

program 6: Linked list application

Date \_\_\_\_\_

Page \_\_\_\_\_

pseudo  
code:

```
void insertAtBeginning(int Data) {  
    struct Node *newNode = (struct Node*) malloc  
        ( sizeof( struct Node ) );  
    newNode->data = data;  
    newNode->next = head;  
    head = newNode;  
    printf ("Node inserted at the beginning");  
}
```

void createLinkedList

```
void insertAtEnd(int Data) {  
    struct Node *newNode = (struct Node*) malloc  
        ( sizeof( struct Node ) );  
    newNode->data = data;  
    newNode->next = NULL;  
    if ( head->next == NULL ) {  
        head = newNode;  
    } else {  
        struct Node *temp = head;  
        while ( temp->next != NULL ) {  
            temp = temp->next;  
        }  
        temp->next = newNode;  
    }  
    printf ("Node inserted at end");  
}
```

```

void insertAtPosition(int Data, int pos) {
    int i;
    struct Node *newNode, *temp = head;
    if (pos < 1) {
        printf("wrong position");
        return;
    }
    if (pos == 1)
        insertAtBeginning(data);
    else {
        newNode = (struct Node *) malloc(sizeof(struct Node));
        newNode->data = data;
        for (i = 1; i < pos - 1; if (temp != NULL, i++)) {
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("position out of range");
        } else {
            newNode->next = temp->next;
            temp->next = newNode;
        }
    }
}

void displayList() {
    while (temp != NULL) {
        printf("%d → ", temp->data);
        temp = temp->next;
    }
}

void createList(int n) {
    int i;
    struct Node *newNode, *head = NULL;
    for (i = 1; i <= n; i++) {
        newNode = (struct Node *) malloc(sizeof(struct Node));
        newNode->data = i;
        if (head == NULL) {
            head = newNode;
        } else {
            newNode->next = head;
            head = newNode;
        }
    }
}

```

```

if (newNode == NULL), i?
scanf ("%d", &data) + line
newNode -> data = data; + line
newNode -> next = NULL;
if (head == NULL)
    head = newNode
} else {
    temp -> next = newNode
}
temp = newNode;
printf("linked list created");
}

```

~~Code B~~  
~~Code A~~

```

code:
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int data
    struct Node* next;
};
if (head == NULL) = obaltion * chalt -&+ biov
((obaltion -&+ biov))
struct Node * head = NULL;
void createList(int n) {
    struct Node * newNode, * temp;
    int data, i;
    if (n < 0){
        printf("Number of nodes should be greater than 0");
        return;
    }
    for (i=1, i<=n, i++) {
        newNode = (struct Node*) malloc (sizeof (struct Node));
        newNode -> data = data;
        newNode -> next = NULL;
        if (head == NULL)
            head = newNode;
        else {
            temp -> next = newNode;
        }
        temp = newNode;
    }
}

```

```

if (newNode == NULL) {
    printf("Memory allocation failed\n");
    return;
}

printf("Enter data for node %d: ", i);
scanf("%d", &data);

newNode->data = data;
newNode->next = NULL;

if (head == NULL) {
    head = newNode;
} else {
    temp->next = newNode;
    temp = newNode;
}

printf("In Linked list (%d)", data);
}

```

```

void insertAtBeginning(int data) {
    struct Node *newNode = (struct Node *)malloc
        (sizeof(struct Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
    printf("node inserted.\n");
}

void insertAtEnd(int data) {
    struct Node *newNode = (struct Node *)malloc
        (sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
}

```

```

if (head == NULL) head = newNode;           b100
else {
    struct Node *temp = head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}

printf ("Node inserted at end");

void insertAtPosition(int data, int pos)
{
    int i;
    struct Node *newNode, *temp = head;
    if (pos < 1) {
        printf ("Invalid position\n");
        return;
    }
    if (pos == 1) { // inserting at beginning
        newNode = (struct Node *) malloc (sizeof (struct
            Node));
        newNode->data = data;
        for (i=1; i<pos-1 && temp != NULL; i++)
            temp = temp->next;
        if (temp == NULL) {
            printf ("position out of range");
            free (newNode);
        } else {
            (newNode->next = temp->next
            temp->next = newNode;
            printf ("Inserted at position"));
        }
    }
}

```

```

void displaylist(){
    struct Node *temp = head;
    if (head == NULL) {
        printf("List is empty");
        return;
    }
    printf("\nLinked list: ");
    while (temp != NULL) {
        printf("%d → ", temp->data);
        temp = temp->next;
    }
    printf("NULL");
}

int main() {
    int choice, n, data, pos;
    printf("(1) Create (2) Insert At beginning\n"
           "(3) insert at any position (4)\n"
           "(4) Insert at end (5) display (6) exit");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    while (1) {
        switch (choice) {
            case 1: {
                cout << "Enter number of nodes: ";
                cin >> n;
                createList(n);
            }
            case 2: {
                cout << "Enter data to insert ";
                cin >> data;
                insertAtBeginning(data);
            }
            case 3: {
                cout << "Enter position: ";
                cin >> pos;
                cout << "Enter data: ";
                cin >> data;
                insertAtPosition(pos, data);
            }
            case 4: {
                cout << "Enter data: ";
                cin >> data;
                insertAtEnd(data);
            }
            case 5: display();
            case 6: break;
        }
    }
}

```

case 3 :

```
printf("enter data and position:");
scanf ("%d %d", &data, &pos);
insertAtPosition(data, pos); break;
```

case 4 :

```
printf("enter data to insert:");
scanf ("%d", &data);
insertAtEnd(data); break;
```

case 5 :

```
displayList();
break;
```

case 6 :

```
printf("Exiting\n");
exit(0);
```

default :

```
printf("invalid choice");
```

~~3~~ prompt to press (3  
~~scanf printf ("enter choice");~~ press to press (3  
~~scanf ("%d", &choice);~~ to press (A  
~~3~~ to (A  
~~3~~ to (A  
~~3~~ to (A

output --- singly linked list operations

- 1) created linked list
- 2) insert at beginning
- 3) insert at any position
- 4) insert at end
- 5) display list
- 6) Exit

Enter your choice 1 2 3 4 5 6 7 8 9 10 11 12 13

Enter number of nodes 1 2 3 4 5 6 7 8 9 10 11 12 13

Enter data for node 1: 1

Enter data for node 2: 2

Linked list created

Enter your choice: 2

Enter data to insert 0

Node inserted at No beginning

Enter your choice 4

Enter data to insert 4

Enter your choice 3

Enter data and position 4

Enter your choice: 5

Linked list: 0 → 1 → 2 → 3 → 4 → NULL

Enter your choice 6

Exiting

Tracing -- Singly linked list operations --

1) Create Linked list

2) insert at beginning

3) insert at any position

4) insert at end

5) display list

6) Exit

Enter your choice : 1

Enter number of nodes

Enter data for node 1: 100

Enter data for node 2: 200

Enter data for node 3: 300

Linked list created

Enter your choice 3

Enter data & position: 250 3

Enter your choice 4:

Enter data to insert 200

Node inserted at the end

Enter your choice: 5

Linked list: 100 → 200, 1 → 250, 2 → 300, 3 → 150, 4 → NULL

enter your choice: 6 = 0 = menu list

Exiting program \*shortest route

~~8 Feb 25~~

~~2023-02-11 11:11:25~~

\* returns

(choose option = quit)

next

{

c) return = sibling tail

bread - quit

7(i+1): sibling > i; o = i do not

; choose quit = quit

{

i quit routes

{

shortest tail before longest (s)

parent tail, etc.) shortest before \*shortest route?

? (inv tail, bread)

parent, shortest route

; bread = front, parent

parent = domain \* shortest route

? (max of front < times) slider