insert\_last(); |
|  | void insert\_specified(); |
|  | void delete\_begun(); |
|  | void delete\_last(); |
|  | void delete\_random(); |
|  | void display(); |
|  | void search(); |
|  |  |
|  | void main() |
|  | { |
|  | int choice = 0; |
|  | while (choice != 9) |
|  | { |
|  | printf("1 Insert at beginning\n2 Insert at last\n3 Insert at random\n4 Delete at beginning\n5 Delete at last\n6 Delete at random\n7 Display\n8 Search\n9 Exit\n"); |
|  | printf("Choose one option\n"); |
|  | scanf("%d", &choice); |
|  |  |
|  | switch (choice) |
|  | { |
|  | case 1: |
|  | insert\_begun(); |
|  | break; |
|  | case 2: |
|  | insert\_last(); |
|  | break; |
|  | case 3: |
|  | insert\_specified(); |
|  | break; |
|  | case 4: |
|  | delete\_begun(); |
|  | break; |
|  | case 5: |
|  | delete\_last(); |
|  | break; |
|  | case 6: |
|  | delete\_random(); |
|  | break; |
|  | case 7: |
|  | display(); |
|  | break; |
|  | case 8: |
|  | search(); |
|  | break; |
|  | case 9: |
|  | exit(0); |
|  | break; |
|  | default: |
|  | printf("Invalid choice\n"); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void insert\_begun() |
|  | { |
|  | struct node \*ptr; |
|  | int item; |
|  | ptr = (struct node \*)malloc(sizeof(struct node)); |
|  | if (ptr == NULL) |
|  | printf("Overflow\n"); |
|  |  |
|  | else |
|  | { |
|  | printf("Enter item value\n"); |
|  | scanf("%d", &item); |
|  | if (head == NULL) |
|  | { |
|  | ptr->next = NULL; |
|  | ptr->prev = NULL; |
|  | ptr->data = item; |
|  | head = ptr; |
|  | } |
|  |  |
|  | else |
|  | { |
|  | ptr->data = item; |
|  | ptr->prev = NULL; |
|  | ptr->next = head; |
|  | head->prev = ptr; |
|  | head = ptr; |
|  | } |
|  | printf("Node inserted\n"); |
|  | } |
|  | } |
|  |  |
|  | void insert\_last() |
|  | { |
|  | struct node \*ptr, \*temp; |
|  | int item; |
|  | ptr = (struct node \*)malloc(sizeof(struct node)); |
|  | if (ptr == NULL) |
|  | printf("Overflow\n"); |
|  |  |
|  | else |
|  | { |
|  | printf("Enter item value:\n"); |
|  | scanf("%d", &item); |
|  | ptr->data = item; |
|  | if (head == NULL) |
|  | { |
|  | ptr->next = NULL; |
|  | ptr->prev = NULL; |
|  | head = ptr; |
|  | } |
|  |  |
|  | else |
|  | { |
|  | temp = head; |
|  | while (temp->next != NULL) |
|  | { |
|  | temp = temp->next; |
|  | } |
|  | temp->next = ptr; |
|  | ptr->prev = temp; |
|  | ptr->next = NULL; |
|  | } |
|  |  |
|  | printf("Node inserted\n"); |
|  | } |
|  | } |
|  |  |
|  | void insert\_specified() |
|  | { |
|  | struct node \*ptr, \*temp; |
|  | int item, loc, i; |
|  | ptr = (struct node \*)malloc(sizeof(struct node)); |
|  | if (ptr == NULL) |
|  | printf("Overflow\n"); |
|  |  |
|  | else |
|  | { |
|  | temp = head; |
|  | printf("Enter the location\n"); |
|  | scanf("%d", &loc); |
|  | for (i = 0; i < loc; i++) |
|  | { |
|  | temp = temp->next; |
|  | if (temp == NULL) |
|  | { |
|  | printf("Less than %d elements\n", loc); |
|  | return; |
|  | } |
|  | } |
|  |  |
|  | printf("Enter value\n"); |
|  | scanf("%d", &item); |
|  | ptr->data = item; |
|  | ptr->next = temp->next; |
|  | ptr->prev = temp; |
|  | temp->next = ptr; |
|  | temp->next->prev = ptr; |
|  | printf("Node inserted\n"); |
|  | } |
|  | } |
|  |  |
|  | void delete\_begun() |
|  | { |
|  | struct node \*ptr; |
|  | if (head == NULL) |
|  | printf("Underflow\n"); |
|  |  |
|  | else if (head->next == NULL) |
|  | { |
|  | head = NULL; |
|  | free(head); |
|  | printf("Node deleted\n"); |
|  | } |
|  |  |
|  | else |
|  | { |
|  | ptr = head; |
|  | head = head->next; |
|  | head->prev = NULL; |
|  | free(ptr); |
|  | printf("Node deleted\n"); |
|  | } |
|  | } |
|  |  |
|  | void delete\_last() |
|  | { |
|  | struct node \*ptr; |
|  | if (head == NULL) |
|  | printf("Underflow\n"); |
|  |  |
|  | else if (head->next == NULL) |
|  | { |
|  | head = NULL; |
|  | free(head); |
|  | printf("Node deleted\n"); |
|  | } |
|  |  |
|  | else |
|  | { |
|  | ptr = head; |
|  | if (ptr->next != NULL) |
|  | ptr = ptr->next; |
|  |  |
|  | ptr->prev->next = NULL; |
|  | free(ptr); |
|  | printf("Node delted\n"); |
|  | } |
|  | } |
|  |  |
|  | void delete\_random() |
|  | { |
|  | struct node \*ptr, \*temp; |
|  | int val; |
|  | printf("Enter the data after which the node is to be deleted\n"); |
|  | scanf("%d", &val); |
|  | ptr = head; |
|  | while (ptr->data != val) |
|  | ptr = ptr->next; |
|  |  |
|  | if (ptr->next == NULL) |
|  | printf("Can't delete\n"); |
|  |  |
|  | else if (ptr->next->next == NULL) |
|  | ptr->next = NULL; |
|  |  |
|  | else |
|  | { |
|  | temp = ptr->next; |
|  | ptr->next = temp->next; |
|  | temp->next->prev = ptr; |
|  | free(temp); |
|  | printf("Node deleted\n"); |
|  | } |
|  | } |
|  |  |
|  | void display() |
|  | { |
|  | struct node \*ptr; |
|  | printf("Printing values\n"); |
|  | ptr = head; |
|  |  |
|  | while (ptr != NULL) |
|  | { |
|  | printf("%d\n", ptr->data); |
|  | ptr = ptr->next; |
|  | } |
|  | } |
|  |  |
|  | void search() |
|  | { |
|  | struct node \*ptr; |
|  | int item, i = 0, flag; |
|  | ptr = head; |
|  | if (ptr == NULL) |
|  | printf("Empty list\n"); |
|  |  |
|  | else |
|  | { |
|  | printf("Enter item to search\n"); |
|  | scanf("%d", &item); |
|  | while (ptr != NULL) |
|  | { |
|  | if (ptr->data == item) |
|  | { |
|  | printf("Item found at location %d\n", i + 1); |
|  | flag = 0; |
|  | break; |
|  | } |
|  |  |
|  | else |
|  | flag = 1; |
|  |  |
|  | i++; |
|  | ptr = ptr->next; |
|  | } |
|  | if (flag == 1) |
|  | printf("Item not found\n"); |
|  | } |
|  | } |

**Output:**

****